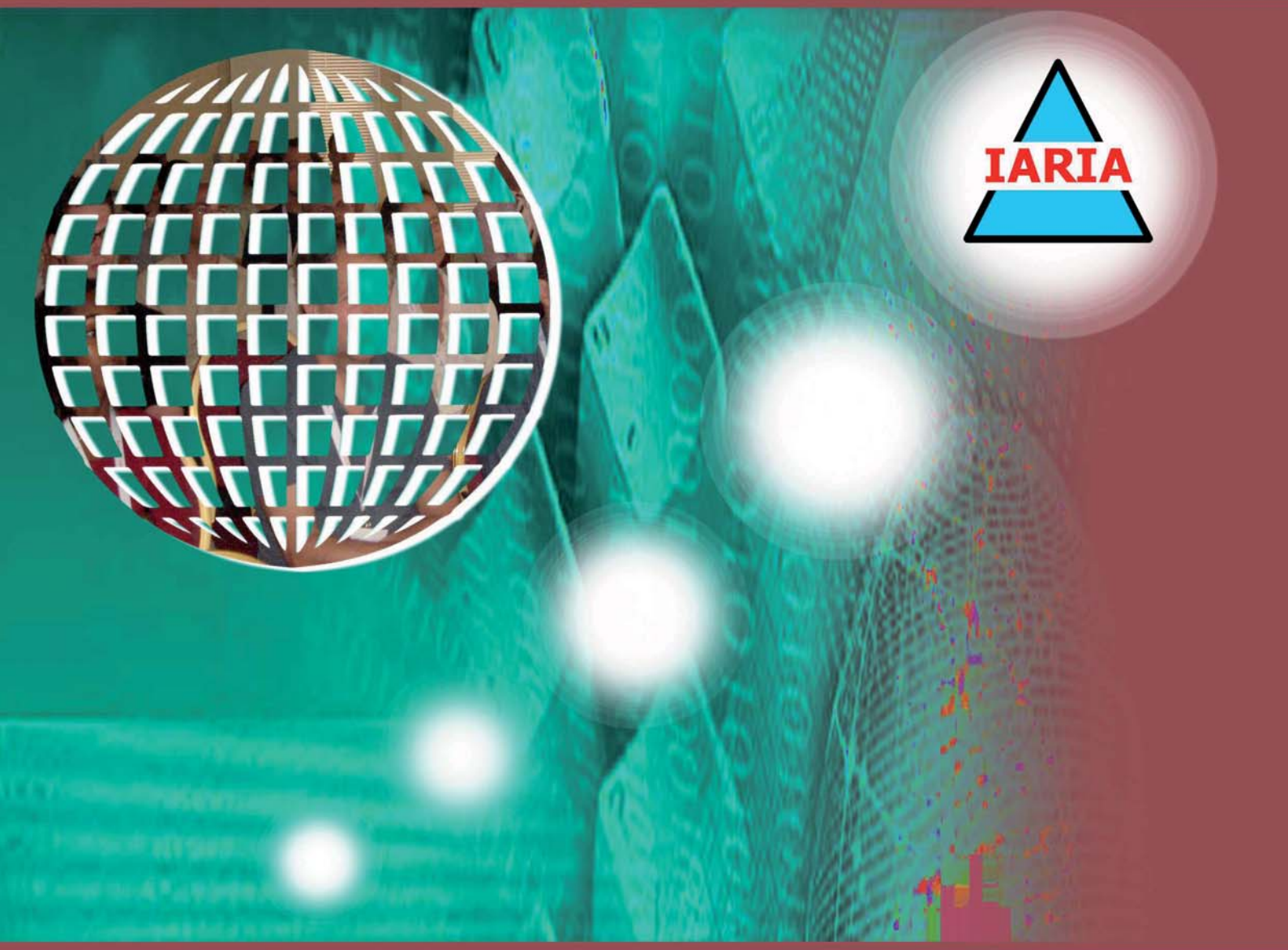
International Journal on Advances in Software



2010 vol. 3 nr. 1&2

The *International Journal on Advances in Software* is published by IARIA.
ISSN: 1942-2628
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal on Advances in Software
Volume 3, Number 1 & 2, 2010

- Olga Ormandjieva, Concordia University-Montreal, Canada
- Constantin Paleologu, University 'Politehnica' of Bucharest, Romania
- Liviu Panait, Google Inc., USA
- Kenji Saito, Keio University, Japan
- Ashok Sharma, Satyam Computer Services Ltd – Hyderabad, India
- Marcin Solarski, IBM-Software Labs, Germany

Advanced Computing

- Matthieu Geist, Supelec / ArcelorMittal, France
- Jameleddine Hassine, Cisco Systems, Inc., Canada
- Sascha Opletal, Universitat Stuttgart, Germany
- Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
- Meikel Poess, Oracle, USA
- Kurt Rohloff, BBN Technologies, USA
- Said Tazi, LAAS-CNRS, Universite de Toulouse / Universite Toulouse1, France
- Simon Tsang, Telcordia Technologies, Inc. - Piscataway, USA

Geographic Information Systems

- Christophe Claramunt, Naval Academy Research Institute, France
- Dumitru Roman, Semantic Technology Institute Innsbruck, Austria
- Emmanuel Stefanakis, Harokopio University, Greece

Databases and Data

- Peter Baumann, Jacobs University Bremen / Rasdaman GmbH Bremen, Germany
- Qiming Chen, HP Labs – Palo Alto, USA
- Ela Hunt, University of Strathclyde - Glasgow, UK
- Claudia Roncancio INPG / ENSIMAG - Grenoble, France

Intensive Applications

- Fernando Boronat, Integrated Management Coastal Research Institute, Spain
- Chih-Cheng Hung, Southern Polytechnic State University, USA
- Jianhua Ma, Hosei University, Japan
- Milena Radenkovic, University of Nottingham, UK
- DJamel H. Sadok, Universidade Federal de Pernambuco, Brazil
- Marius Slavescu, IBM Toronto Lab, Canada
- Cristian Ungureanu, NEC Labs America - Princeton, USA

Testing and Validation

- Michael Browne, IBM, USA
- Cecilia Metra, DEIS-ARCES-University of Bologna, Italy
- Krzysztof Rogoz, Motorola, USA
- Sergio Soares, Federal University of Pernambuco, Brazil

- Alin Stefanescu, University of Pitesti, Romania
- Massimo Tivoli, Universita degli Studi dell'Aquila, Italy

Simulations

- Robert de Souza, The Logistics Institute - Asia Pacific, Singapore
- Ann Dunkin, Hewlett-Packard, USA
- Tejas R. Gandhi, Virtua Health-Marlton, USA
- Lars Moench, University of Hagen, Germany
- Michael J. North, Argonne National Laboratory, USA
- Michal Pioro, Warsaw University of Technology, Poland and Lund University, Sweden
- Edward Williams, PMC-Dearborn, USA

Additional reviews

- Alexandre Mota, UFPE, Brazil

International Journal on Advances in Software
Volume 3, Numbers 1 & 2, 2010

CONTENTS

Sebastian Abeck, Karlsruhe Institute of Technology, Germany

Combining Formal Methods and MDE Techniques for Model-driven System Design and Analysis

AngeloGargantini*, Elvinia Riccobene,† and Patrizia Scandurra*
*Dipartimento di Ingegneria dell'Informazione e Metodi Matematici (DIIMM)
Università di Bergamo, Viale Marconi, 5 - 24044 Dalmine (BG), Italy
Email: {angelo.gargantini,patrizia.scandurra}@unibg.it
†Dipartimento di Tecnologie dell'Informazione (DTI)
Università degli Studi di Milano, via Bramante 65 - 26013 Crema (CR), Italy
E-mail: elvinia.riccobene@dti.unimi.it

***Abstract*—The use of *formal methods*, based on rigorous mathematical foundations, is essential for system specification and proof, especially for safety critical systems. On the other hand, *Model-driven Engineering* (MDE) is emerging as new approach to software development based on the systematic use of models as primary artifacts throughout the engineering life-cycle by combining domain-specific modeling languages (DSMLs) with model transformers, analyzers, and generators. This paper presents our position and experience on combining flexibility and automation of the MDE approach with rigorousness and preciseness of formal methods to achieve significant boosts in both productivity and quality in model-driven design and analysis of software and systems. An *in-the-loop integration* is proposed where, on one hand, MDE principles are used to engineer a language and a toolset around a formal method for its practical adoption in systems development life cycle, and, on the other hand, the same formal method is used in the same MDE context to endow modeling languages with a precise and (possibly) executable semantics and to perform formal analysis of systems models written in those languages. A concrete scenario of in-the-loop integration is presented in terms of the Abstract State Machine formal method and the Eclipse Modeling Framework. This integration allows system design using the Eclipse Modeling Framework and formal system analysis by Abstract State Machines in a seamless and systematic way, as shown by a concrete case study.**

***Keywords*-Formal methods; Model Driven Engineering; Abstract State Machines; model semantics; model execution and analysis**

I. Introduction

Using *Formal Methods* (FMs), which have rigorous mathematical foundations, for system development is nowadays extremely important, especially for high-integrity systems where safety or security need to be formally proved. On the other hand, the *Model-driven Engineering* (MDE) [2], [3] is emerging as a new paradigm in software engineering, which bases system development on (meta-)modeling and model transformations, and provides methods to build bridges between similar or different technical spaces and domains.

Both approaches have advantages and disadvantages that we here shortly summarize (see Fig. 1).

This paper is the extended version of the conference paper [1].

This work is supported in part by the Italian Government under the project PRIN 2007 *D-ASAP: Architetture Software Adattabili e Affidabili per Sistemi Pervasivi* (2007XKEHFA).

Advantages of FMs The use of formal methods in system engineering is becoming essential, especially during the early phases of the development process. Indeed, an abstract model of the system can be used to understand if the system under development satisfies the given requirements (by simulation and model-based testing), and guarantees certain properties by formal analysis (validation & verification).

Disadvantages of FMs While there are several cases proving the applicability of formal methods in industrial applications and showing very good results, many practitioners are still reluctant to adopt formal methods. Besides the well-known lack of training, this skepticism is mainly due to: the complex notations that formal techniques use rather than other lightweight and more intuitive graphical notations, like the Unified Modeling Language (UML) [4]; the lack of easy-to-use tools supporting a developer during the life cycle activities of system development, possibly in a seamless manner; and the lack of integration among formal methods themselves and their associated tools.

Advantages of MDE MDE technologies with a greater focus on architecture and automation yield higher levels of abstraction in system development by promoting models as first-class artifacts to maintain, analyze, simulate, and eventually transform into into code or into other models. Meta-modeling is a key concept of the MDE paradigm and it is intended as a way to endow a language or a formalism with an abstract notation, so separating the abstract syntax

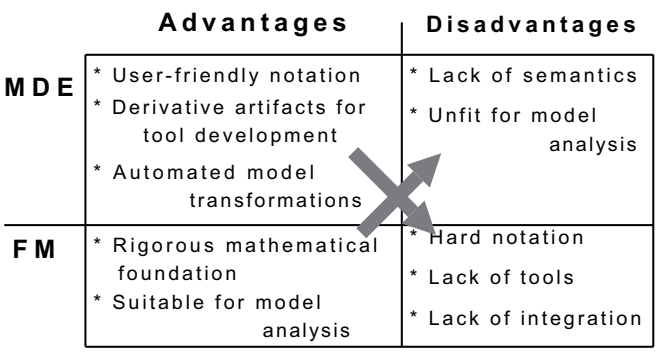


Fig. 1: Formal methods and MDE

and semantics of the language from its different concrete notations. Although the foundation constituents of the MDE are still evolving, some MDE principles are implemented in meta- modeling/programming frameworks like the OMG MDA (Model Driven Architecture) [5], Model-integrated Computing (MIC) [6], Software Factories and Microsoft Domain-Specific Languages (DSLs) tools (as part of the Visual Studio SDK) [7], Eclipse/EMF [8], etc. Metamodel-based modeling languages are increasingly being defined and adopted for specific domains of interest addressing the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively [3].

Disadvantages of MDE Although the definition of a language abstract syntax by a metamodel is well mastered and supported by many meta-modeling environments (EMF/Ecore, GME/MetaGME, AMMA/KM3, XMF-Mosaic/Xcore, etc.), the semantics definition of this class of languages is an open and crucial issue. Currently, metamodeling environments are able to cope with most syntactic and transformation definition issues, but they lack of any standard and rigorous support to provide the (possibly executable) semantics of metamodels, which is usually given in natural language. This implies that most currently adopted metamodel-based languages (such as the UML) are not yet suitable for effective model analysis due to their lack of a strong semantics necessary for a formal model analysis assisted by tools.

In [1], we described how these two approaches can be combined showing how the advantages of one can be exploited to cover or weaken the disadvantages of the other. In this paper, we extend and deepen this combination view with the final goal of developing a model-driven approach for designing systems according to the MDE principles, and analyzing models by exploiting formal techniques.

Section II provides some related work concerning connections between formal methods and MDE.

Section III describes an overall process, based on the MDE approach, for engineering a language and a tool-set for a formal method. This allows to overcome the lack of user-friendly notations, of integration of techniques, and of their tool inter-operability. This deficiency still poses a significant challenge for formal methods.

Section IV presents an approach to endow language metamodels with precise executable semantics, and we discuss techniques for formal analysis that can be used once formal models are associated to language terminal models by, possibly, automatic model mapping. This addresses the problem of expressing semantics of metamodel-based languages and performing model validation and formal verification.

In order to combine in a tight way rigorousness and preciseness of FMs with flexibility and automation of the MDE, in Section V an *in-the-loop* integration is proposed, where the same MDE technology and FM techniques are involved in both the two activities: MDE for FMs and FMs for MDE.

Section VI provides basic concepts concerning the Abstract State Machine formal method which is later used to implement the in-the-loop approach.

Sections VII and VIII show a concrete scenario of in-the-loop integration between the ASM formal method and the EMF framework. On one side, we report our experience in exploiting MDE methodology to engineer a language and a tool-set for the ASMs in order to support their practical use in systems development life cycle. On the other side, we show how ASMs can be used to provide semantics to languages defined in the MDE context and how to perform formal analysis of models developed by MDE technology.

A complete case study is presented in Section IX which shows how MDE-based technologies are used to define a metamodel-based language for the Tic-Tac-Toe, and the ASM-based semantic framework is used to define an executable semantics of the language and to support semantics validation and formal verification of models.

Section X shows how to get a tighter integration between ASM and EMF by *closing the loop*, i.e. by using the ASM formal method itself to define the semantics of the ASMs in the EMF framework.

Finally, our conclusion and future directions are provided in Section XI.

II. Related work

Software languages play a cornerstone role in system development. Language engineering processes have been considered in many contexts of software engineering [9]. Concerning the metamodeling technique of MDE for (software) language engineering, many proposals have been presented, which pay attention to the fact that language descriptions take different form in different technical spaces (e.g. metamodels, schemas, grammars, and ontologies) and typically multiple languages (from different technical spaces) need to be used together and integrated in most software development scenarios. A process to engineer languages address several aspects of a language: structure, constraints, textual and graphical representation, parser/compiler, transformational and executional behavior. Research usually faced only one of these aspects, therefore, a comparison with related work can be often done considering single aspects of a language development process.

Formal methods communities have only recently started to settle their tools on metamodels and MDE platforms. A non exhaustive list of such efforts follows. An Event-B metamodel and an EMF-based Framework for Event-B have been recently developed [10] to provide an EMF-based front-end to the Rodin platform, an Eclipse-based IDE for Event-B that provides support for refinement and mathematical proof of Event-B models.

The Maude Formal Tool Environment [11] is an executable rewriting logic language suited for the specification of object-oriented open and distributed systems. It offers tool support for reasoning about Maude specifications and, recently, also an Eclipse plug-in that allows to connect the Maude environment to the KM3 metamodeling framework using ATL (the ATLAS Transformation Language) [12] transformations.

Within the Graph Transformation community, using the concepts of graph transformations and metamodeling, the transformation language GReAT (Graph Rewriting And Transformation language) [13] has been designed to address the

specific needs of the model transformation area of the Model Integrated Computing. It is supported by tools that allow the rapid prototyping and realization of transformation tools.

To the best of our knowledge, the development of the above mentioned languages and tools did not follow a model-driven engineering process like the one described here in Section III.

A metamodel for the ITU language SDL-2000 has been also developed [14]. The authors present also a semi-automatic *reverse engineering* methodology that allows the derivation of a metamodel from a formal syntax definition of an existing language. The SDL metamodel has been derived from the SDL grammar using this methodology. A very similar method to bridge *grammarware* and *modelware* is also proposed by other authors in [15] and in [16]. These approaches are complementary to the development process presented in Sect. III. Our approach has to be considered a *forward engineering* process consisting in deriving a concrete textual notation from an abstract metamodel.

A recent result [17] shows how to apply metamodel-based technologies for the creation of a language description for Sudoku. This is on the same line of our approach of exploiting MDE technologies to develop a tool-set around ASMs.

Within the ASM community, a number of notations and tools have been developed for the specification and analysis [18]. The Abstract State Machine Language (AsmL) developed by the Foundation Software Engineering group at Microsoft is the greatest effort. AsmL is a rich executable specification language, based on the theory of Abstract State Machines, expression- and object- oriented, and fully integrated into the Microsoft .NET framework. However, AsmL does not provide a semantic structure targeted for the ASM method. "One can see it as a fusion of the Abstract State Machine paradigm and the .NET type system, influenced to an extent by other specification languages like VDM or Z" [19]. Adopting a terminology currently used, AsmL is a platform-specific modeling language for the .NET type system. Of the remaining tools for ASMs, let us mention the more popular ones: the CoreASM, an extensible execution engine developed in Java, TASM (Timed ASMs), an encoding of Timed Automata in ASMs, and a simulator-model checker for reactive real-time ASMs [20] able to specify and verify First Order Timed Logic (FOTL) properties on ASM models. Among these, the CoreASM engine is the more comparable to our. Other specific languages for the ASMs, no longer maintained, are ASM-SL, which adopts a functional style being developed in ML and which has inspired us in the language of terms, the AsmGofer language based on the Gofer environment, and XASM which is integrated in Montages, an environment generally used for defining semantics and grammar of programming languages. All the above tools, however, do not rely on MDE principles and techniques, and, except CoreASM that is based on an extensible architecture, none of the others are designed to support model exchange and tool integration. Recently, a metamodel for the AsmL language is available as part of a zoo of metamodels defined by using the KM3 meta-language. However, this metamodel is not appropriately documented or described elsewhere, so this prevented us to evaluate it.

Regarding the derivation of concrete grammars for metamodels, developing a grammar for the ASMs from the metamodel was challenging and led us to the definition of a bridge between grammars and metamodels as explained in [21]. This part of the process required at least six man month. Although we did not automatize these rules, because no advanced model-to-text tools were available at that time and because we wanted to derive only one grammar for AsmetaL, the rules may be easily reused for other formalisms. Several model-to-text tools exist now: EMFText [22] working for Ecore metamodels, TCS [23] (Textual Concrete Syntax) for metamodels written in KM3, TEF (Textual Editing Framework) for EMF-based metamodels, etc. Vice versa, Xtext [24] allows to derive a language metamodel from the language concrete textual grammar. An overview of textual grammars and metamodel is given in [25]. Other more complex model-to-text tools, capable of generating text grammars from specific MOF based repositories, exist [26], [27]. These tools render the content of a MOF-based repository (known as a MOFlet) in textual form, conforming to some syntactic rules (grammar). However, though automatic, since they are designed to work with any MOF model and generate their target grammar based on predefined patterns thus they do not permit a detailed customization of the generated language.

On the problem of integrating graphical notations and formal methods, [28] shows how the process algebra CSP and the specification language Object-Z, can be integrated into an object-oriented software engineering process employing the UML as a modeling and Java as an implementation language. In [29], the author presents an approach to formal methods technology exploitation which introduces formal notations into critical systems development processes. Furthermore, [30] proposes a metamodel-based transformation technique, which is founded by a set of structural and semantic mappings between UML and B, to assist derivation of formal B specifications from UML diagrams. All these approaches are based on translating graphical models to formal specifications, and are similar to our approach on moving from terminal models of a metamodel-based language to an ASM specification. However, they are tailored for the UML, while our approach refer to generic metamodel-based languages, and they perform only one side of the in-the-loop integration.

An MDE-based approach for integrating different formal methods was recently proposed in [31]. As in our approach, formal models are introduced into MDE as domain specific languages by developing their metamodels. Then, transformation rules are defined to obtain notation bridges. At last, model-to-text syntax rules are developed, so to map models into programs. As case study, the approach was applied for bridging MARTE to LOTOS. The main goal of their work is to integrate different formal notations in software development, however they do not provide semantics to them. General challenges of tool integration are discussed in [32], where a software language engineering solution technique is presented that apply MDE principles to address tool interoperability.

Concerning the problem of specifying the semantics of metamodel-based languages, some recent works, such as Kermeta [33], aim at providing executability into current metamodeling frameworks. Another effort toward this same

direction is presented in [34] where the authors describe the M3Actions framework to support operational semantics for EMF models. The Maude formalism is also proposed in [35] as a way for specifying the semantics of visual modeling languages.

On the application of ASMs for specifying the execution semantics of metamodel-based languages in a MDE style, we can mention the translational approach described in [36]. They propose a *semantic anchoring* to well-established formal models of computation (such as FSMs, data flow, and discrete event systems) built upon AsmL, by using the transformation language GME/GReAT. The proposed approach offers up predefined and well-defined sets of *semantic units* for future (conventional) anchoring efforts. However, we see two main disadvantages in this approach: first, it requires well understood and safe behavioral language units and it is not clear how to specify the language semantics from scratch when these language units do not yet exist; second, in *heterogeneous systems*, specifying the language semantics as composition of some selected primary semantic units for basic behavioral categories [37] is not always possible, since there may exist complex behaviors which are not easily reducible to a combination of existing ones. Still concerning the translational category, in [38] the dynamic semantics of the AMMA/ATL transformation language was specified in the XASM [39] ASM dialect. A direct mapping from the AMMA meta-language KM3 to an XASM metamodel is used to represent metamodels in terms of ASM universes and functions, and this ASM model is taken as basis for the dynamic semantics specification of the ATL metamodel. However, this mapping is neither formally defined nor the ATL transformation code which implements it have been made available in the ATL transformations Zoo or as ATL use case [12]; only the Atlantic XASM Zoo [40], a mirror of the Atlantic Zoo metamodels expressed in XASM (as a collection of universes and functions), has been made available. A further recent result [41] proposes ASMs, Prolog, and Scheme as description languages in a framework named EProvide 2.0 for prototyping the operational semantics of metamodel-based languages. Their approach is also translational as it is based on three bridges: a physical, a logical, and a pragmatical bridge between grammarware language and modeling framework.

By exploiting our ASM-based semantic framework [42], we also defined the semantics of the AVALLA language [43] of the AsmetaV validator, a domain-specific modeling language for scenario-based validation of ASM models. Moreover, in [44] we adapt one of the techniques in [42], the *meta-hooking*, for UML profiles, and we show its application to the *SystemC Process (SCP) state machines* formalism of the SystemC UML profile [45].

III. MDE for FMs

Applying the MDE development principles to a formal method has the overall goal of engineering a language and a tool-set around the formal method in order to support its practical use in systems development life cycle.

The MDE methodology for engineering software languages is well established in the context of domain-specific languages [46]. Nevertheless, this model-driven development process can be adapted to formal methods, too.

The first step of this engineering process is the *choice of a metamodeling framework and its supporting technologies*. In principle, the choice of a specific meta-modeling framework should not prevent the use of models in other different meta-modeling spaces, since model transformations among meta-modeling framework should be theoretically supported by the environments. However, although in theory one could switch framework later, a commitment with a precise meta-modeling framework is better done at the very early stage of the development process, mainly for practical reasons. The chosen MDE framework should support easy (e.g., graphical) editing of (meta) models, model to model transformations, and text to model and model to texts mappings to assist the development of concrete notations in textual form. It should also provide a mapping towards programming languages (i.e. API artifacts) to allow the integration with other software applications.

Once a metamodeling framework has been chosen, the further main steps, that might require iterative processing, of the process are the following.

Design of a language abstract syntax. In the MDE context, the *abstract syntax* of a specification language is defined by means of a *metamodel* [47]. It is an object-oriented model of the vocabulary of the language. It represents concepts provided by the language, the relationships existing among those concepts, and how they may be combined to create models. Precise guide lines exist (e.g., [46]) to drive this modeling activity that leads to an instantiation of the chosen metamodeling framework for a specific domain of interest. This is a critical process step since the metamodel is the starting point for tool development.

Development of tools. Software tools are developed starting from the language metamodel. They can be classified in *generated*, *based*, and *integrated*, depending on the decreasing use of MDE generative technologies for their development. The effort required by the user increases, instead. Software tools automatically derived from the metamodel are considered generated. Based tools are those developed exploiting artifacts (APIs and other concrete syntaxes) and contain a considerable amount of code that has not been generated. Integrated tools are external and existing tools that are connected to the language artifacts: a tool may use just the XMI format, other tools may use the APIs or other derivatives. In the sequel we explain these kinds of tools.

1) *Development of language artifacts.* From the language metamodel, several *language artifacts* are generated for model handling – i.e. model creation, storage, exchange, access, manipulatation –, and these artifacts can be reused during the development of other applications. Artifacts are obtained by exploiting standard or proprietary mappings from the metamodeling framework to several technical spaces, as XMLware for model serialization and interchange, and Javaware for model representation in terms of programmable objects (through standard APIs).

2) *Definition and validation of* concrete syntax(es). Language concrete notations (textual, graphical or both) can be introduced for the human use of editing models conforming

to the metamodel. Several tools exist to define (or derive) concrete textual grammars for metamodels. For example, EMFText [22] allows defining text syntax for languages described by an Ecore metamodel and it generates an ANTLR grammar file. TCS [23] (Textual Concrete Syntax) enables the specification of textual concrete syntaxes for Domain-Specific Languages (DSLs) by attaching syntactic information to metamodels written in KM3. A similar approach is followed by the TEF (Textual Editing Framework) [48]. Other tools, like the Xtext by openArchitectureWare [49], following different approaches, may fit in our process as well. Depending on the degree of automation provided by the chosen framework, concrete syntax tools can be classified between generated and based software.

Besides to be defined, concrete grammars must be also validated. To this aim, a pool of models written in the concrete syntax and acting as benchmark has to be selected. During this activity it is important to collect information about the coverage of language constructs (classes, attributes and relations) to check that all them are used by the examples. Writing wrong models and checking that they are not accepted is important as well. Coverage evaluation can be performed by using a code coverage tool and instrumenting the parser accordingly. This validation activity is also useful to provide confidence that the metamodel correctly captures concepts and constructs of the underline formal method.

3) *Development of other tools.* Metamodel, language artifacts, and concrete syntaxes are the foundations over which new tools can be developed and existing ones can be integrated.

IV. FMs for MDE

Applying a formal method to a language L defined in a meta-modeling framework should have the following overall goals: (a) allow the definition of the behaviors (semantics) of models conforming to L and (b) provide several techniques and methods for the formal analysis (e.g., validation, property proving, model checking, etc.) of such models.

A. Language semantics definition

A metamodel-based language L has a well-defined semantics if a semantic domain S is identified and a semantic mapping $M_S : A \rightarrow S$ is provided [50] between the L's abstract syntax A (i.e. the metamodel of L) and S to give meaning to syntactic concepts of L in terms of the semantic domain elements.

The semantic domain S and the mapping M_S can be described in varying degrees of formality, from natural language to rigorous mathematics. It is very important that both S and M_S are defined in a precise, clear, and readable way. The semantic domain S is usually defined in some formal, mathematical framework (transition systems, pomsets, traces, the set of natural numbers with its underlying properties, are examples of semantic domains). The semantic mapping M_S is not so often given in a formal and precise way, possibly leaving some doubts about the semantics of L. Thus, a precise and formal approach to define it is desirable.

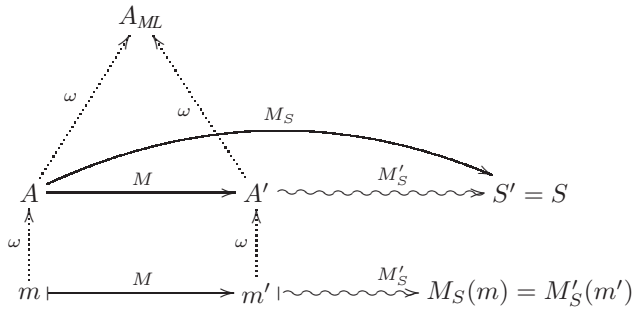


Fig. 2: The building function M

Sometimes, in order to give the semantics of a language L, another helper language L', whose semantics is clearly defined and well established, is introduced. Therefore, M'_S and S' should be already well-defined for L'. L' can be exploited to define the semantics of L by:

1) taking S' as semantic domain for L too, i.e. $S = S'$,
2) introducing a *building function* $M : A \rightarrow A'$, being A' the abstract syntax of L', which associates an element of A' to every construct of A, and
3) defining the semantic mapping $M_S : A \rightarrow S$ as

$$M_S = M'_S \circ M$$

The M function *hooks* the semantics of A to the S' semantic domain of the language L'. The complexity of this approach depends on the complexity of building the function M.

Note that the function M can be applied to terminal models conforming to A in order to obtain models conforming to A', as shown in Fig. 2. In this way, the semantic mapping $M_S : A \rightarrow S$ associates a well-formed terminal model m conforming to A with its semantic model $M_S(m)$, by first translating m to m' conforming to A' by means of the M function, and then applying the mapping M'_S which is already well-defined.

To be a good candidate, a language L' should (i) be abstract and formal to rigorously define model behavior at different levels of abstraction, but without formal overkill; (ii) be able to capture heterogeneous models of computation (MoC) in order to smoothly integrate different behavioral models; (iii) be endowed with a model refinement mechanism leading to correct-by-construction system artifacts. Furthermore, as MDE specific requirement (iv), L' should be possibly endowed with a metamodel-based definition in order to automatize the application of building function M by exploiting MDE techniques of automatic model transformation.

B. Formal analysis

Besides the above stated requirements about the expressive power of L' as notation, it is important that formal analysis of models written in L' is supported by a set of tools for model execution, as simulation or testing, and for model verification. Indeed, the main goal of applying a formal notation to the semantics of L is to allow formal analysis of the models written in L.

As main formal activities that are allowed by applying a formal method to a language L, we identify at least: *model validation* and *property verification*.

Validation is intended as the process of investigating a model (intended as formal specification) with respect to its user perceptions, in order to ensure that the specification really reflects the user needs and statements about the application, and to detect faults in the specification as early as possible with limited effort. Techniques for validation include *scenarios generation*, when the user builds scenarios describing the behavior of a system by looking at the observable interactions between the system and its environment in specific situations; *simulation*, when the user provides certain input and observes if the output is the expected one or not (it is similar to code debugging); *model-based testing*, when the specification is used as oracle to compute test cases for a given critical behavior of the system at the same level of the specification. These abstract test cases cannot be executed at code level since they are at a wrong level of abstraction. Executable test cases must be derived from the abstract ones and executed at code level to guarantee conformance between model and code.

In any case, validation should precede the application of more expensive and accurate methods, like *requirements formal analysis* and *verification of properties*, that should be applied only when a designer has enough confidence that the specification captures all informal requirements. Formal verification has to be intended as the mathematical proof of system properties, which can be performed by hand or by the aid of model checkers (which are usable when the variable ranges are finite) or of theorem provers (which require strong user skills to drive the proof).

Model validation techniques can be also used during the development of the language semantics of L for *semantic validation*. This activity consists in checking (or proving, if possible) that the building function M really captures the intended semantics of L, and it must be performed before any formal analysis of models. Indeed every later formal activity on models written in L is based on M and a faulty M would jeopardize the results obtained.

V. In-the-loop integration

Although the two activities of applying the MDE to a FM and apply a FM to the MDE can be considered unrelated and could be performed in parallel even by using two different notations for the MDE and FMs, the best results can be obtained by a tight integration between the MDE and a FM in an *in-the-loop* integration approach. In this approach, the MDE framework and the FM notation are the same in both of the above activities and the application of the MDE to the FM is carried out before the application of the FM to the MDE. Thanks to the first activity, the FM will be endowed with a metamodel and possibly a set of tools (e.g., a grammar, artifacts, etc.) which can be used in the second activity to automatize (meta-)model transformations and apply suitable tools for formal analysis (i.e. validation and verification) of models. Indeed, although for applying FM to the MDE it is in principle not required that the FM is provided with a metamodel (see Sect. IV), a formal notation endowed with a representation of its concepts in terms of a metamodel would allow the use of MDE transformation languages (as ATL) to define the building function M and to automatize the application of M as model transformation by means of a transformation engine. Therefore, having a metamodel is a further constraint for an helper language L', and it justifies why the second activity must precede the first one.

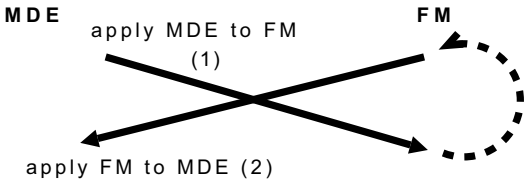


Fig. 3: In the loop integration of FM and MDE

Sect. VII and VIII present our instantiation of the *in-the-loop* integration with the EMF (Eclipse Modeling Framework) as MDE framework and the ASMs (Abstract State Machines) as formal method. This choice is justified by the following motivations:

- EMF is based on an open-source Eclipse framework and unifies the three well known technologies, i.e. Java, XML, and UML, currently used for software development.
- ASMs own all the characteristics of preciseness, abstraction, refinement, executability, metamodel-based definition that we identified as the desirable properties a FM should have in order to be a good candidate for integration.

In order to make a further step in the direction of a tighter integration between ASM and EMF, Sect. X shows how effectively we can *close the loop* (see Fig. 3) by describing the semantics of ASMs representation in the EMF framework by using the ASM formal method itself.

VI. Abstract State Machines

Abstract State Machines are an extension of FSMs [51], where unstructured control states are replaced by states comprising arbitrary complex data. The *states* of an ASM are multi-sorted first-order structures, i.e. domains of objects with functions and predicates (boolean functions) defined on them, while the *transition relation* is specified by "rules" describing the modification of the functions from one state to the next.

Basically, a transition rule has the form of *guarded update* "**if** *Condition* **then** *Updates*" where *Updates* is a set of function updates of the form $f(t_1, \ldots, t_n) := t$ that are simultaneously executed when *Condition* is true, f is an arbitrary n-ary function, and $t_1, \ldots, t_n, t$ are first-order terms. To fire this rule to a state S_i, $i \geq 0$, evaluate all terms $t_1, \ldots, t_n, t$ at S_i and update the function f to t on parameters $t_1, \ldots, t_n$. This produces another state S_{i+1} which differs from S_i only in the new interpretation of the function f. An ASM M is therefore a finite set of rules for such guarded multiple function updates.

Function are classified as *derived* functions, i.e. those coming with a specification or computation mechanism given in terms of other functions, and *basic* functions which can be

static (never change during any run of the machine) or *dynamic* (may change as a consequence of agent actions or *updates*). Dynamic functions are further classified into: *monitored* (only read, as events provided by the environment), *controlled* (read and write), *shared* and *output* (only write) functions.

These is a limited but powerful set of *rule constructors* that allow to express simultaneous parallel actions (`par`), sequential actions (`seq`), iterations (`iterate`, `while`, `rec-while`), and submachine invocations returning values. Appropriate rule constructors also allow non-determinism (existential quantification `choose`) and unrestricted synchronous parallelism (universal quantification `forall`).

A *computation* of an ASM M is a finite or infinite sequence $S_0, S_1, \ldots, S_n, \ldots$ of states of M, where S_0 is an initial state and each S_{n+1} is obtained from S_n by firing simultaneously all of the transition rules which are enabled in S_n.

The notion of ASMs formalizes simultaneous parallel actions of a single agent, either in an atomic way, *Basic ASMs*, or in a structured and recursive way, *Structured or Turbo ASMs*. Furthermore, it supports a generalization where multiple agents interact in parallel in a synchronous/asynchronous way, *Synchronous/Asynchronous Multi-agent ASMs*.

Although the ASM method comes with a rigorous mathematical foundation, ASMs provide accurate yet practical industrially viable behavioral semantics for pseudocode on arbitrary data structures. We quote here this *working* definition of an ASM defined as a tuple (*header*, *body*, *main rule*, *initialization*).

The *header* contains the *name* of the ASM and its *signature*, namely all declarations of domains, functions, and predicates. The header may contain also *import* and *export* clauses, i.e., all names for functions and rules that are, respectively, imported from other ASMs, and exported from the current one. We assume that there are no name clashes in these signatures.

The *body* of an ASM consists of (static) domain and (static/derived) function definitions according to domain and function declarations in the signature of the ASM. It also contains declarations (definitions) of transition rules and definitions of *axioms* for invariants one wants to assume for domains and functions of the ASM.

The (unique) *main rule* is a transition rule and represents the starting point of the machine program (i.e. it calls all the other ASM transition rules defined in the body). The main rule is *closed* (i.e. it does not have parameters) and since there are no free global variables in the rule declarations of an ASM, the notion of a move does not depend on a variable assignment, but only on the state of the machine.

The *initialization* of an ASM is a characterization of the initial states. An initial state defines an initial value for domains and functions declared in the signature of the ASM. *Executing* an ASM means executing its main rule starting from a specified initial state.

A complete mathematical definition of the ASM method can be found in [52], together with a presentation of the great variety of its successful application in different fields such as: definition of industrial standards for programming and modeling languages, design and re-engineering of industrial control systems, modeling e-commerce and web services, design and analysis of protocols, architectural design, language design, verification of compilation schemas and compiler back-ends, etc.

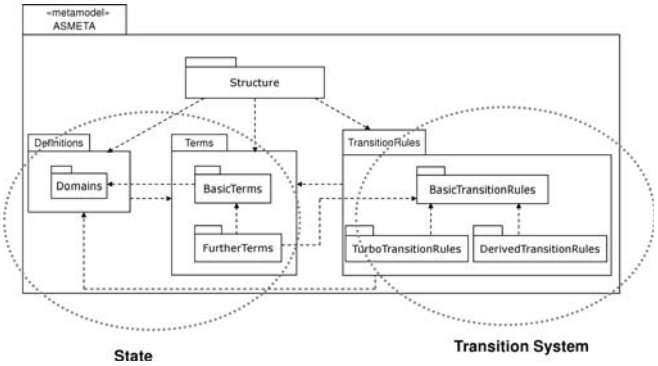


Fig. 4: Package structure of the AsmM metamodel

VII. EMF FOR ASMS

In addition to its mathematical-based foundation, a metamodel-based definition for ASMs has been given [53], [54]. This ASM metamodel allowed us to apply MDE techniques for developing a general framework, called ASMmETA- modeling framework (ASMETA) [55], for a wide inter-operability and integration of new and existing tools around ASMs (ASM model editors, ASM model repositories, ASM model validators, ASM model verifiers, ASM simulators, ASM-to-Any code generators, etc.).

A. ASM Metamodel

We started by defining a metamodel [55], [56], [53], [54], the *Abstract State Machine Metamodel* (AsmM), as abstract syntax description of a language for ASMs. The aim was that of developing a *unified* abstract notation for the ASMs, independent from any specific implementation syntax and allowing a more direct encoding of the ASM mathematical concepts and constructs.

The complete AsmM metamodel is organized in one package called ASMETA containing 115 classes, 114 associations, and 150 class invariants expressed in the OMG OCL language [57], approximatively. The ASMETA package is further divided into four packages as shown in Fig. 4. Each package covers different aspects of the ASMs. The dashed gray ovals in Fig. 4 denote packages representing the notions of *State* and *Transition System*, respectively. The `Structure` package defines architectural constructs (modules and machines) required to specify the backbone of an ASM model. The `Definitions` package contains all basic constructs (functions, domains, constraints, rule declarations, etc..) which characterize algebraic specifications. The `Terms` package provides all kinds of syntactic expressions which can be evaluated in a state of an ASM. The `TransitionRules` package contains all possible transition rules schemes of Basic and Turbo ASMs. All *derived* transition rules are contained in the `DerivedTransitionRules` package. These rules are other ASM transition rule schemes derived from the basic

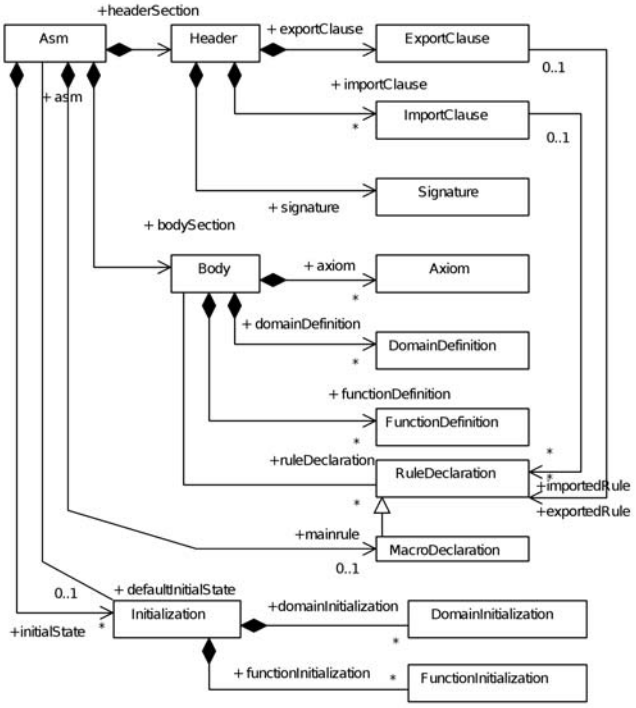


Fig. 5: Backbone

and the turbo ones, respectively. Although they could be easily expressed at model level in terms of other existing rule schemes, they are considered "syntactic sugar" and therefore they have been included in the metamodel. Example of such rules are the case-rule and the (turbo) iterative/recursive while-rule. All relations between packages are of type *uses*.

We present here only a very small fragment of the AsmM whose complete description can be found in [53], [55]. Fig. 5 shows the backbone of a *basic ASM*. An instance of the root class `Asm` represents an entire ASM specification. According to the definition given in Sect. VI, a basic ASM has a `name` and is defined by a `Header` (to establish the signature), a `Body` (to define domains, functions, and rules), a `main rule`, and a set of initial states (instances of the `Initialization` class). All possible initial states are linked to an ASM by the association end `initialState` and one initial state is elected as *default* (see the association end `defaultInitialState`). ASM rule constructors are represented by subclasses of the class `Rule`, not reported here.

B. ASMETA *tool-set*

From the AsmM, by exploiting the MDE approach and its facilities (derivative artifacts, APIs, transformation libraries, etc.), we obtained in a generative manner (i.e. semi-automatically) several artifacts (an interchange format, APIs, etc..) for the creation, storage, interchange, access and manipulation of ASM models [58]. The AsmM and the combination of these language artifacts lead to an instantiation of the EMF metamodeling framework for the ASM application domain, the ASMETA framework that provides a global infrastructure

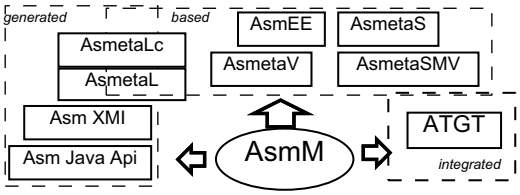


Fig. 6: The ASMETA tool set

for the interoperability of ASM tools (new and existing ones) [59].

The ASMETA tool set (see Fig. 6) includes (among other things) a textual concrete syntax, *AsmetaL*, to write ASM models (conforming to the AsmM) in a textual and human-comprehensible form; a text-to-model compiler, *AsmetaLc*, to parse AsmetaL models and check for their consistency w.r.t. the AsmM constraints expressed in the OCL language; a simulator, *AsmetaS*, to execute ASM models; the *Avalla* language for scenario-based validation of ASM models, with its supporting tool, the *AsmetaV* validator; a model checker *AsmetaSMV* [60] for model verification by NuSMV; the *ATGT* tool that is an ASM-based test case generator based upon the SPIN model checker; a graphical front-end called *ASMEE* (ASM Eclipse Environment) which acts as IDE and it is an eclipse plug-in.

All the above artifacts/tools are classified in: *generated*, *based*, and *integrated*. Generated artifacts/tools are derivatives obtained (semi-)automatically by applying appropriate Ecore projections to the technical spaces Javaware, XMLware, and grammarware. Based artifacts/tools are those developed exploiting the ASMETA environment and related derivatives; an example of such a tool is the simulator AsmetaS). Integrated artifacts/tools are external and existing tools that are connected to the ASMETA environment.

VIII. ASMs FOR EMF

We here describe how the ASM formal method can be exploited as helper language to define a formal *semantic framework* to provide languages with their (possible *executable*) semantics natively with their metamodels. We also describe how the ASM tool-set provides a concrete support for model analysis.

A. Language semantics definition

Recall, from Sect. IV, that the problem of giving the semantics of a metamodel-based language L is reduced to define the function $M : A \rightarrow A'$, being A and A' the language and the helper language abstract syntaxes, respectively. Let us assume the ASMs as helper language satisfying the requirements, given in Sect. IV, of having a mathematical well-founded semantics and a metamodel-based representation. The semantic domain S_{AsmM} is the first-order logic extended with the logic for function updates and for transition rule constructors defined in [52] and the *semantic mapping* $M_S : AsmM \rightarrow S_{AsmM}$ to relate syntactic concepts to those of the semantic domain is given in [58].

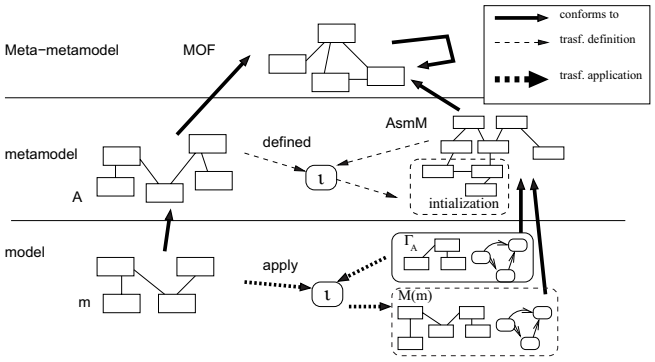


Fig. 7: Semantic hooking

The semantics of a metamodel-based language is expressed in terms of ASM transition rules by providing the building function $M : A \longrightarrow AsmM$. As already mentioned, the definition of the function M may be accomplished by different techniques (see [42]), which differ in the way a terminal model is mapped into an ASM. As example of such techniques, the *semantic hooking* technique is presented below. This technique is used in Section IX-B to provide behavioral semantics of the language in our case study.

The *semantic hooking* endows a language metamodel A with a semantics by means of a unique ASM for any model conforming to A. By using this technique, designers *hook* to the language metamodel A an abstract state machine Γ_A, which is an instance of *AsmM* and contains all data structures modeling elements of A with their relationships, and all transition rules representing behavioral aspects of the language. Γ_A does not contain the initialization of functions and domains, which will depend on the particular instance of A. The function which adds the initialization part is called ι. Formally, the building function M is given by $M(m) = \iota_A(\Gamma_A, m)$, for all m conforming to A.

Γ_A: $AsmM$, is an abstract state machine which contains only declarations of functions and domains (the signature) and the behavioral semantics of L in terms of ASM transition rules.

ι_A: $AsmM \times A \longrightarrow AsmM$, properly initializes the machine. ι_A is defined on an ASM a and a terminal model m instance of A; it navigates m and sets the initial values for the functions and the initial elements in the domains declared in the signature of a. The ι_A function is applied to Γ_A and to the terminal model m for which it yields the final ASM.

Examples of applying the semantic hooking technique to define the semantics of a metamodel-based language can be found in [42] for a metamodel of Finite State Machines and in [1] for a metamodel of the Petri net formalism. The latter is also reported in Appendix A and can be viewed as an example which facilities the reader in understanding our approach since the semantics of Petri nets is well-known.

B. Formal analysis

The ASM-based semantic framework supports formal analysis of ASM models by exploiting the ASMETA tool-set (see Section VII-B for details) for model validation and verification.

1) Model validation: Simple model validation can be performed by *simulating* ASM models with the ASM simulator (see Section VII-B) to check a system model with respect to the desired behavior to ensure that the specification really reflects the user needs and statements about the system, and to detect faults in the specification as early as possible with limited effort.

The AsmetaS simulator can be used in a standalone way to provide basic simulation of the overall system behavior. As key features for model validation, AsmetaS supports *axiom checking* to check whether axioms expressed over the currently executed ASM model are satisfied or not, *consistent updates checking* for revealing inconsistent updates, *random simulation* where random values for monitored functions are provided by the environment, *interactive simulation* when required input are provided interactively during simulation, and configurable *logging* facilities to inspect the machine state. Axiom checking and random simulation allow the user to perform a draft system validation with minimal effort, while interactive simulation, although more accurate, requires the user interaction.

The most powerful validation approach is the *scenario-based validation* [61] by the ASM validator (see Section VII-B). The AsmetaV validator is based on the AsmetaS simulator and on the Avalla modeling language. This last provides constructs to express execution scenarios in an algorithmic way as interaction sequences consisting of *actions* committed by the *user actor* to `set` the environment (i.e. the values of monitored/shared functions), to `check` the machine state, to ask for the `execution` of certain transition rules, and to enforce the machine itself to make one `step` (or a sequence of steps by `step until`) as reaction of the actor actions.

AsmetaV reads a user scenario written in Avalla, it builds the scenario as instance of the Avalla metamodel by means of a parser, it transforms the scenario and the AsmetaL specification which the scenario refers to, to an executable AsmM model. Then, AsmetaV invokes the AsmetaS interpreter to simulate the scenario. During simulation the user can pause the simulation and watch the current state and value of the update set at every step, through a watching window. During simulation, AsmetaV captures any check violation and if none occurs it finishes with a "PASS" verdict. Besides a "PASS"/"FAIL" verdict, during the scenario running AsmetaV collects in a final report some information about the *coverage* of the original model; this is useful to check which transition rules have been exercised.

2) Model checking: The ASMETA tool-set provides support for temporal properties verification of ASM models by means of the model checker AsmetaSMV [60], which takes in input ASM models written in AsmetaL and maps these models into specifications for the model checker NuSMV [62].

AsmetaSMV supports both the declaration of *Computation Tree Logic* (CTL) and *Linear Temporal Logic* (LTL) formulas. CTL/LTL properties to verify are declared directly into the ASM model as (special) axioms of the form:

$$\textbf{axiom over } [ctl \mid ltl] : \quad p$$

where the over section specifies if p is a CTL or a LTL formula. No knowledge of the NuSMV syntax is required to

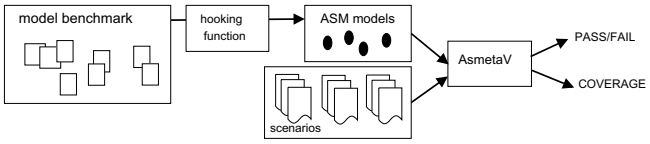


Fig. 8: Semantic validation by AsmetaV

the user in order to use AsmetaSMV.

3) Language semantics validation: The ASMETA tool-set and the validation techniques can also be used for *language semantics validation*. Indeed, this activity is performed through the validation of the hooking function M presented in Section VIII-A by applying it to a collection of meaningful examples. The ASM models obtained form the application of M to the examples can be validated in different ways providing increasing degrees of confidence in the semantics correctness. *Random simulation* allows checking if errors like inconsistent updates and type errors, occur. *Interactive simulation* can provide evidence that the semantics captures the intended behavior, but it requires the user to provide the correct inputs and to judge the correctness of the observed behavior. The most powerful validation approach is the *scenario-based validation*. As shown in Fig. 8, a suitable set of models are selected as benchmark for language semantic validation; these models are translated into ASM models by the hooking function M; moreover, a set of scenarios specifying the expected behavior of the models must be provided by the user and are used for validation. These scenarios can be written from scratch in the Avalla language, or alternatively, if the language L has already a simulator, these scenarios may be derived from the execution traces generated by such a simulator. The second approach is useful to check the conformance of the semantics implemented by L_S with respect to the semantics defined by the hooking function M. The ASM validator provides also useful information about the coverage obtained by the scenarios.

IX. The Tic-Tac-Toe example

As a case study, we consider Tic-Tac-Toe as a language, where a Tic-Tac-Toe board is an instance of the language. We use MDE-based technologies to define a metamodel for a description language of the Tic-Tac-Toe game, and the ASM-based semantic framework for the definition of the execution semantics of a board (for playing) including correctness checking by validation and verification.

A. Tic-Tac-Toe abstract syntax

Fig. 9 shows the metamodel for the Tic-Tac-Toe. It describes the static structure of a board (the `Board` class) maintaining data seen by users: rows (the `Row` class) and squares (the `Square` class). A board has (see references `hrows`, `vrows`, and `drows`): three horizontal rows, three vertical rows, and two diagonal rows. Totally, in a board there are nine squares (see the reference `square`), three per each row (the `squareInRow` reference). The `SKind` enumeration type denotes the kind of symbols a square can contain (cross, nought, empty). The default symbol is empty.

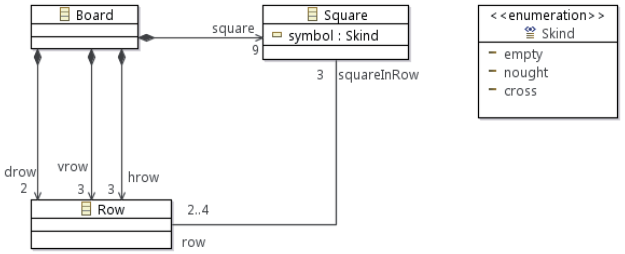


Fig. 9: A metamodel for Tic-Tac-Toe

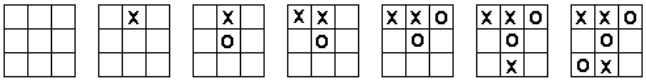


Fig. 10: Examples of Tic-Tac-Toe boards

Each square is contained in one row and one vertical row. Some squares may be contained in more than one row. The square in the center, for example, is contained in the middle vertical row and horizontal row, and in the two diagonal rows. All these structural constraints can be expressed in OCL. For example, the following OCL invariant

```
Context:Board
inv RowColumnCommonSquares:
self.hrow.squareInRow->
 intersection(self.vrow.squareInRow)->size()=1
```

states that an horizontal row and a vertical row can only have exactly one square in common.

Fig. 10 shows (using a graphical concrete syntax) examples of Tic-Tac-Toe boards as instances (terminal models) of the Tic-Tac-Toe metamodel in Fig 9.

B. Tic-Tac-Toe semantics definition

According to the hooking technique, first we have to specify an ASM $\Gamma_{Tic-Tac-Toe}$ containing the signature and the behavioral semantics of the Tic-Tac-Toe metamodel in terms of ASM transition rules. Listings 1 (for the signature), 2 and 3 (for the transition rules) report portions of a possible $\Gamma_{Tic-Tac-Toe}$ in AsmetaL for a computer (symbol O) vs user (symbol X) Tic-Tac-Toe game. The complete ASM model is reported in Appendix B.

The signature (see Listing 1) introduces domains and functions for representing a board such as the enumeration `SKind`, domains for squares and rows as subsets of the predefined `Integer` domain, and so on. The signature also provides domain and functions for managing the overall game. Each player takes alternating turns (see the function `status`) trying to earn three of their symbols in a row horizontally, vertically, or diagonally. The game can end with a player winning (represented by the `whoWon` function) by getting three of his/her symbol in row (as denoted by the function `hasThreeOf`) or end in a draw, i.e. no spaces left on the board with none winning (as denoted by the `noSquareLeft` function). The winner is determined by position of board; no history needs to be recorded (only board position before and after turn). If there is no winner after nine clicks, there is a

Listing 1: $\Gamma_{Tic-Tac-Toe}$ signature

```
asm Tictactoe
signature:
  //For representing a board
  enum domain Skind = {CROSS|NOUGHT|EMPTY}
  domain Square subsetof Integer
  domain Row subsetof Integer
  static squaresInRow: Prod(Row,Integer) -> Square
  controlled symbol: Square -> Skind

  //For managing the game
  enum domain Finalres = {PLAYERX|PC|TIE}
  enum domain Status = {TURNX|CHECKX|TURNPC|CHECKPC
                        |GAMEOVER}
  monitored playerX:Square // move of X
  controlled status: Status
  controlled whoWon: Finalres
  derived noSquareLeft : Boolean
  derived hasThreeOf: Prod(Row,Skind) -> Boolean

  //For PC strategies
  controlled count: Integer
  derived openingPhase: Boolean
  controlled lastMoveX: Square
  static isCorner: Square -> Boolean
  static isEdge: Square -> Boolean
  static isCenter: Square -> Boolean
  derived hasTwo: Row -> Boolean
  static opposite: Square -> Square
```

Listing 2: $\Gamma_{Tic-Tac-Toe}$ transition rules for game management

```
asm Tictactoe
...
 rule r_movePC = par
                   r_tryStrategy[NOUGHT]
                   count := count + 1
                   status := CHECKPC
                 endpar

 rule r_movePlayerX = if symbol(playerX)= EMPTY
                      then par
                             symbol(playerX):= CROSS
                             count := count + 1
                             lastMoveX := playerX
                             status := CHECKX
                           endpar
                      else status := TURNX
                      endif


 rule r_checkForAWinner($symbol in Skind) =
          //GAME OVER WITH A WINNER?
          if (exist $r in Row with hasThreeOf($r,$symbol)) then
           par
             status := GAMEOVER
             if $symbol = CROSS then whoWon:= PLAYERX
             else whoWon:= PC
             endif
           endpar
               //GAME TIE?
          else if ( noSquareLeft )
           then par
             status := GAMEOVER
                    whoWon := TIE
           endpar
          else
             if $symbol = CROSS then status:= TURNPC
             else status:= TURNX
          endif endif endif

  main rule r_Main =
          if status = TURNX then r_movePlayerX[]
          else if status = CHECKX then r_checkForAWinner[CROSS]
          else if status = TURNPC then r_movePC[]
          else if status = CHECKPC then r_checkForAWinner[NOUGHT]
          endif endif endif endif
```

tie. Note that the square selected by the player X (the user) is represented by a monitored function `moveX`, and therefore is provided at each step as input value to the ASM; the computer move (the square to mark) is instead calculated according to some playing strategies. Further domains and functions are introduced in the signature to implement these PC strategies, as better explained later in the text.

The behavior of the overall game is provided by the main rule *r_Main* (see Listing 2) where at each step a check for a winner or a tie (rule *r_checkForAWinner*) or a move of a player is executed depending on the status of the game. The two rules *r_movePlayerX* and *r_movePC* specify the execution behavior of the two players. The behavior of the user (player X) is straightforward as the square to mark is provided interactively through the monitored function `moveX`. The behavior of the computer depends instead by the chosen strategy as formalized by the invoked *r_tryStrategy* rule.

Listing 3 reports the definition of the *r_tryStrategy* rule and of the invoked macro rules for making a computer play the game. To this goal, we formalize by ASM rules a children's strategy that is divided in two phases: *opening phase* (opening of the game) and *draw phase* (after opening of both players). Note that to build an unbeatable opponent (especially if we want to learn a computer to play it), we need to use a *minimax* approach of Game Theory. We remark that this is out of the scope of this work. So, here we limit to express a children's strategy.

For the opening phase (see the *r_opening_strategy* rule in Listing 3), as first player the computer has three possible positions to mark during the first turn. Superficially, it might seem that there are nine possible positions, corresponding to the nine squares in the board. However, by rotating the board, we will find that in the first turn, every corner mark is strategically equivalent to every other corner mark. The same is true of every edge mark. For strategy purposes, there are therefore only three possible first marks: corner, edge, or center. The computer can win or force a draw from any of these starting marks; however, playing the corner gives the opponent the smallest choice of squares which must be played to avoid losing. In the *r_opening_strategy* rule, the computer chooses therefore a corner (see the rule *r_playACorner*) in case of first player. As second player, the computer must respond to X's opening mark in such a way as to avoid the forced win. The computer (player O) must always respond to a corner opening with a center mark, and to a center opening with a corner mark. An edge opening must be answered either with a center mark, a corner mark next to the X, or an edge mark opposite the X. For semplicity, in this case we play always the center as formalized in the *r_opening_strategy* rule. Any other responses will allow X to force the win. Once the opening is completed, O's task is to follow the below draw strategy in order to force

Listing 3: $\Gamma_{Tic-Tac-Toe}$ transition rules for the game strategies

```
asm Tictactoe
...
  //A very naive player: choose an empty square and mark it.
  rule r_naive_strategy ($symbol in Skind)=
   choose $s in Square with symbol($s)=EMPTY
   do symbol($s):= $symbol

  rule r_playACorner($symbol in Skind) =
   choose $s in Square with (symbol($s)=EMPTY and isCorner($s))
   do symbol($s):= $symbol

  //Opening strategy
  rule r_opening_strategy ($symbol in Skind)=
      if (count=0) //first mark
      then r_playACorner[$symbol]
      else //second mark
          if symbol(5) = EMPTY then symbol(5):=$symbol //play the center
          else r_playACorner[$symbol] //we play a corner
          endif
      endif

  //Mark with $symbol the last empty square within row $r
  rule r_markLastEmpty ($r in Row, $symbol in Skind) =
   choose $x in {1,2,3} with symbol(squaresInRow($r,$x))=EMPTY
    do symbol(squaresInRow($r,$x)) := $symbol

  //Draw strategy (with no fork creation/block)
  rule r_draw_strategy ($symbol in Skind) =
    choose $wr in Row with hasTwo($wr)
    do r_markLastEmpty[$wr,$symbol] //1. Win or 2. Block
    ifnone
      if (symbol(5)=EMPTY)
      then symbol(5):=$symbol //3. Center
      else if (isCorner(lastMoveX) and symbol(opposite(lastMoveX))=EMPTY)
      then symbol(opposite(lastMoveX)):= $symbol //4. Opposite corner
      else choose $s in Square with (symbol($s)=EMPTY and isCorner($s))
            do symbol($s):= $symbol //5. Empty Corner
            ifnone r_naive_strategy[$symbol] //6. Empty edge
      endif endif

  //Computer strategy selection
  rule r_tryStrategy ($symbol in Skind) =
   if openingPhase then r_opening_strategy[$symbol]
   else r_draw_strategy[$symbol]
   endif
```

Listing 4: A winning scenario for player O

```
1  scenario winPC
2  load Tictactoe.asm
3  set playerX := 2;
4  step until status = TURNPC;
5  step until status = TURNX;
6  check symbol(2)=CROSS;
7  check symbol(5)=NOUGHT;
8  set playerX := 1;
9  step until status = TURNPC;
10 step until status = TURNX;
11 check symbol(1)=CROSS;
12 check symbol(3)=NOUGHT;
13 set playerX := 8;
14 step until status = GAMEOVER;
15 check symbol(7)=NOUGHT;
16 check whoWon = PC;
```

the draw, or else to gain a win if X makes a weak play.

For the draw phase (see the *r_draw_strategy* rule in Listing 3), the PC try a *draw strategy* with no fork creation or block. Essentially, the computer can play Tic-Tac-Toe if it chooses the move with the highest priority in the following list:
1. Win: you have two in a row, play the third to get three in a row.
2. Block: the opponent has two in a row, play the third to block.
3. Center: Play the center.
4. Opposite Corner: the opponent is in the corner, play the opposite corner.
5. Empty Corner: Play an empty corner.
6. Empty Side: Play an empty edge.

For this example, the function $\iota_{Tic-Tac-Toe}$ that adds to $\Gamma_{Tic-Tac-Toe}$ the initialization necessary to make the ASM model executable do not present variability among terminal models (unless one want to start playing from a partially full board). In this case, $\iota_{Tic-Tac-Toe}$ is to be intended as a constant function always producing in the target ASM model the same ASM initial state. One possible, for example, is as follows:

```
default init s0:
 function symbol($s in Square) = EMPTY
 //A polite computer: it allows the user (X) to play first
 function status = TURNX
 function count = 0
```

C. Tic-Tac-Toe semantic validation

The validation of the semantics of the Tic-Tac-Toe case study consists in checking that the mapping function defined in IX-B really captures the intended semantics of the case study language. Among the semantics validation techniques discussed in Section VIII-B, we have used interactive and scenario-based simulation. By interactive simulation, we have used the ASM specification and the AsmetaS simulator to interactively play Tic-Tac-Toe (player vs computer) and check that the ASM model actually captures the desired behavior.

For scenario-based simulation, Listing 4 reports a scenario in Avalla corresponding to the board configurations shown in Fig. 10. In this scenario, the player opens by crossing cell 2 (line 3), the PC responds in the cell 5 (line 7), and the player crosses cell 1. At this point the PC correctly responds by occupying cell 3 (line 12). If the player puts the cross in cell 8 (line 13), the PC takes advantage of that and wins. This scenario shows the smart opening of the PC (as second player) and that the PC is able both to block the player to win and to take advantage of the opportunity to win.

D. Tic-Tac-Toe formal verification

Once we were confident that the semantics of the Tic-Tac-Toe as specified really captures the intended behavior, we tried to model and *prove* some formal properties. The first one states that the specification is fair and allows both player to win. To model this fact, we have introduced in the specification the following three temporal properties written in Computational Tree Logic (CTL).

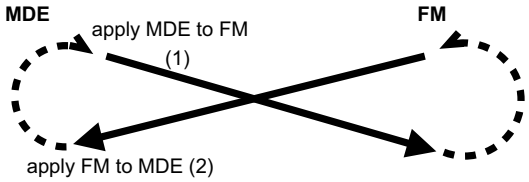


Fig. 11: Closing the in-the-loop integration

//the player can win
axiom over CTL: EF(whoWon=PLAYER)
//the computer can win
axiom over CTL: EF(whoWon=PC)
//the match can terminate tie
axiom over CTL: EF(whoWon=TIE)

The meaning of `EF`(ϕ) is given by the `E` (*exist*) operator which means along at least one path (possibly) and the `F` operator which means finally: eventually ϕ has to hold (somewhere on the subsequent path). We have automatically proved the three properties via model checking by using the AsmetaSMV component [60].

We wanted also to prove that the match always finishes and we added the following property:

axiom over CTL: AF((status = GAMEOVER))

It means that on all paths (`A`) starting from the initial state, *status* will eventually (`F`) become *GAMEOVER*. This was proved false by the model checker which provided a counter example for it. Analyzing the counter example, we noticed that the player can indefinitely postpone the end of a game by keeping to try to put a cross in an already occupied cell.

X. Closing the Loop

This section shows a portion of the definition of the executable semantics of the AsmM metamodel itself by using the ASM-based semantic framework outlined in Sect. IV. We apply the semantic hooking approach on a small portion of the AsmM metamodel concerning the interpretation of the ASM update-rule. In this way, we close the in-the-loop integration between the formal method (ASM) and the MDE framework (EMF), as depicted in Fig. 11.

A. AsmM semantics

We have to specify, in general, an ASM Γ_{AsmM} (i.e. a model conforming to the AsmM metamodel) containing declarations of functions and domains (the signature) and the behavioral semantics of the AsmM metamodel itself in terms of ASM transition rules.

ASM rule constructors are represented in the AsmM metamodel by subclasses of the class `Rule`. Fig. 12 shows a subset of basic forms of a transition rule under the class hierarchy rooted by the class `BasicRule`: update-rule, conditional-rule, skip, do-in-parallel (block-rule), extend, etc.

Listing 5 reports a fragment Γ_{AsmM} in AsmetaL notation, for the interpretation of an ASM update-rule. It contains domains and function declarations induced from the AsmM metaclasses themselves for static/structural concepts (terms, rule constructors, etc.). Further domains and functions are introduced to denote run-time concepts like locations, values, updates, etc., according to the theoretical definitions given in [52] to construct the *run* of the ASM model under simulation.

A supporting execution engine has to keep the current state of the ASM model and, on request, evaluates the values of terms and computes (and applies) the update set to obtain the next state. To this purpose, an abstract domain `Value` and its sub-domains are introduced to denote all possible values of ASM terms. The function `eval` computes the value for every term (expression) in the current ASM state. The abstract domain `Location` represents the ASM concept of basic object containers (memory units), named *locations*, abstracting from particular memory addressing and object referencing mechanisms. Functions `signt` and `elements` denote, respectively, the pair of a function name f, which is fixed by the signature, and an optional argument $(v_1, \ldots, v_n)$, which is formed by a list of dynamic parameter values v_i of whatever type, forming a location. Two functions `currentState`, which represents the state of an ASM, and `updateSet`, which represents an update set, are used as tables to denote location-value pairs (loc, v) (updates) and are the basic units of state change. The `assignment` function maps location variables to their values for variable assignment in a state.

The very crucial task is that of computing at each step the ASM update set. To this purpose, there exist a rule `visit(RuleType R)` for every `RuleType` subclass of the `Rule` class of the AsmM. Given a rule `R`, the matching visit method is invoked accordingly to the type of `R` to obtain the update set of `R`. As example of such a kind of rule, Listing 5 reports the rule `r_visit` to compute the update set for an update-rule type.

One has also to define a function ι_{PT} which adds to Γ_{AsmM} the initialization necessary to make the ASM model executable. Any model transformation tool can be used to automatize the ι_{AsmM} mapping by retrieving data from a terminal model m and creating the corresponding ASM initial state in the target ASM model. A model transformation engine may implement such a mapping. Essentially, for each class instance of the terminal model, a static 0-ary function is created in the signature of the ASM model Γ_{AsmM} in order to initialize the domain corresponding to the underlying class. Moreover, class instances with their properties values and links are inspected to initialize the ASM functions declared in the ASM signature.

B. AsmM semantics validation

We applied the scenario-based approach for the validation of the semantics. We initially collected a set of AsmetaL examples representing all ASM constructs. In order to build an extensive set of scenario specifying the expected behavior of the system, instead of writing the scenario by hand, we simulated the original examples with AsmetaS (the simulator of AsmetaL models, see Sect. VII) itself, parsed the log files produced by AsmetaS in order to obtain valid scenario files in the Avalla syntax. Then we run the validator with the scenarios and the translation of the input examples by the semantic

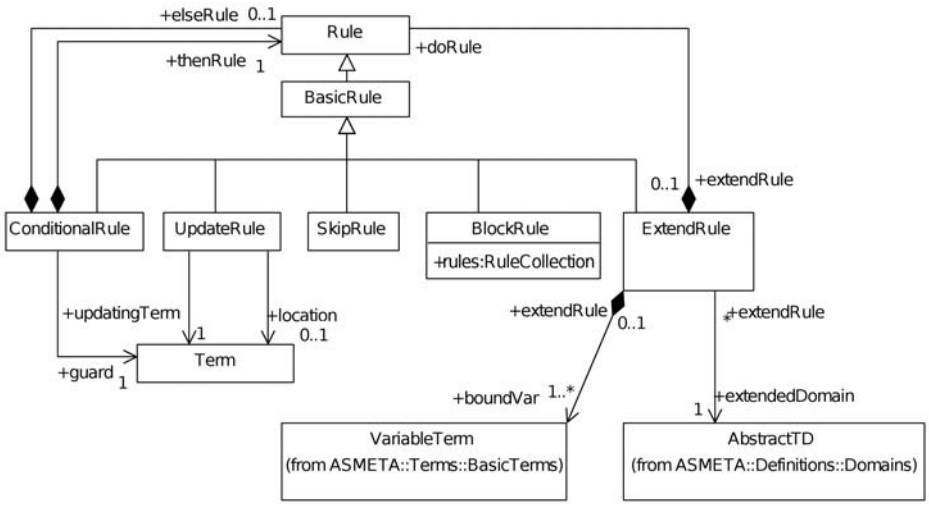


Fig. 12: A fragment of the AsmM metamodel for function terms and update-rules

Listing 5: Γ_{AsmM}

```
asm AsmM_hooking
signature:
// Signature induced from the AsmM metamodel:
 abstract domain Function
 abstract domain Term
 concrete domain VariableTerm subsetof Term
 concrete domain FunctionTerm subsetof Term
 concrete domain LocationTerm subsetof FunctionTerm
 ...
 abstract domain Rule
 concrete domain UpdateRule subsetof Rule
 ...
 controlled updatingTerm: UpdateRule −> TupleTerm
 controlled location: UpdateRule −> Term
 ...
// Signature for run−time concepts:
 abstract domain Value
 abstract domain Location
 controlled signt: Location −> Function
 controlled elements: Location −> Seq(Value)
 //Function for the evaluation of ASM terms
 static eval: Term −> Value
 ...
 //Functions for the current state of the ASM and memory updates
 controlled currentState: Location −> Value
 controlled updateSet: Location −> Value
 controlled assignment: VariableTerm −> Value
 ...
definitions:
 rule r_visit($r in UpdateRule) =
  let ( content = eval(updatingTerm($r))) in
    if isLocationTerm(location($r))
    then extend Location with $l do
            par
              signt($l):= funct(location($r))
              elements($l):= values(eval(arguments(location($r))))
              updateSet($l):= content
            endpar
    else if isVariableTerm(location($r))
          then assignment(location($r)):= content
          endif
    endif
  endlet
 ...
```

proposed above. In this way we have checked the conformance of AsmetaS with the semantics of the ASM as defined by the hooking function M.

XI. Conclusion and future directions

On the basis of our experience in developing the ASMETA toolset, we believe a formal method can gain benefits from the use of MDE automation means either for itself and toward the integration of different formal techniques and their tool interoperability. Indeed, the metamodel-based approach has the advantage of being suitable to derive from the same metamodel several artifacts (concrete syntaxes, interchange formats, APIs, etc.). They are useful to create, manage and interchange models in a model-driven development context, settling, therefore, a flexible infrastructure for tools development and interoperability. Moreover, metamodeling allows to establish a "global framework" to enable otherwise dissimilar languages (of possibly different domains) to be used in an inter-operable manner by defining precise *bridges* (or *projections*) among different domain-specific languages to automatically execute model transformations. That is in sympathy with the *SRI Evidential Tool Bus idea* [63], and can contribute positively to solve inter-operability issues among formal methods, their notations, and their tools.

On the other hand, the definition of a means for specifying rigorously the semantics of metamodels is a necessary step in order to develop formal analysis techniques and tools in the model-driven context. Along this research line, for example, we are tackling the problem of formally analyzing visual models developed with the SystemC UML Profile [64]. Formal ASM models obtained from graphical SystemC-UML models can potentially drive practical SoC model analysis like simulation, architecture evaluation and design exploration.

In conclusion, we believe MDE principles and technologies combined with formal methods elevate the current level of automation in system development and provide the widely demanded formal analysis support.

REFERENCES

[1] A. Gargantini, E. Riccobene, and P. Scandurra, "Integrating formal methods with model-driven engineering," in *The Fourth International Conference on Software Engineering Advances, ICSEA 2009, 20-25 September 2009, Porto, Portugal*, K. Boness, J. M. Fernandes, J. G. Hall, R. J. Machado, and R. Oberhauser, Eds. IEEE Computer Society, 2009, pp. 86–92.

[2] J. Bézivin, "On the Unification Power of Models," *Software and System Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

[3] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[4] "OMG. The Unified Modeling Language (UML), v2.1.2," `http://www.uml.org`, 2007.

[5] "OMG. The Model Driven Architecture (MDA Guide V1.0.1)," `http://www.omg.org/mda/`, 2003.

[6] S. J. and K. G., "Model-Integrated Computing," *IEEE Computer*, pp. 110–112, 1997.

[7] S. Cook, G. Jones, S. Kent, and A. C. Wills, *Domain-Specific Development with Visual Studio DSL Tools*. Addison Wesley, 2007.

[8] "Eclipse Modeling Framework (EMF)," `http://www.eclipse.org/emf/`.

[9] D. Gasevic, R. Lämmel, and E. V. Wyk, Eds., *Software Language Engineering, First International Conference, SLE 2008, Toulouse, France, September 29-30, 2008. Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 5452. Springer, 2009.

[10] C. Snook, F. Fritz, and A. Illisaov, "An EMF Framework for Event-B," in *Workshop on Tool Building in Formal Methods - ABZ Conference*, 2010.

[11] "The Maude System," `http://maude.cs.uiuc.edu/`.

[12] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, "ATL: a QVT-like transformation language," in *Proc. OOPSLA'06*. ACM, 2006, pp. 719–720.

[13] A. Agrawal, G. Karsai, S. Neema, F. Shi, and A. Vizhanyo, "The design of a language for model transformations," *Software and System Modeling*, vol. 5, no. 3, pp. 261–288, 2006.

[14] J. Fischer, M. Piefel, and M. Scheidgen, "A Metamodel for SDL-2000 in the Context of Metamodelling ULF," in *Proc. SAM'04*, 2004, pp. 208–223.

[15] M. Alanen and I. Porres, "A Relation Between Context-Free Grammars and Meta Object Facility Metamodels," Turku Centre for Computer Science, Tech. Rep., 2003.

[16] M. Wimmer and G. Kramler, "Bridging grammarware and modelware," in *Proc. of the 4th Workshop in Software Model Engineering (WiSME'05)*, Montego Bay, Jamaica, 2005.

[17] T. Gjøsæter, I. F. Isfeldt, and A. Prinz, "Sudoku - a language description case study," in *Proc. SLE'08*, 2008, pp. 305–321.

[18] "Abstract State Machines tools," `http://www.eecs.umich.edu/gasm/tools.html`.

[19] Y. Gurevich and B. Rossman and W. Schulte, "Semantic Essence of AsmL," Microsoft Research Technical Report MSR-TR-2004-27, March 2004 .

[20] A. Slissenko and P. Vasilyev, "Simulation of timed abstract state machines with predicate logic model-checking," *J. UCS*, vol. 14, no. 12, pp. 1984–2006, 2008.

[21] A. Gargantini, E. Riccobene, and P. Scandurra, "Deriving a textual notation from a metamodel: an experience on bridging Modelware and Grammarware," in *3M4MDA'06 workshop at the European Conference on MDA*, 2006.

[22] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende, "Derivation and refinement of textual syntax for models," in *ECMDA-FA*, 2009.

[23] F. Jouault, J. Bézivin, and I. Kurtev, "TCS: a DSL for the specification of textual concrete syntaxes in model engineering." in *Proceedings of the fifth international conference on Generative programming and Component Engineering (GPCE'06)*, 2006.

[24] S. Efftinge, "oAW xText - A framework for textual DSLs," in *Workshop on Modeling Symposium at Eclipse Summit*, 2006.

[25] P.-A. Muller, F. Fondement, F. Fleurey, M. Hassenforder, R. Schneckenburger, S. Gérard, and J.-M. Jézéquel, "Model-driven analysis and synthesis of textual concrete syntax," *Software and System Modeling*, vol. 7, no. 4, pp. 423–441, 2008.

[26] "OMG, Human-Usable Textual Notation, v1.0. Document formal/04-08-01," `http://www.uml.org/`.

[27] D. Hearnden, K. Raymond, and J. Steel, "Anti-Yacc: MOF-to-text," in *Proc. of EDOC*, 2002, pp. 200–211.

[28] M. Möller, E.-R. Olderog, H. Rasch, and H. Wehrheim, "Integrating a formal method into a software engineering process with UML and Java," *Form. Asp. Comput.*, vol. 20, no. 2, pp. 161–204, 2008.

[29] J. Armstrong, "Industrial integration of graphical and formal specifications," *J. of Systems and Software*, vol. 40, no. 3, pp. 211–225, 1998.

[30] A. Idani, J.-L. Boulanger, and L. P. 0002, "A generic process and its tool support towards combining uml and b for safety critical systems," in *Proc. CAINE*, 2007, pp. 185–192.

[31] T. Zhang, F. Jouault, J. Bézivin, and J. Zhao, "A MDE Based Approach for Bridging Formal Models," in *TASE '08.* IEEE Computer Society, 2008, pp. 113–116.

[32] Y. Sun, Z. Demirezen, F. Jouault, R. Tairas, and J. Gray, "A model engineering approach to tool interoperability," in *SLE*, 2008, pp. 178–187.

[33] P.-A. Muller, F. Fleurey, and J.-M. Jezequel, "Weaving Executability into Object-Oriented Meta-Languages," in *Proc. MODELS*, 2005.

[34] M. Soden and H. Eichler, "Towards a model execution framework for Eclipse," in *Proc. of the 1st Workshop on Behavior Modeling in Model-Driven Architecture*. ACM, 2009.

[35] J. E. Rivera, E. Guerra, J. de Lara, and A. Vallecillo, "Analyzing rule-based behavioral semantics of visual modeling languages with maude," in *SLE*, ser. Lecture Notes in Computer Science, D. Gasevic, R. Lämmel, and E. V. Wyk, Eds., vol. 5452. Springer, 2008, pp. 54–73.

[36] K. Chen, J. Sztipanovits, and S. Neema, "Toward a semantic anchoring infrastructure for domain-specific modeling languages," in *EMSOFT*, 2005, pp. 35–43.

[37] ——, "Compositional specification of behavioral semantics," in *DATE*, 2007, pp. 906–911.

[38] D. Di Ruscio, F. Jouault, I. Kurtev, J. Bézivin, and A. Pierantonio, "Extending AMMA for Supporting Dynamic Semantics Specifications of DSLs," LINA, Tech. Rep. 06.02, 2006.

[39] M. Anlauff, "XASM - An Extensible, Component-Based ASM Language," in *Proc. of Abstract State Machines*, 2000, pp. 69–90.

[40] "Atlantic XASM Zoo," `http://www.emn.fr/z-info/atlanmod/index.php/Xasm/`, 2001.

[41] D. A. Sadilek and G. Wachsmuth, "Using grammarware languages to define operational semantics of modelled languages," in *TOOLS (47)*, 2009, pp. 348–356.

[42] A. Gargantini, E. Riccobene, and P. Scandurra, "A semantic framework for metamodel-based languages," *Journal of Automated Software Engineering*, vol. 16, no. 3-4, pp. 415–454, 2009.

[43] A. Carioni, A. Gargantini, E. Riccobene, and P. Scandurra, "Exploiting the ASM method for Validation & Verification of Embedded Systems," in *Proc. of ABZ'08, LNCS 5238*. Springer, 2008, pp. 71–84.

[44] E. Riccobene and P. Scandurra, "An executable semantics of the SystemC UML profile," in *ABZ 2010*, ser. LNCS, M. F. et al., Ed., vol. 5977, 2010, pp. 75–90.

[45] E. Riccobene, P. Scandurra, S. Bocchio, A. Rosti, L. Lavazza, and L. Mantellini, "SystemC/C-based model-driven design for embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 8, no. 4, 2009.

[46] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages," *Software: Practice and Experience*, vol. 39, no. 15, pp. 1253 – 1292, October 2009.

[47] J. Bézivin, "In Search of a Basic Principle for Model Driven Engineering," *CEPIS, UPGRADE, The European Journal for the Informatics Professional*, vol. V, no. 2, pp. 21–24, 2004. [Online]. Available: http://www.upgrade-cepis.org/issues/2004/2/up5-2Bezivin.pdf

[48] "Textual Editing Framework." `http://www2.informatik.hu-berlin.de/sam/meta-tools/tef`, 2009.

[49] "openArchitectureware website," `www.openarchitectureware.org`, 2009.

[50] D. Harel and B. Rumpe, "Meaningful modeling: What's the semantics of "semantics"?" *IEEE Computer*, vol. 37, no. 10, pp. 64–72, 2004.

[51] E. Börger, "The ASM method for system design and analysis. A tutorial introduction," in *Frontiers of Combining Systems, 5th International Workshop, FroCoS 2005, Vienna, Austria, September 19-21, 2005, Proceedings*, ser. Lecture Notes in Computer Science, B. Gramlich, Ed., vol. 3717. Springer, 2005, pp. 264–283.

[52] E. Börger and R. Stärk, *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.

[53] A. Gargantini, E. Riccobene, and P. Scandurra, "Metamodelling a Formal Method: Applying MDE to Abstract State Machines," DTI Dept., University of Milan, Tech. Rep. 97, 2006.

[54] ——, "Ten reasons to metamodel ASMs," in *Dagstuhl Workshop on Rigorous Methods for Software Construction and Analysis, LNCS Festschrift*. Springer, 2007.

[55] "The Abstract State Machine Metamodel website," http://asmeta.sf.net/, 2006.

[56] E. Riccobene and P. Scandurra, "Towards an Interchange Language for ASMs," in *Abstract State Machines. Advances in Theory and Practice*, ser. LNCS 3052, W. Zimmermann and B. Thalheim, Eds. Springer, 2004, pp. 111 – 126.

[57] "OMG. Object Constraint Language (OCL), v2.0 formal/2006-05-01," 2006.

[58] A. Gargantini, E. Riccobene, and P. Scandurra, "A Metamodel-based Language and a Simulation Engine for Abstract State Machines," *J. UCS*, vol. 14, no. 12, pp. 1949–1983, 2008.

[59] ——, "Model-driven language engineering: The ASMETA case study," in *International Conference on Software Engineering Advances, ICSEA*. IARIA: Published by IEEE Computer Society, 2008, pp. 373–378.

[60] P. Arcaini, A. Gargantini, and E. Riccobene, "AsmetaSMV: A way to link high-level ASM models to low-level NuSMV specifications," in *ABZ 2010*, ser. LNCS, M. F. et al., Ed., vol. 5977, 2010, pp. 61–74.

[61] E. Börger, M. J. Butler, J. P. Bowen, and P. Boca, Eds., *Abstract State Machines, B and Z, First International Conference, ABZ 2008, London, UK, September 16-18, 2008. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5238. Springer, 2008.

[62] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, ser. LNCS, vol. 2404. Copenhagen, Denmark: Springer, July 2002.

[63] J. M. Rushby, "Harnessing Disruptive Innovation in Formal Verification," in *Proc. SEFM*, 2006, pp. 21–30.

[64] A. Gargantini, E. Riccobene, and P. Scandurra, "Model-driven design and ASM-based analysis of embedded systems," in *Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation*, L. Gomes and J. M. Fernandes, Eds. Norwell, MA, USA: IGI Global, 2009, pp. 24–54.

APPENDIX A
BASIC PETRI NETS SEMANTICS

A concrete example is here provided by applying the semantic hooking technique to a possible metamodel for the Petri net formalism. The results of this activity are executable semantic models for Petri nets which can be made available in a model repository either in textual form using AsmetaL or also in abstract form as instance model of the AsmM metamodel.

Fig. 13 shows the metamodel for the basic Petri net formalism. It describes the static structure of a net consisting of places and transitions (the two classes `Place` and `Transition`), and of directed arcs (represented in terms of associations between the classes `Place` and `Transition`) from a place to a transition, or from a transition to a place. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. Places may contain (see the attribute `tokens` of the `Place` class) any non-negative number of tokens, i.e. infinite capacity. Moreover, arcs are assumed to have a unary weight. Fig. 14 shows (using a graphical concrete syntax) an example of Petri net (with its initial marking) that can be intended as instance (a terminal model) of the Petri net metamodel in Fig 13.

According to the semantic hooking approach, first we have to specify an ASM Γ_{PT} (i.e. a model conforming to the AsmM metamodel) containing only declarations of functions and domains (the signature) and the behavioral semantics of the Petri net metamodel in terms of ASM transition rules. Listing 6 reports a possible Γ_{PT} in AsmetaL notation. It introduces abstract domains for the nets themselves, transitions, and

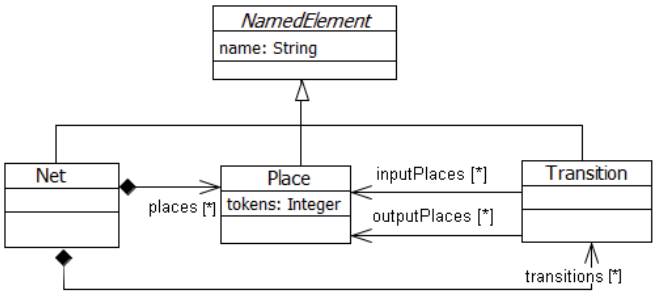


Fig. 13: A metamodel for basic Petri nets

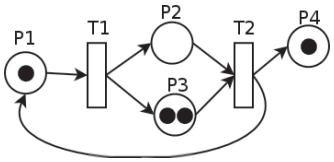


Fig. 14: A basic Petri net with its initial marking

places. The static function *isEnabled* is a predicate denoting whether a transition is enabled or not. The behavior of a generic Petri net is provided by two rules: *r_fire*, which express the semantics of token updates upon firing of transitions, and *r_PetriNetReact*, which formalizes the firing of a non-deterministic subset of all enabled transitions. The main rule executes all nets in the *Net* set.

One has also to define a function ι_{PT} which adds to Γ_{PT} the initialization necessary to make the ASM model executable. Any model transformation tool can be used to automatize the ι_{PT} mapping by retrieving data from a terminal model m and creating the corresponding ASM initial state in the target ASM model. We adopted the ATL model transformation engine to implement such a mapping. Essentially, for each class instance of the terminal model, a static 0-ary function is created in the signature of the ASM model Γ_{PT} in order to initialize the domain corresponding to the underlying class. Moreover, class instances with their properties values and links are inspected to initialize the ASM functions declared in the ASM signature. For example, for the Petri net m_{PT} shown in Fig. 14, the ι_{PT} mapping would automatically add to the original Γ_{PT} the initial state (and therefore the initial marking) leading to the final ASM model shown in Listing 7. The initialization of the abstract domains `Net`, `Transition`, and `Place`, and of all functions defined over these domains, are added to the original Γ_{PT}.

Listing 6: Γ_{PT}

```
asm PT_hooking
signature:
 abstract domain Net
 abstract domain Place
 abstract domain Transition

 //Functions on Net
   controlled places: Net −> Powerset(Place)
   controlled transitions: Net −> Powerset(Transition)

 //Functions on Place
   controlled tokens : Place −> Integer

 //Functions on Transition
   controlled inputPlaces: Transition −> Powerset(Places)
   controlled outputPlaces: Transition −> Powerset(Places)
   static isEnabled : Transition −> Boolean

definitions:
  function isEnabled ($t in Transition) =
            (forall $p in inputPlaces($t) with tokens($p)>0)

  rule r_fire($t in Transition) =
    seq
      forall $i in inputPlaces($t) do tokens($i) := tokens($i)−1
      forall $o in outputPlaces($t) do tokens($o) := tokens($o)+1
    endseq

  rule r_PetriNetReact($n in Net) =
   choose $transSet in Powerset(Transitions($n))
         with (forall $t in $transSet with isEnabled($t)) do
    iterate let ($t = chooseOne($transSet)) in par
        remove($t,$transSet)
        if isEnabled($t) then r_fire[$t] endif
      endpar endlet

//Run all Petri nets
    main rule r_Main = forall $n in Net do r_PetriNetReact[$n]
```

Listing 7: $\iota_{PT}(\Gamma_{PT}, m_{PT})$

```
asm PT_hooking
signature:
  ....
  static myNet: Net
  static P1,P2,P3,P4:Place
  static t1,t2:Transition
  ....
default init s0:
 //Functions on Net
 function places($n in Net) = at({myNet −> {p1,p2,p3,p4}},$n)
 function transitions($n in Net) = at({myNet −> {t1,t2}},$n)

 //Functions on Place (the "initial marking")
 function tokens($p in Places) =
                  at({p1−>1,p2−>0,p3−>2,p4−>1},$p)

 //Functions on Transition
 function inputPlaces($t in Transition) =
                  at({t1−>p1,t2−>{p2,p3}},$t)
 function outputPlaces($t in Transition) =
                  at({t1−>{p2,p3},t2−>{p4,p1}},$t)
```

Appendix B
ASM specification for Tic-Tac-Toe

Listing 8: $\Gamma_{Tic-Tac-Toe}$ - the complete signature

```
asm Tictactoe
signature:
 //For representing a board
 enum domain Skind = {CROSS|NOUGHT|EMPTY}
 domain Square subsetof Integer
 domain Row subsetof Integer
 domain Three subsetof Integer
 static squaresInRow: Prod(Row,Three) −> Square
 controlled symbol: Square −> Skind
 //For managing the game
 enum domain Finalres = {PLAYERX|PC|TIE}
 enum domain Status = {TURNX|CHECKX|TURNPC|CHECKPC
                       |GAMEOVER}
 monitored playerX:Square // move of X
 controlled status: Status
 controlled whoWon: Finalres
 derived noSquareLeft : Boolean
 derived hasThreeOf: Prod(Row,Skind) −> Boolean
 //For PC strategies
 domain Count subsetof Integer
 controlled count: Count
 derived openingPhase: Boolean
 controlled lastMoveX: Square
 static isCorner: Square −> Boolean
 static isEdge: Square −> Boolean
 static isCenter: Square −> Boolean
 derived hasTwo: Row −> Boolean
 static opposite: Square −> Square

definitions:
 domain Square = {1..9}
 domain Count = {0..9}
 domain Row = {1..8}
 domain Three = {1..3}

 function squaresInRow($r in Row,$x in Three) =
 if $r = 1 then if $x = 1 then 1 else if $x = 2 then 2 else 3 endif endif
 else if $r = 2 then if $x = 1 then 4 else if $x = 2 then 5 else 6 endif endif
 else if $r = 3 then if $x = 1 then 7 else if $x = 2 then 8 else 9 endif endif
 else if $r = 4 then if $x = 1 then 1 else if $x = 2 then 4 else 7 endif endif
 else if $r = 5 then if $x = 1 then 2 else if $x = 2 then 5 else 8 endif endif
 else if $r = 6 then if $x = 1 then 3 else if $x = 2 then 6 else 9 endif endif
 else if $r = 7 then if $x = 1 then 1 else if $x = 2 then 5 else 9 endif endif
 else if $x = 1 then 3 else if $x = 2 then 5 else 7 endif endif
 endif endif endif endif endif endif endif

 function noSquareLeft = not(exist $s in Square with symbol($s)=EMPTY)

 function hasThreeOf ($r in Row, $symbol in Skind) =
  (symbol(squaresInRow($r,0)) = $symbol) and
  (symbol(squaresInRow($r,0)) = symbol(squaresInRow($r,1))) and
(symbol(squaresInRow($r,0)) = symbol(squaresInRow($r,2)))

 function openingPhase = count=0 or count=1

 function isCenter($s in Square) = $s =5
 function isCorner($s in Square) = $s =1 or $s=3 or $s=7 or $s=9
 function isEdge($s in Square) = $s =2 or $s =4 or $s=6 or $s=8

 //return true iff $r has two equal symbols and the third square is EMPTY
 function hasTwo($r in Row) =
   (exist $i1 in Three, $i2 in Three, $i3 in Three
    with ($i1!=$i2 and $i1!=$i3 and $i2!=$i3 and
     (symbol(squaresInRow($r,$i1)) = symbol(squaresInRow($r,$i2))) and
     (symbol(squaresInRow($r,$i1)) != EMPTY) and
     (symbol(squaresInRow($r,$i3)) = EMPTY)))

 function opposite($s in Square) =
    if $s=1 then 9 else if $s=3 then 7 else if $s=7 then 3
    else if $s=9 then 1 endif endif endif endif
```

Listing 9: $\Gamma_{Tic-Tac-Toe}$ transition rules

```
//A very naive player: choose an empty square and mark it.
 rule r_naive_strategy ($symbol in Skind)=
  choose $s in Square with symbol($s)=EMPTY
  do symbol($s):= $symbol

 rule r_playACorner($symbol in Skind) =
  choose $s in Square with (symbol($s)=EMPTY and isCorner($s))
  do symbol($s):= $symbol

 //Opening strategy
 rule r_opening_strategy ($symbol in Skind)=
   if (count=0) then r_playACorner[$symbol]
   else if symbol(5) = EMPTY then symbol(5):=$symbol //play the center
   else r_playACorner[$symbol] //we play a corner
   endif endif

 //Mark with $symbol the last empty square within row $r
 rule r_markLastEmpty ($r in Row, $symbol in Skind) =
   choose $x in {1,2,3} with symbol(squaresInRow($r,$x))=EMPTY
    do symbol(squaresInRow($r,$x)) := $symbol

  //Draw strategy (with no fork creation/block)
 rule r_draw_strategy ($symbol in Skind) =
  choose $wr in Row with hasTwo($wr)
  do r_markLastEmpty[$wr,$symbol] //1. Win or 2. Block
  ifnone
    if (symbol(5)=EMPTY) then symbol(5):=$symbol //3. Center
    else if (isCorner(lastMoveX) and symbol(opposite(lastMoveX))=EMPTY)
    then symbol(opposite(lastMoveX)):= $symbol //4. Opposite corner
    else choose $s in Square with (symbol($s)=EMPTY and isCorner($s))
           do symbol($s):= $symbol //5. Empty Corner
           ifnone r_naive_strategy[$symbol] //6. Empty edge
    endif endif

 //Computer strategy selection
 rule r_tryStrategy ($symbol in Skind) =
  if openingPhase then r_opening_strategy[$symbol]
  else r_draw_strategy[$symbol] endif

 rule r_movePC = par r_tryStrategy[NOUGHT]
                     count := count + 1
                     status := CHECKPC
                 endpar

 rule r_movePlayerX = if symbol(playerX)= EMPTY
                      then par symbol(playerX):= CROSS
                                count := count + 1
                                lastMoveX := playerX
                                status := CHECKX
                           endpar
                      else status := TURNX endif

 rule r_checkForAWinner($symbol in Skind) =
          //GAME OVER WITH A WINNER?
          if (exist $r in Row with hasThreeOf($r,$symbol)) then
           par status := GAMEOVER
               if $symbol = CROSS then whoWon:= PLAYERX
               else whoWon:= PC endif
           endpar
                else if ( noSquareLeft ) //GAME TIE?
               then par status := GAMEOVER whoWon := TIE endpar
               else if $symbol = CROSS then status:= TURNPC
                    else status:= TURNX endif endif endif

  main rule r_Main = if status = TURNX then r_movePlayerX[]
          else if status = CHECKX then r_checkForAWinner[CROSS]
          else if status = TURNPC then r_movePC[]
          else if status = CHECKPC then r_checkForAWinner[NOUGHT]
          endif endif endif endif
```

Video Notation (ViNo): A Formalism for Describing and Evaluating Non-sequential Multimedia Access

Anita Sobe, Laszlo Böszörmenyi, Mario Taschwer
Institute of Information Technology
Klagenfurt University
Austria
{anita, laszlo, mt}@itec.uni-klu.ac.at

***Abstract*—The contributions of this paper are threefold: (1) the extensive introduction of a formal Video Notation (ViNo) that allows for describing different multimedia transport techniques for specifying required QoS; (2) the application of this formal notation to analyzing different transport mechanisms without the need of detailed simulations; (3) further application of ViNo to caching techniques, leading to the introduction of two cache admission policies and one replacement policy supporting non-sequential multimedia access.**
The applicability of ViNo is shown by example and by analysis of an existing CDN simulation. We find that a pure LRU replacement yields significantly lower hit rates than our suggested popularity-based replacement. The evaluation of caches was done by simulation and by usage of ViNo.

***Keywords*-Multimedia Formalism; QoS; Caching; CDN; Non-sequential Multimedia**

I. Introduction

A. Motivation

In the end of 1895 the Lumière brothers presented the first moving pictures in France (Lyon and Paris). They stored the movie as a sequence of images on a perforated celluloid tape. They were able to record and play this back at such a speed that the viewers had the impression of - more or less - continuous movement. This was the birth of the movie. People were so much fascinated from moving pictures that on the very first posters for movie performances we cannot find a title, author or the like - people just went to see moving pictures; whatever was showed.

Since then we have got used to the idea that a movie is a long sequence of images recorded and played back at a more or less constant speed. Even though this basic principle is still valid, the uncritical usage of this paradigm causes a lot of unnecessary difficulties in modern video access. Usage scenarios are rapidly changing. We have reported [2] how arthroscopic videos are used. The camera plays a central role in this kind of surgery and the recorded videos are highly interesting for research and education. The users - medical doctors - are interested to find special situations in a large archive of visually very similar recordings, e.g., the usage of surgery equipment of a given type, in a special pathological situation. They might be interested in comparing similar scenes, watch them in parallel and create sub-sequences or even single images from them. Several persons may do this in cooperation, in a distributed manner. Such usage patterns are obviously very different from that of the first viewers of Lumières' movies. In the following non-exhaustive list, we summarize the main aspects of the current situation:

1) Virtually everybody can create and consume videos.
2) The length of movies available on the Internet varies from a few seconds up to several hours.
3) Besides entertainment, professional use is gaining importance, e.g., in medicine, news production, traffic control and so on.
4) Both in entertainment and in professional usage, people are often only interested in a small fraction of long video sequences.

We conclude that we could gain a lot if we regarded videos as non-sequential, direct access media. To put it in another way: It is time to switch from the tape to the disk paradigm. Or again in other words: Instead of enforcing users to be passive viewers of movies it would be desirable to enable them to become active *composers* of video presentations.

B. Compositions

Instead of offering prefabricated long, sequential videos, we propose to offer a set of elementary *video units*, which can be composed with the help of sequential and parallel composition operators to arbitrary Video Notation (ViNo) *compositions*. A unit itself is regarded as an atomic composition. The result of a composition operation is also a composition. Thus, compositions can be constructed from video units by repeated application of composition operators.

We do not constrain the exact semantics of a video unit. It could be a single bit or byte, a video frame, or a semantically meaningful, short clip. By short we mean that the download time is short enough not to justify streaming. Streaming should rather be expressed as a composition (see below).

A given video can be logically described by several different compositions. It can also be physically decomposed, in order to get physically materialized units. We assume that for a given video usually several logical compositions exist, but only one physical decomposition exists. The problem of finding a physical decomposition of maximal unit size that is compatible with a given set of logical compositions is the subject of related research, but out of scope of this paper. In the following

we assume that compositions rely on a given suitable physical decomposition.

C. Quality of Service (QoS) constraints on compositions

Compositions may be subject to certain QoS constraints to describe video processing requirements and properties using ViNo. For example, we may require that the processing delay for video units must not exceed a certain maximum. Or a given network bandwidth must be available when transmitting units according to a given ViNo composition. It is thus possible to describe a video streaming scenario as a sequential composition of bits with a bandwidth requirement of 1 Mbps, or as a sequential composition of video frames with an average throughput of 25 frames/sec. A video playback scenario could be expressed as a parallel composition of some clips with a maximal start-up delay of 500 ms. We can describe both required and provided QoS using the same formalism.

ViNo compositions may also describe pipelined video processing by an appropriate combination of sequential and parallel operators according to a certain number of stages. Each stage in the pipeline represents a buffering element. Classical video streaming can be described as a pipelined sequential composition, constrained by bandwidth and start-up delay.

More generally, ViNo can be used to express temporal relations between video units for the purpose of: (1) video presentation requests, (2) video delivery execution plans, and (3) video access methods.

D. Putting it together

Let us consider a simple example to put the elements together. Assume that a ski-jumping video has already been decomposed into six meaningful, short clips. The clips show two essential moments (jump-off and touch-down) for three athletes. The user would like to see the two clips belonging to the same jumper sequentially, but the three clip pairs in parallel. We assume that a video player capable of such presentation modes is available. The user creates a ViNo composition expressing her video presentation request (see formal examples below) with the help of some appropriate GUI. Now, the video transportation system transforms the request to an execution plan, which is again represented by a ViNo composition. For instance, if all clips are stored on the same network node then the clips must be downloaded sequentially (as we assume that clips are video units, which are handled atomically by the video transportation system). However, their order should be interleaved: first the jump-off clips of all three athletes and then the touch-down clips). On the other hand, if the clips happen to be distributed in the delivery network in an optimal way, i.e. the clips belonging to the same jumper are stored on the same network node and clips belonging to different athletes are stored on different nodes, with equal and minimal distance to the client, then the execution plan is actually represented by the same ViNo composition as the request. A good video transportation system obviously strives for finding such optimal placement of clips for popular requests. In any case the execution plan must, of course, fulfill the requested QoS constraints. If this is not possible, a good implementation is supposed to take certain adaptation actions such as replicating video units. These actions can again be expressed as ViNo composition transformations.

E. What are ViNo compositions good for?

We see two main advantages:

1) Flexibility: If we get rid of the dictatorship of the long, sequential, and continuous streams, then we gain a lot of freedom in the handling of video systems. By using ViNo we make any video delivery system programmable in a certain sense. This apparently rather theoretical point should not be underestimated. The success of digital computers relies exactly on this kind of flexibility. Analog computers had a number of advantages over the digital ones in solving differential equations. They were faster and more precise – but less flexible. No need to say who won the race between analog and digital computers. Note that in the early 1960s, this was not yet obvious.

2) Simplification: This is the actual topic of this paper. Making explanations and descriptions simpler had always been a driving power in science. It is not only a matter of costs – a simple solution is usually cheaper than a complex one, but a simple description is also easier to understand and therefore less error-prone. On the other hand, if something is getting very complicated then this is usually a sign of missing understanding.

F. Using ViNo for deriving delay bounds

In the first part of this paper we introduce syntax and semantics of ViNo and the associated QoS description language in detail. In describing QoS constraints we rely on QL, as defined by Blair and Stefani [3]. In the second part, we show how to model Content Delivery Systems (CDNs) with the help of ViNo. We can perform delay estimations of arbitrary complex compositions in a recursive manner. We use the CDNSim [4] simulator as a reference for evaluating our results. We already obtain good estimations using a rough model, which can easily be refined. Thus, we are able to estimate transport delay bounds of complex, distributed video delivery systems using a small set of ViNo expressions. The results can be sufficient to support system design decisions, thereby eliminating the need of sophisticated simulations. To the best of our knowledge, this is a unique achievement. ViNo expressions can easily be modified and extended. When creating the examples, we experienced indeed that we could not test all required situations using CDNSim. Modifying the simulation would have needed days – if not weeks – of work. Extending ViNo expressions, however, is a matter of hours or minutes (for an experienced user).

In the third part of the paper we use ViNo to experiment with some simple but novel video caching methods [1] based on units. An own prototype implementation serves as a reference. Also in this case ViNo yields suitable delay estimations

(under the assumption, of course, that QoS parameter estimations are correct). The prototype implementation is a first step towards a novel, self-organizing video delivery system, where ViNo compositions and decompositions play a central role. However, this system will not be discussed in this paper.

II. Related Work

QoS languages have been defined to help a user or application to specify requirements and to formally define actions for recovery if the given requirements are not met. In [5] the authors give an overview and classification of QoS languages, which are categorized into user-layer QoS, application-layer QoS and resource-layer QoS. Examples are INDEX [6] as an expressive user-layer QoS language that helps translating the user's preferences to more specific network-related QoS. Application-layer QoS regards parameters such as frame rate and frame resolution. The authors of HQML [7] took advantage of XML for allowing developers to specify their own multimedia-related tags. Another example is QML [8], which is an object-oriented CORBA-based QoS language that allows for QoS hierarchies and reusability. Resource-layer QoS languages such as RSL [9] concentrate on resource management and allocation.

However, ViNo's aim is not to define a new QoS paradigm. ViNo uses QoS languages, in particular QL [3], in order to clarify QoS requirements. MMC# (see [10], [11]), a QL based QoS extension of C#, provides automatic QoS requirement formulation checking. A ViNo-compliant application might take advantage of that by being implemented in MMC#. However, calculations done with ViNo cannot be performed with any of the examples given in [5] nor with MMC#.

In contrary to QoS languages, an XML-based language exists that handles the presentation of autonomous media objects, namely SMIL [12]. SMIL is a description language for synchronizing different media channels like sound, video and text in a SMIL player. Although ViNo might also be used to describe multimedia presentations without the XML overhead of SMIL, ViNo's main strength is its general applicability to the analysis of flexible transport mechanisms and related calculations.

ViNo was designed to be able to compare existing multimedia transport technologies, such as Client/Server, Content Delivery Networks (CDNs) or P2P download and streaming, to more flexible approaches. In this context, non-sequential multimedia access patterns open new possibilities of video services and require new ways of transport.

As described in [1], a first step in the direction of non-sequential media was investigated by Zhao et al. [13]. The authors define "non-linear" media as video consisting of several parallel branches. The streaming system maintains a channel per branch. The authors observed as major problem that there is no possibility to explore bandwidth reduction by sharing connections, because it is not known if the client will choose the branch just transmitted in advance. Nevertheless, the authors showed that some hints regarding the client branch selection lead to remarkable server bandwidth and client data overhead reduction. However, the possible paths are predefined and limited compared to the possibilities offered by our video unit model.

Videos are considered as too large with respect to size to be cached completely. A lot of research has been done on partial caching. Generally, the idea of caching only parts of a video supports our non-sequential media model.

In [14], a detailed overview of different caching strategies is given. Prefix caching and segment-based caching are most closely related to our work. A prefix may be fixed (e.g., the first 10 minutes of a video) or dynamic (for every video a proper prefix size is defined), see also [15]. Segment-based caching increases the number of cached segments of a video based on popularity measurements. Segments may be uniformly sized or grow exponentially [16].

The authors of [17] propose a caching algorithm for streaming media based on a measured popularity distribution of segments. Considering fixed-sized segments of one second, they observed that the popularity of segments of a single video (*internal popularity*) follows a k-transformed Zipf-like distribution (for $k_x = 10$ and $k_y > 200$). Hence the first segments of a video are most popular, so the proposed caching policy prioritizes prefix caching. However, we cannot expect that non-sequential media access exhibits the same internal popularity pattern. A user may not be aware of which unit is the "beginning" of a video.

Another segment-based caching mechanism aims to support interactive "jumps" in a video stream [18]. The authors introduce a basic interleaved segment caching (BISC) policy, which disperses prefetched segments uniformly over the video length to reduce response time for jump requests at the cost of a reduced hit rate for sequential access. When the client jumps to an uncached segment the cache delivers the closest cached segment. Since segments are likely to be accessed sequentially after a jump, BISC was extended to a dynamic interleaved segment caching (DISC) policy, which dynamically selects an interleaved or continuous segment caching strategy based on observed client access patterns. However, the authors assume, based on their analysis of a real RTSP workload in the year 2004, that video segments will be accessed sequentially in most cases. In our video unit approach we refrain from this restriction.

III. The ViNo formalism

As described in the Introduction ViNo is based on so called *compositions*. Its general syntax is given by the following definition (the EBNF specification appears in the appendix).

Definition 1: A *composition* is an expression defined inductively by these rules:

1) A single video unit is a composition.
2) Let $c_1, c_2, ..., c_n$ with $n \geq 2$ be compositions, which have already been defined. Then, the following expressions are compositions, too:
   a) $[c_1 \,||\, c_2 \,||\, \ldots \,||\, c_n]$ is called a *parallel* composition.
   b) $(c_1 \leftarrow_{Q_1} c_2 \leftarrow_{Q_2} \cdots \leftarrow_{Q_{n-1}} c_n)$ is called a *sequential* composition. A symbol Q_i, where

$i = 1, \ldots, n-1$, represents an optional QoS parameter and may be omitted.

Throughout this paper, u_i $(i \geq 1)$ always denotes a single video unit. The brackets or parentheses of a parallel or a sequential composition c, respectively, may be omitted if c does not appear as proper subexpression of a composition. So both $[u_1 \,||\, u_2]$ and $u_1 \,||\, u_2$ are valid compositions on their own, but $u_1 \,||\, u_2 \leftarrow u_3$ is not.

We define the semantics of ViNo in the context of video transmission, but analogous interpretations apply in other contexts as explained in the Introduction.

Definition 2: Semantics.

1) If $c = c_1 \,||\, c_2$ for some compositions c_1 and c_2, then the transport of c starts as soon as c_1 or c_2 starts, whatever is earlier; and it is finished when the transport of both c_1 and c_2 is completed.
2) If $c = c_1 \leftarrow_Q c_2$ then the transmission of c_2 must not start before the completion of c_1; the QoS predicate Q applies to the time period between completion of c_1 and completion of c_2.
3) The semantics of $c = c_1 \leftarrow_{Q_1} c_2 \leftarrow_{Q_2} c_3$ is defined as that of $(c_1 \leftarrow_{Q_1} c_2) \leftarrow_{Q_2} c_3$.

We consider two compositions c_1 and c_2 as *equivalent* if they lead to the same semantics according to Definition 2. We then write $c_1 = c_2$. It is easy to check that the following equations hold:

$$[c_1 \,||\, c_2] \,||\, c_3 = c_1 \,||\, [c_2 \,||\, c_3] \quad (1)$$

$$[c_1 \,||\, c_2] = [c_2 \,||\, c_1] \quad (2)$$

$$(c_1 \leftarrow c_2) \leftarrow c_3 = c_1 \leftarrow (c_2 \leftarrow c_3) \quad (3)$$

$$(c_1 \leftarrow_{Q_1} c_2) \leftarrow_{Q_2} c_3 = c_1 \leftarrow_{Q_1 \circ Q_2} (c_2 \leftarrow_{Q_2} c_3) \quad (4)$$

where c_1, c_2, c_3 are arbitrary compositions and $Q_1 \circ Q_2$ means a suitable combination of both QoS predicates Q_1 and Q_2, e.g. the sum if Q_1 and Q_2 refer to maximal delay. Note that according to (1) a parallel composition c is an associative binary operation, so the semantics of c is well defined. The same applies to sequential composition without QoS predicate.

Definition 3: The null unit u_0 is a video unit of length 0 (empty).

The null unit u_0 serves as a "dummy" composition (in a similar way as dummy targets are used to express side-effects in functional languages). The following properties apply:

- $u_0 \,||\, c_1 = c_1$
- $u_0 \leftarrow c_1 = c_1$
- $c_1 \leftarrow u_0 = c_1$

A. Simple examples

In order to show how the before mentioned definitions work, three artificial examples are created. All examples are based on the same video delivery system architecture. It consists of one origin server, four interconnected proxies and one client. We show how the transmission of six video units can be described using ViNo for different configurations with respect to unit placement. For sake of simplicity, QoS is postponed to the next section.

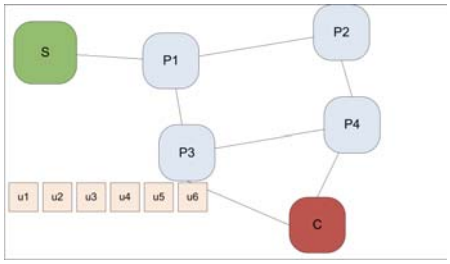


Fig. 1. Sample video delivery system with one origin server S, four proxies $P1 - P4$, and one client C, where all video units $u1 - u6$ are available at proxy $P3$.

As described in the Introduction a presentation request may be created using some appropriate GUI. The request of displaying video units 1–6 sequentially can be expressed as:

$$r = u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6$$

The actual transport of video units may differ from the presentation request. To keep it simple, we assume that all video units are downloaded to the client completely before play back starts. So the video delivery process can be decomposed into one or more *download stages* corresponding to a ViNo *transport description* s, followed by a *play back stage* equivalent to the presentation request r: $s \leftarrow r$.

Example 1. All video units are located at proxy $P3$ as shown in Fig. 1. The units will be downloaded sequentially to the client, corresponding to the ViNo transport description:

$$s_1 = u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6$$

By adding the play back stage $s_2 = r$ we obtain the complete video delivery description:

$$\begin{aligned} c = s_1 \leftarrow s_2 = s_1 \leftarrow r = \\ u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6 \\ \leftarrow u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6 \end{aligned}$$

Example 2. Three of the units are located on proxy $P3$ and the other three are located on proxy $P4$. Both proxies are direct neighbors of the client (Fig. 2). The download from $P3$ is described as $s_1 = u_1 \leftarrow u_2 \leftarrow u_3$ and the download from $P4$ is described as $s_2 = u_4 \leftarrow u_5 \leftarrow u_6$. There are two possibilities to combine these download stages to describe the overall video transport:

(1) The client downloads everything from $P3$ and afterwards everything from $P4$, resulting in the ViNo expression: $s_1 \leftarrow s_2 = (u_1 \leftarrow u_2 \leftarrow u_3) \leftarrow (u_4 \leftarrow u_5 \leftarrow u_6)$.

(2) While downloading everything from $P3$ the units are downloaded from $P4$ in parallel: $s_1 \,||\, s_2 = (u_1 \leftarrow u_2 \leftarrow u_3) \,||\, (u_4 \leftarrow u_5 \leftarrow u_6)$. The video transport expressed by this composition is finished when all video units have been transmitted. Note that there is no temporal relation between downloading units of s_1 and s_2, respectively. That is, u_2 can be received before or after u_5 by the client. However, a system designer may decide to synchronize the transport of s_1 and s_2;

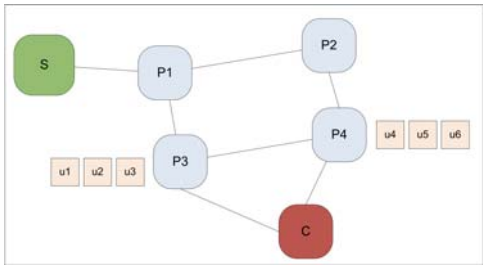


Fig. 2. Sample video delivery system with one origin server S, four proxies $P1 - P4$, and one client C, where the video units $u1 - u6$ are available at proxies $P3$ and $P4$ near the client.

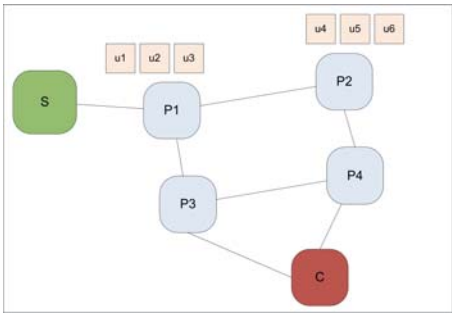


Fig. 3. Sample video delivery system with one origin server S, four proxies $P1 - P4$, and one client C, where the video units $u1 - u6$ are available at proxies $P1$ and $P2$, which are not directly connected to the client.

then the description specializes to $[u_1 \,||\, u_4] \leftarrow [u_2 \,||\, u_5] \leftarrow [u_3 \,||\, u_6]$.

As in example 1 the transport description involving the download stages s_1 and s_2 is followed by a play back stage $s_3 = r$.

Example 3. Three of the video units are located on proxy $P1$ and the other three units are located on proxy $P2$. None of these proxies is directly connected to the client (Fig 3), so units have to be replicated to proxies $P3$ or $P4$, respectively, before they are downloaded to the client. This results in 5 stages: the transport of 3 units from $P1$ to $P3$ (stage s_1), from $P2$ to $P4$ (stage s_2), from $P3$ to the client (stage s_3), and from $P4$ to the client (stage s_4); and finally, the play back of all 6 units at the client (stage $s_5 = r$).

Let us assume a pipelined transport where proxies $P3$ and $P4$ forward units immediately after receiving them from $P1$ or $P2$, respectively. For the sake of simplicity, let us further assume that the transmission times of all video units between adjacent network nodes and the play back duration of a single video unit are roughly equal to a certain time period t. Then the temporal evolution of the video delivery process can be represented by TABLE I. Consequently, the user has to wait 4 time slots until play back can start, and the entire video delivery process takes 10 time slots.

The table also helps creating the appropriate ViNo expression that describes the given video delivery scenario. Units within the same time slot are transmitted in parallel, units in different time slots are processed sequentially. The resulting

t	1	2	3	4	5	6	7	8	9	10
s_1	u_1	u_2	u_3							
s_2	u_4	u_5	u_6							
s_3		u_1	u_2	u_3						
s_4		u_4	u_5	u_6						
s_5					u_1	u_2	u_3	u_4	u_5	u_6

TABLE I
TEMPORAL EVOLUTION OF VIDEO DELIVERY SCENARIO OF EXAMPLE 3.

ViNo expression is:

$$[u_1 \,||\, u_4] \leftarrow [u_2 \,||\, u_5 \,||\, u_1 \,||\, u_4] \leftarrow [u_3 \,||\, u_6 \,||\, u_2 \,||\, u_5] \leftarrow [u_3 \,||\, u_6] \leftarrow u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4 \leftarrow u_5 \leftarrow u_6 \quad (5)$$

B. Introducing QoS

To specify a request a client has only to provide information about the required video units and whether these units have to arrive in order (e.g. at the player). For example, a user may express "I want to download units x and y, the order does not matter" as $u_x \,||\, u_y$. Note that this does not mean that the units have to be delivered in parallel. A user who wants to watch the units using a video player would be more specific: "I want to watch unit x and then unit y, and unit x has to arrive within the next 30 seconds". This request can be expressed as $u_0 \leftarrow_{D=30sec} u_x \leftarrow u_y$, where u_0 is the null unit needed only to express the required delay for unit x.

In the sequel the usage of QoS parameters is demonstrated for expressing video unit transport. However, if ViNo is used in a different context, the semantics of QoS annotations may differ and need to be clarified prior to any calculations based on ViNo expressions. We provide examples for the well-known transport-related QoS parameters bandwidth, delay, and jitter.

We derive the notation and semantics of QoS parameters from the QoS language QL [3]. QL is based on events like reception and sending of messages. It uses a function τ mapping events to points in time. Since ViNo is based on compositions, we restrict ourselves to the event of *receiving a composition* c at a given network node or video display. This event occurs as soon as all video units referenced by c have been received completely. We denote the corresponding point in time as $\tau(c)$. In this paper we focus on QoS parameters that can be used for delay calculations of video transport processes described by ViNo expressions.

Definition 4: We define a recursive function $delay$ to calculate a delay bound for a QoS-annotated ViNo transport description c:

1) The null unit causes no delay: $delay(u_0) = 0$.
2) If $c = c_1 \leftarrow_Q u$ for some composition c_1 and a video unit u, then

$$delay(c) = delay(c_1) + delay(u, Q)$$

where $delay(u, Q)$ is defined to be the delay $\tau(u) - \tau(c_1)$ assuming a provided QoS parameter Q (trivial case of recursion).

3) If $c = c_1 \leftarrow c_2$ for compositions c_1 and c_2, then the delay bound is computed recursively as:

$$delay(c) = delay(c_1) + delay(c_2)$$

For delay calculations, we assume that the transmission of c_2 occurs as soon as possible after the transmission of c_1, which is expressed by omitting the QoS parameter.

4) If $c = c_1 \,||\, c_2$ for compositions c_1 and c_2, then the delay bound is computed recursively as:

$$delay(c) = max(delay(c_1), delay(c_2))$$

Note that the $delay$ function is defined only on a subset of all possible ViNo expressions. The following two expression types will occur frequently in the subsequent examples, so we introduce a separate notation for the corresponding delay bounds:

- If $c = u_0 \leftarrow_{Q_1} u_1 \leftarrow_{Q_2} \cdots \leftarrow_{Q_n} u_n$ for video units u_i (u_0 is the null unit), then

$$\begin{aligned} delay(c) &= \sum_{i=0}^{n} delay(u_i, Q_i) \\ &= delay(u_1, \ldots, u_n, Q_1, \ldots, Q_n, seq) \end{aligned} \quad (6)$$

  where the last term introduces a new notation.
- If $c = (u_0 \leftarrow_{Q_1} u_1) \,||\, \ldots \,||\, (u_n \leftarrow_{Q_n} u_n$ for video units u_i (u_0 is the null unit), then

$$\begin{aligned} delay(c) &= \max_{1 \le i \le n} (delay(u_i, Q_i)) \\ &= delay(u_1, \ldots, u_n, Q_1, \ldots, Q_n, par) \end{aligned} \quad (7)$$

  where the last term introduces a new notation.

Whether the $delay$ function represents a lower or upper delay bound depends on the definition of the $delay(u, Q)$ values. We now demonstrate how to define these values if the QoS parameters are given in terms of bandwidth, delay, or jitter, respectively.

1) Bandwidth: By $Q = BW$ we express that a given bandwidth BW is available for transmission. The delay of transmitting a video unit u is defined as:

$$delay(u, BW) = \frac{size(u)}{BW}$$

The $delay$ function therefore computes a *lower bound* of the end-to-end delay corresponding to a given ViNo expression. We assume that video units transmitted in parallel according to some parallel ViNo composition use separate links, so that the available bandwidth is not reduced by parallel transmissions.

Calculation of the lower delay bounds of sequential and parallel compositions of video units according to equations (6) and (7) results in:

$$\begin{aligned} &delay(u_1, \ldots, u_n, BW_1, \ldots, BW_n, seq) \\ &\qquad = \sum_{i=1}^{n} \frac{size(u_i)}{BW_i} \end{aligned} \quad (8)$$

$$\begin{aligned} &delay(u_1, \ldots, u_n, BW_1, \ldots, BW_n, par) \\ &\qquad = \max_{1 \le i \le n} \left(\frac{size(u_i)}{BW_i} \right) \end{aligned} \quad (9)$$

2) Delay: By $Q = D$ we express that transmission yields a given delay D. The delay of transmitting a video unit u is defined as:

$$delay(u, D) = D$$

If all delays occurring in a video delivery system are expressed as provided QoS parameters of a corresponding ViNo composition and if the composition is an appropriate model of the system, the calculated end-to-end delay value should be accurate. However, for practical purposes, the ViNo composition is constructed to provide an *upper delay bound* only, which may lead to a simpler ViNo expression.

3) Jitter: According to the QoS language QL, jitter can be defined by specifying *lower and upper delay bounds* (D_{min}, D_{max}). To calculate the jitter of a given video transport system described by an appropriate ViNo expression, we therefore just need to apply the $delay$ function to both bounds separately. We obtain two functions $delay_{min}$ and $delay_{max}$ with appropriate definitions of delay bounds for transmitting a video unit u:

$$\begin{aligned} delay_{min}(u, D_{min}) &= D_{min} \\ delay_{max}(u, D_{max}) &= D_{max} \end{aligned}$$

The jitter of a ViNo composition c is then computed as $(delay_{min}(c), delay_{max}(c))$.

To illustrate delay calculations, we now apply the $delay$ function to example 3 of section III-A. We restrict the discussion to delay as QoS parameter. Let the delay D_1 for one unit delivered from $P1$ to $P3$ be 300 ms, and the delay D_2 from $P2$ to $P4$ be 350 ms. The delay D_3 from both proxies $P3$ and $P4$ to the client shall be 200 ms each. We need to extend the ViNo transport description (see (5) and TABLE I) to introduce delay parameters:

$$\begin{aligned} c = &[(u_0 \leftarrow_{D_1} u_1) \,||\, (u_0 \leftarrow_{D_2} u_4)] \\ &\leftarrow [(u_0 \leftarrow_{D_1} u_2) \,||\, (u_0 \leftarrow_{D_2} u_5) \\ &\qquad ||\, (u_0 \leftarrow_{D_3} u_1) \,||\, (u_0 \leftarrow_{D_3} u_4)] \\ &\leftarrow [(u_0 \leftarrow_{D_1} u_3) \,||\, (u_0 \leftarrow_{D_2} u_6) \\ &\qquad ||\, (u_0 \leftarrow_{D_3} u_2) \,||\, (u_0 \leftarrow_{D_3} u_5)] \\ &\leftarrow [(u_0 \leftarrow_{D_3} u_3) \,||\, (u_0 \leftarrow_{D_3} u_6)] \end{aligned}$$

Note that this ViNo composition is of the form $c = c_1 \leftarrow c_2 \leftarrow c_3 \leftarrow c_4$, where each c_i denotes a parallel composition. By applying equation (7) and case 3 of Definition 4 we therefore obtain:

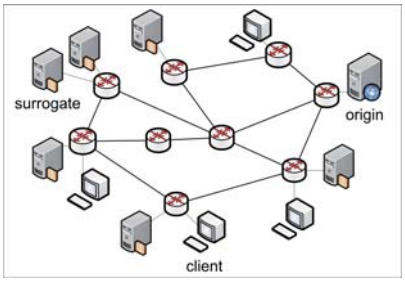


Fig. 4. Sample Architecture in CDNsim

$$
\begin{aligned}
delay(c_1) &= delay(u_1, u_4, D_1, D_2, par) \\
&= \max(D_1, D_2) = D_2 \\
delay(c_2) &= \max(D_1, D_2, D_3, D_3) = D_2 \\
delay(c_3) &= \max(D_1, D_2, D_3, D_3) = D_2 \\
delay(c_4) &= \max(D_3, D_3) = D_3 \\
delay(c) &= delay(c_1) + delay(c_2) + delay(c_3) + delay(c_4) \\
&= D_2 + D_2 + D_2 + D_3 \\
&= (350 + 350 + 350 + 200)\ \text{ms}\ = 1250\ \text{ms}
\end{aligned}
$$

IV. Analyzing Transport and Caching

In this section the potential of ViNo is shown as a tool for analyzing existing delivery systems, such as Content Delivery Networks (CDN). Additionally, we applied the same analysis to our non-sequential multimedia cache to compare its caching techniques to CDN.

A. Content Delivery Networks

CDNs consist of origin servers that are supported by strategically placed surrogate servers to which the content is replicated and/or cached (see [19], [20]). In most of the commercially available CDNs, the content is passively pulled by surrogate servers. Usually, commercial CDNs are not available for research purposes. Even academic CDNs, which are available on PlanetLab, are treated as black boxes. For this reason, Stamos et al. [4] developed a simulation environment, called CDNsim, for large scale CDN simulations. This simulation is the basis for our experiments with ViNo.

A GUI for configuring simulations is also part of CDNSim. CDNSim is an Omnet++ [21] simulation and uses the INET Framework Library [22]. It covers all typical CDN functionality, such as DNS request redirection and LRU replacement. CDNSim supports different cooperation policies such as closest surrogate or random surrogate cooperation, but also simple non-cooperative behavior can be configured. A very interesting point for our investigations is the fact that if the number of nodes and routers remains the same the same architecture is generated for each simulation run. Therefore, the clients connect always to the same surrogate servers.

TABLE II shows the configuration parameters used for our experiments. Sample request traces and router topologies are made available by the CDNSim developers at [23]. For sake of simplicity we decided to use the non-cooperative policy for our experiments, i.e, if a requested object is not available at the client's surrogate server the request is forwarded to the origin server. However, all delay calculations can also be applied to the cooperative policies as well.

The optimal unit size for a given application is an open research issue. For our simulation experiments we simply selected some reasonable size, namely 1500 bytes. This means that a web page of 4,500 bytes is divided into 3 units and is described as $u_1 \leftarrow u_2 \leftarrow u_3$. The link speed is specified to be 200 Mbits/sec, which results in a bandwidth BW of 16,666 units/sec.

We evaluate ViNo by calculating delay in miss and hit situations and compare the results to the simulated values.

In general a hit is represented as the distance from a client to its surrogate, which is 1 hop. On a miss the transport represents a sequential composition of two stages, i.e., from origin to surrogate and from surrogate to client (e.g., $c = (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3) \leftarrow (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3)$). Thus, all calculations can be done without the complete knowledge of the CDN's architecture. The calculations represent the time a transport takes at minimum, i.e., it is the best case transport delay. For the experiments these calculations are referred to as *ViNo generic*.

The delay function for the example above can be described as $delay(c)$ and can be calculated as follows:

$$
\begin{aligned}
delay(c) &= delay(u_1, u_2, u_3, BW, ..., BW, seq) \\
&\quad + delay(u_1, u_2, u_3, BW, ..., BW, seq) \\
&= \frac{6}{16,666} = 0.36\ ms
\end{aligned}
$$

If the architecture is known in more detail, which is the case for CDNSim, more precise calculations can be performed. As shown in Fig. 4 routers are placed on the path of clients and surrogates. The idea was to consider those routers as hops, e.g., a client is 3 hops away from its surrogate server. On a hit the delay can be calculated based on the distance (measured in hops) from client to surrogate. For the experiments these calculations are referred to as *ViNo routers*.

Two experiments were started, (1) all clients download web pages of small size; (2) one client downloads a number of different sized web pages.

Experiment 1. This experiment proves the general applicability of ViNo, its results are shown in Fig. 5. It is seen that both ViNo routers and ViNo generic estimate well the delay pattern of the measured values. The distance of ViNo routers to the simulated delay is smaller because its calculations are more precise, for the price that the transport paths have to be known in advance.

We show by the example of downloading one single object, how the corresponding delay is calculated. A randomly chosen client with the id $c1009$ connects to the surrogate server with id $s1199$ in 4 hops. A miss means a transport over 9 hops from the origin server. This client downloads the

Parameter	Value
Router Topology	Waxman for 1000 Routers
Link speed	200 Mbit/sec
Number of clients	100
Number of surrogate servers	100
Number of origin servers	1
Number of outgoing connections	1000
Websites	50000 Web objects, 100MB max object size, sizes' zipf=1, size vs popularity correlation = 0
Traffic	1000000 requests, popularity's zipf = 1.0, expo mean interarrival time = 1, 100 client groups

TABLE II
CDNSIM CONFIGURATION PARAMETERS

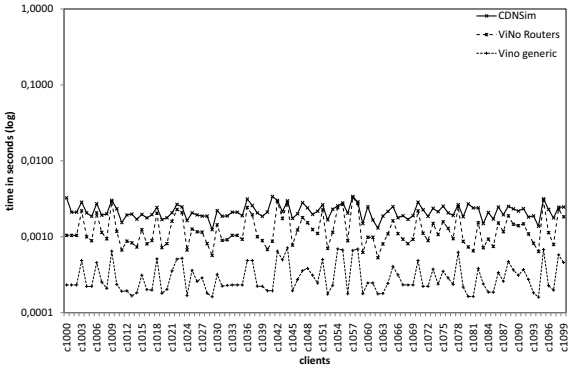


Fig. 5. Comparison of download time ViNo vs. CDNSim

object with id 13, which has a size of 5 units. In ViNo one stage consisting of these 5 units can be described as $c_i = u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3 \leftarrow_{BW} u_4 \leftarrow_{BW} u_5$. In the simulation object nr 13 was not present at the surrogate server. Thus, the overall composition c for ViNo routers is a sequential composition of stages 1-9 (one stage per hop).

$$\mathrm{h} = 9, \mathrm{BW} = 16,666u/sec$$

$$delay(c) = \sum_{i=1}^{h} delay(c_i)$$

$$= h * delay(u_1, ..., u_5, BW, ..., BW, seq) = 2.7\ ms$$

The ViNo generic calculation has no detailed knowledge of the routers and reduces therefore the miss to two hops, such that the calculation changes to:

$$\mathrm{h} = 2, \mathrm{BW} = 16,666u/sec$$

$$delay(c) = \sum_{i=1}^{h} delay(c_i)$$

$$= h * delay(u_1, ..., u_5, BW, ..., BW, seq) = 0.6\ ms$$

The measured value was 3.01 ms, which shows that ViNo routers calculation is a really good estimation.

Experiment 2. This experiment was done to investigate the impact of different file sizes, since video objects are in general larger than web objects. One client was picked out of all clients, which downloads a set of very different sized objects. The generic and the router based delay was calculated and then compared to the simulated results. The ViNo router calculated delay is shown in Fig. 7 and it can be seen that the calculations do not always represent the lower bound of the simulated duration. One extreme case is shown for the object with id 637 (the peak in Fig. 7), which has a size of 24 MBytes (16,666 units). At this point of the simulation the object was not present at the surrogate, thus it had to be downloaded from the origin with a distance of 8 hops. The measured value was 3 seconds. The calculations with ViNo routers are provided below:

$$delay(c) = \sum_{i=1}^{8} delay(c_i)$$

$$= 8 * delay(u_1, ..., u_{16666}, BW, ..., BW, seq) = 8sec$$

This effect appears for files that exceed the size of 10 KBytes, which are routed in a different way than smaller files (as in experiment 1). Larger files are split up and are routed in parallel over several paths. Therefore, the transport is a mixture of parallel and sequential compositions and not purely sequential as assumed before. Since the routing algorithm is part of the INET Framework and the paths are not predictable with reasonable effort, we cannot provide a more detailed calculation. However, the router based calculations might represent the worst case delay if the routing path is always the same.

The generic calculations are always representing the lower bound of the duration as shown in Fig. 6, since in any way the surrogate downloads the complete website before forwarding it to the client. In comparison to the ViNo routers result the ViNo generic result is 2 seconds for the object with id 637.

Thus, the generic case represents the larger file downloads better and the router based calculations represent smaller file downloads better. Which type of calculation is finally taken depends on the knowledge of the architecture and on the purpose of the analysis.

The efficiency of CDNs and caches is usually compared by measuring the hit rate. ViNo can also be used to analyze the impact of the hit rate to the delay.

In our experiments the objects' sizes are Zipf distributed with an alpha value of 1.0 (strongly skewed). This means that

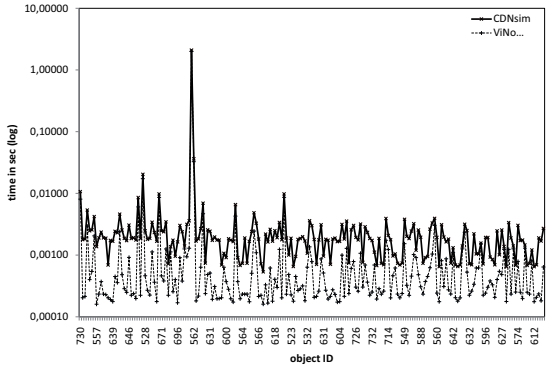


Fig. 6. Comparison of download time ViNo generic vs. CDNSim

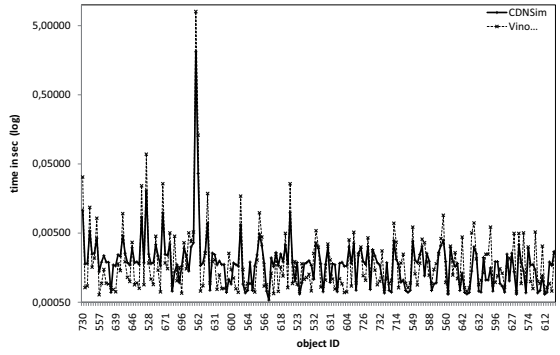


Fig. 7. Comparison of download time ViNo routers vs. CDNSim

80 % of the overall objects' size are represented by 20 % of the objects. This fact has a huge impact on the surrogates' cache size. In CDNSim one cache was able to store 109 MBs, which lead to hit rates of around 80 %. The reason is that most surrogates handle small files and the cache misses only occured in the beginning of the simulation until all objects were loaded from the origin server (i.e., the actual cache size was around 100 %).

If the surrogate servers had used popularity based prefetching the hit rate would have reached 100 %. This leads to the question on how much delay improvement prefetching would make. For this investigation we took ten random surrogate servers out of the simulation and calculated the delay reduction as shown in TABLE III. It can be seen that the number of units (i.e., the size of the original files) to serve vary a lot. E.g., surrogate s1191 only serves 19 files, whereas surrogates s1158 and s1164 serve almost the same amount of units, but the number of objects differ by a factor of 10. In general, those surrogate servers serving large files have advantages if prefetching is used. A CDN provider for

SID	no. clients	no. objects	no. units	prefetch	pipeline
s1110	2	2158	26290	1.57733	1.57727
s1132	1	475	4033	0.24198	0.24190
s1140	3	46	210	0.01260	0.01255
s1158	1	214	18352	1.10110	1.10109
s1164	3	2632	18597	1.11571	1.11571
s1188	4	3007	49100	2.94594	2.94591
s1191	2	19	69	0.00414	0.00408
s1194	2	6658	77183	4.63089	4.63077
s1196	4	3355	56281	3.37679	3.37672
s1198	6	3291	39795	2.38764	2.38758

TABLE III
SURROGATE DELAY REDUCTION IN SECONDS ON PREFETCH AND ON PIPELINING

videos can reach better surrogate efficiency and therefore startup delay minimization if popularity based prefetching per surrogate is applied. However, the popularity measures must include different factors, e.g, region, as we might assume that clients in Europe have different interests than in America, aso.

Another solution for the CDN provider could be to apply pipelined transport on a miss, i.e., a surrogate forwards a unit immediately after download from the origin. For a web site that consists of three units this is described as:

$$c = (u_0 \leftarrow_{BW} u_1) \leftarrow [(u_0 \leftarrow_{BW} u_2)||(u_0 \leftarrow_{BW} u_1)] \leftarrow [(u_0 \leftarrow_{BW} u_3)||(u_0 \leftarrow_{BW} u_2)] \leftarrow (u_0 \leftarrow_{BW} u_3)$$

In comparison to $c_1 \leftarrow c_2 = (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3) \leftarrow (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3)$ for the pure sequential transport. The delay for the pipelined composition is calculated as the sum of all sub-compositions (i.e., $c = c_1 \leftarrow c_2 \leftarrow c_3 \leftarrow c_4$).

$$\begin{aligned} & delay(c) \\ &= \sum_{i=1}^{4} delay(c_i) \\ &= delay(u_1, BW) \\ &+ \max(delay(u_2, BW), delay(u_1, BW)) \\ &+ \max(delay(u_3, BW), delay(u_2, BW)) \\ &+ delay(u_3, BW) \\ &= \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} = 0.24ms \end{aligned}$$

If the units would be transported sequentially as in $c_1 \leftarrow c_2$ the delay would be calculated as the sum of the sub delays, i.e:

$$\begin{aligned} & delay(c) \\ &= delay(u_1, u_2, u_3, BW, ..., BW, seq) \\ &+ delay(u_1, u_2, u_3, BW, ..., BW, seq) \\ &= \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} \\ &= 0.36ms \end{aligned}$$

The transport would need 3+3=6 time slots. The pipelined transport reduces the number of time slots to 4, in more general

(for a 2-stage pipeline):

$$delay_{pipelined} = \frac{delay_{sequential}}{2} + 1\ time\ slots$$

If the surrogates analyzed before used pipelined transport on a miss the delay would reduce in comparison to a sequential transport as shown in TABLE III. Which of the both techniques a CDN provider chooses is a matter of implementation.

B. Non-sequential Multimedia Caching

If resources at a surrogate server are more limited and the access patterns more flexible than in the CDNSim case before, a CDN provider might be interested in a more efficient replacement and prefetching policy. This analysis was done in [1] and extended results are shown in the following.

We assume that the units are self-contained and equipped with metadata comprising further information about the content. Furthermore, we assume that a smart user application exists that provides information about user intentions (see [24]). User intentions are metadata about semantic roles a user can be categorized to. Such a role could be, e.g., "informational" denoting users looking for many but unspecific data and "transactional" denoting users wishing to buy a specific content. A semantic group of units is therefore a number of units that is of interest for a category of users. Note that the unit groups are not disjoint, but a user mapped to a given role is expected to request those units that are mapped to that role. This favors units that are more popular than others. Units in each group are ordered according to their popularity. This knowledge is exploited in the proposed cache admission policy.

The initial content of the cache is prefetched at system start and regards the most popular units of all groups, depending on the cache's size. Subsequently, the next fitting unit from a user group will be prefetched. The next fitting unit is the unit following the currently requested unit regarding popularity within the current group. This policy is called "simple cache admission policy" and formally defined as follows:

$$\text{prefetch}= \begin{cases} u_{next} & \text{if hit } u_{current} \\ 0 & \text{if hit } u_{current} \text{ \&\& hit } u_{next} \\ u_{current} \leftarrow u_{next} & \text{else} \end{cases}$$

However, this policy is inefficient, since unpopular units are prefetched as well. Therefore, the second admission policy is based on a rank calculation (r_{all}) over all groups for each unit. If the calculated global rank is below a predefined rank (rank 0 is the highest rank level), the unit is considered for prefetching. The impact of low popularity is minimized using the logarithm of the group rank.

$$r_{all} = \frac{1}{n} \sum_{i=1}^{n} \ln r_i$$

If a unit is in the top 5 of one group and less popular in another group, it is more likely that this unit is cached than a unit that has an average popularity within several groups.

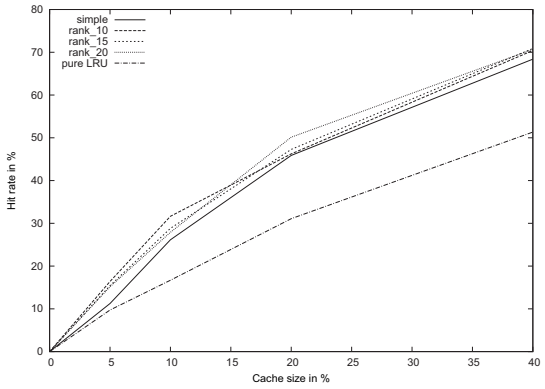


Fig. 8. Hit rate comparison of pure, simple and rank-based admission using LRU.

This approach was evaluated with a discrete-event based simulation using Omnet++. The user requests were generated with Medisyn [25], in a similar way as done in CDNSim.

For 100 units with different popularity, 10000 requests per user group were generated. Initially, LRU was implemented as the replacement strategy. The threshold was mapped to ranks 10, 15 and 20. Furthermore, we used CDNSim with the same parameters to compare pure LRU without prefetching to the non-sequential cache.

First, the hit rate comparison is done for the LRU-based admission policies, the results are depicted in Fig. 8. The rank-based admission policy shows an improvement of 5-10 % according to the simple admission and up to 20 % of improvement in comparison to the non-prefetching policy. It can also be seen that the thresholds of the rank-based policy show small differences in hit rate, but the number of prefetches increases the higher the threshold is specified. For further experiments the threshold of rank 10 is seen to be sufficient.

Although LRU supports popular units to remain longer in the cache, for small cache sizes even popular units are often replaced in the case of prefetching. Fig. 9 shows the factor of requests sent to the server in comparison to the client requests. This shows the maybe surprising result for small cache sizes it would be more efficient to send the units directly from the server, because the number of units sent from the server exceeds the number of requests. For the simple admission policy the server ought to send units in vain until 30 % of cache size. Whereas the rank-based policy decreases the load to an efficient level already at a cache size of 10-15 %. For less unnecessary replacements LRU has to be substituted by a replacement policy that considers the unit popularity.

A unit has to be prefetched and cached if it is popular enough. The rank calculation of the admission policy can also be applied to the replacement policy.

The effect on the load is shown in Fig.11. The simple admission policy starts to be efficient from a cache size of 10 % in comparison to LRU replacement. Also the rank-based

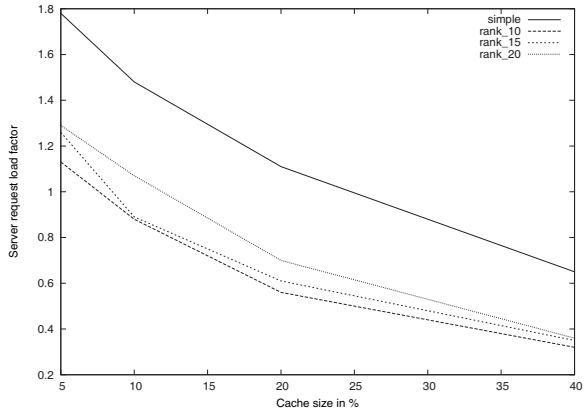


Fig. 9. Factor of server requests compared to user requests (LRU)

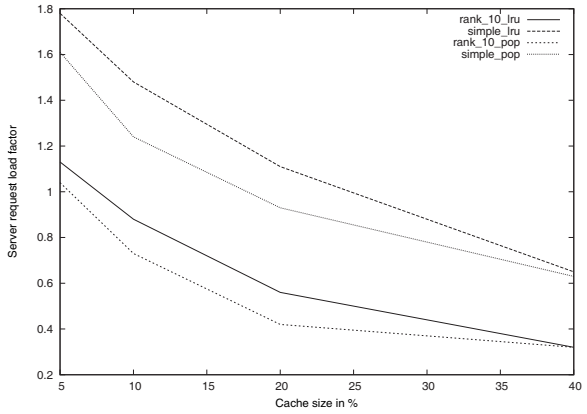


Fig. 11. Factor of server requests compared to user requests (LRU, popularity Relacement)

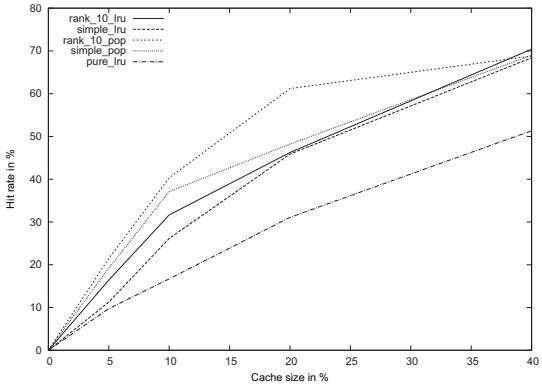


Fig. 10. Hit rate comparison of the admission policies using LRU and popularity replacement

admission policy starts to be efficient from very small cache sizes. However, this policy is complex and might not be used for caches with limited computing resources. For this cases the simple admission policy in combination with rank-based replacement is preferable.

Fig. 10 compares the hit rate evolution for both replacement policies. The comparison to pure LRU shows that the hit rate increases remarkably if applying a combination of rank-based admission and rank-based replacement. For a cache size of 20 % the hit rate differs by 40 %. Video CDN providers would gain remarkable storage cost reductions if applying rank-based caching.

V. Conclusion and Future Work

In this paper we extended the definition of the syntax and semantics of the Video Notation (ViNo) published in [26] and in [1]. Its applicability for describing and analyzing video transport is shown by simple examples and by the evaluation of an existing transport technique (CDN). For this reason we investigated two calculation types: (1) a simple, generic calculation that is always valid for the estimation of the best case delay and (2) a router based calculation that is more precise if the architecture is known in advance.

We showed in a caching scenario by taking advantage of ViNo how the use of prefetching and pipelining reduces startup delay. It is shown that ViNo could support content providers' system design decisions without extensive simulations.

The analysis of the CDN simulation was further used to investigate flexible access patterns in a non-sequential caching technique. The next unit to prefetch is depending on the popularity of this unit in all defined user groups. By comparing these prefetching techniques LRU replacement was found to replace units too frequent. Thus, LRU should be substituted by a popularity based replacement. The popularity based replacement technology improves hit rate and reduces the load of the server remarkably. This proactive caching and prefetching policy can be an efficient technique for multimedia CDNs, because storage and costs would be reduced and quality be increased (smaller start-up delays). The calculation effort of rank-based prefetching combined with rank-based replacement might have huge impact on the performance of a system. Thus, the decision of which cache admission and replacement policy to use depends on the resources available in the system to analyze.

However, ViNo cannot fully substitute a simulation, since the prediction of specific steps in a system (e.g., dynamic routing paths) cannot be made with reasonable effort. ViNo can be used as a tool for approximating general behavior of multimedia transport, e.g., to compute the best case delay on a miss or on a hit. One of ViNo's strength is the expression of different transport techniques, which allows a simple comparison on the first sight.

By using ViNo in research articles authors can explain new

transport techniques. For example, a new flexible approach can be formally described in a few lines. This would banish a lot of ambiguity from the scientific discussion.
Future work will regard further QoS calculations beyond delay, and the definition of unit loss. Another issue is dynamic unit size, which will be needed for semantically meaningful units. In this context ViNo will be applied to compare self-organizing multimedia transport to existing techniques. It is expected that in non-sequential cases the flexible system will outperform the traditional systems regarding startup delay and user experience, even though the proposed caching technique will also have its costs.

VI. Acknowledgements

The results presented in this paper are part of the research efforts for the SOMA (Self-organizing Multimedia Architecture) project, a Klagenfurt University and Lakeside Labs GmbH cooperation (URL: http://soma.lakeside-labs.com/)

Appendix

The following ViNo specification was created by using ANTLR a LL(*) parser generator [27]. Since special signs are not allowed in ANTLR, we used <_Q for $\leftarrow_Q$.

```
SEQ : '<' Q? ;
PAR : '||';
NUMBER : ('0'..'9');
VALUE : NUMBER+;
LETTER : ('a'..'z'|'A'..'Z');
NAME : LETTER (NUMBER | LETTER)*;
Q : ('_' NAME ('=' | '>=' | '<=') VALUE)+;

unit : NAME;

primitive: unit | group;

par: (PAR primitive)+;

seq: (SEQ primitive)+;

pargroup: '[' primitive par ']';

seqgroup: '(' primitive seq ')';

group: pargroup | seqgroup;

comp: primitive ( par | seq )?;
```

References

[1] A. Sobe and L. Böszörmenyi, "Non-sequential multimedia caching," in *International Conference on Advances in Multimedia MMedia2009*. IEEE Computer Society, 2009, pp. 158–161.

[2] M. Lux, O. Marques, K. Schöffmann, L. Böszörmenyi, and G. Lajtai, "A novel tool for summarization of arthroscopic videos," *Multimedia Tools and Applications*, vol. 46, no. 2, pp. 521–544, January 2010.

[3] G. S. Blair and J.-B. Stefani, *Open Distributed Processing and Multimedia*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[4] K. Stamos, G. Pallis, A. Vakali, D. Katsaros, A. Sidiropoulos, and Y. Manolopoulos, "Cdnsim: A simulation tool for content distribution networks," *ACM Transactions on Modeling and Computer Simulation*, 2009.

[5] J. Jin and K. Nahrstedt, "Qos specification languages for distributed multimedia applications: a survey and taxonomy," *Multimedia, IEEE*, vol. 11, no. 3, pp. 74–87, July-Sept. 2004.

[6] J. Altmann and P. Varaiya, "Index project: user support for buying qos with regard to user's preferences," in *Quality of Service, 1998. (IWQoS 98) 1998 Sixth International Workshop on*, May 1998, pp. 101–104.

[7] X. Gu, K. Nahrstedt, W. YUAN, D. Wichadakul, and D. Xu, ""an xml-based quality of service enabling language for the web"," *Journal of Visual Language and Computing, special issue on multimedia languages for the Web*, vol. 3, pp. 61–95, 2002.

[8] S. Frolund and J. Koistinen, "Qml: A language for quality of service specification, hpl-98-10," HP Laboratories, Tech. Rep., 1998.

[9] I. Foster and C. Kesselman, "The globus project: a status report," in *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, Mar 1998, pp. 4–18.

[10] O. Lampl, E. Stellnberger, and L. Boeszoermenyi, "Programming language concepts for multimedia application development," in *Modular Programming Languages*. Springer, September 2006, pp. 23–36.

[11] O. Lampl and L. Böszörmenyi, "Adaptive quality-aware programming with declarative qos constraints," in *Internet and Multimedia Systems and Applications, EuroIMSA 2008*, M. Roccetti, Ed., 2008.

[12] D. C. Bulterman and L. W. Rutledge, *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer Publishing Company, Incorporated, 2008.

[13] Y. Zhao, D. L. Eager, and M. K. Vernon, "Scalable on-demand streaming of nonlinear media," *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1149–1162, 2007.

[14] S. Podlipnig and L. Böszörményi, "A survey of web cache replacement strategies," *ACM Computing Surveys*, vol. 35, pp. 331–373, 2003.

[15] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 1999, pp. 1310–1319 vol.3.

[16] K. L. Wu, P. S. Yu, and J. L. Wolf, "Segment-based proxy caching of multimedia streams," in *WWW '01: Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA: ACM, 2001, pp. 36–44.

[17] J. Yu, C. Chou, Z. Yang, X. Du, and T. Wang, "A dynamic caching algorithm based on internal popularity distribution of streaming media," *Multimedia Systems*, pp. 135–149, October 2006.

[18] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "Disc: Dynamic interleaved segment caching for interactive streaming," *Distributed Computing Systems, International Conference on*, vol. 0, pp. 763–772, 2005.

[19] G. Pallis and A. Vakali, "Insight and perspectives for Content Delivery Networks," *Commununications of the ACM*, vol. vol. 49, no. 1, 2006.

[20] A. Vakali and G. Pallis, "Content Delivery Networks: Status and Trends," *IEEE Internet Computing*, vol. 7, no. 6, 2003.

[21] "Omnet++ discrete event simulator." [Online]. Available: http://www.omnetpp.org

[22] "Inet framework." [Online]. Available: http://inet.omnetpp.org

[23] "Cdnsim." [Online]. Available: http://oswinds.csd.auth.gr/~cdnsim/; accessed 12/09

[24] C. Kofler and M. Lux, "Dynamic presentation adaptation based on user intent classification," in *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*. New York, NY, USA: ACM, 2009, pp. 1117–1118.

[25] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat, "Medisyn: a synthetic streaming media service workload generator," in *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2003, pp. 12–21.

[26] A. Sobe and L. Böszörmenyi, "Towards self-organizing multimedia delivery," Reports of the Institute of Information Technology, Klagenfurt University, TR/ITEC/12/2.08, Tech. Rep., 2008.

[27] T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.

Ontology-based Indexing and Contextualization of Multimedia Documents for Personal Information Management Applications

Annett Mitschick
Dresden University of Technology, Department of Computer Science
Chair of Multimedia Technology
Dresden, Germany
annett.mitschick@tu-dresden.de

***Abstract*—With the help of Semantic Web technologies, which ensure machine processability and interchangeability, we are able to apply semantic knowledge models to organize and describe heterogeneous multimedia items and their context. However, an ontology-based document management system has to meet a number of challenges regarding flexibility, soundness, and controllability of the semantic data model. This paper presents an integrated approach for ontology-based multimedia document management which covers the process of automated modeling of semantic descriptions for multimedia objects and their long-term maintenance, allowing for the domain-specific customization of the used ontology. Furthermore, the proposed approach addresses the problems of data validation and consolidation to ensure semantic descriptions of proper quality. We demonstrate the practicability of our concept by a prototypical implementation of a service platform for personal information management applications.**

***Keywords*-personal multimedia document management; semantic metadata; generation; maintenance;**

I. Introduction

As digital devices have found their way into nearly all domains of every-day life, the amount of digital multimedia content is increasing and becomes more valuable and important. Managing a considerable quantity of multimedia documents involves administration efforts and certain strategies for ordering and arrangement to keep track of content and structure of the collection – esp. over a long period [1]. The problem is intensified by the complex and partly high-dimensional characteristics of multimedia objects. Problems which appear when users deal with search and retrieval tasks within personal document collections mainly result from lacking expressiveness and flexibility of the structure of traditional file systems. Another problem users are facing today is an increasing *information fragmentation* [2]. A large number of desktop applications for personal document management exists, typically applying individual storage and indexing structures for specific document types (e. g., photo management software) and providing different access to the content. The reuse of metadata across personal desktop applications is rather restricted.

With the help of Semantic Web technologies, which ensure machine processability and interchangeability, we are able to apply semantic knowledge models and paths to organize and describe heterogeneous multimedia items and their context. A document collection is no longer an aggregation of separate items, but forms an individual knowledge base providing rich and valuable data for innovative PIM (personal information management) applications which present an aggregated view of the relations and links between personal documents, dates, contacts, e-mails, etc. To avoid the information fragmentation mentioned above, such PIM applications should be lightweight solutions, accessing a central ontology-based document management system. Such an ontology-based personal multimedia document management system has to meet several challenges:

- Diverse file and metadata formats are in use today and even more will evolve in the near future. Thus, it is of utmost importance that suitable document analysis, metadata and feature extraction modules can be added to the system without any difficulty. Extensibility of supported schemas or standards also means that knowledge modeling and processing modules must be flexible and configurable enough to allow for media or format specific knowledge instantiation.
- The ontology model used as a foundation of the instantiated document and context descriptions must be expressive and efficient. In the context of personal document and information management, the design of a suitable ontology model is non-trivial, only few proposals exist, and standards are still missing. Thus, the design of modules for knowledge processing, storing and provision should take into account that ontology models need to be replaced or changed. This must not result in substantial re-engineering work.
- As semantic data about documents and their relations to other resources tends to become very complex over time and therefore difficult to handle, it is necessary to integrate and apply control facilities, to enable the user to take corrective action and prevent him from being overstrained. Another result of the growing complexity of a knowledge base can be a loss of confidence – if the users are no longer able to check its correctness

themselves. Thus, appropriate support must be provided to make sure that the content of the knowledge base is sound and reliable at any time.

In this paper we present an integrated approach for ontology-based personal multimedia document management which addresses these issues, developed within the K-IMM (Knowledge through Intelligent Media Management) project [3]. After a discussion of related work in Section II, we present the process of generating semantic descriptions for personal multimedia documents in Section III and our approach for data consolidation and document life cycle management in Section IV. The architecture of the K-IMM system, including a prototypical example application for personal document and information management, is presented in Section V. Section VI concludes the paper and suggests future research directions and open issues.

II. Related Work

Existing projects with comparable goals (ontology-based document management) can be classified according to their focus on either manual ontology-based annotation or (semi-) automatic semantic data modeling. A comprehensive survey of the state of the art of semantic annotation for knowledge management is presented in [4]. Manual annotation systems mostly emerged in the context of Web document annotation, e. g., Annotea [5], SMORE [6] and CREAM [7]. Later, dedicated multimedia document annotation solutions like, e. g., Caliph&Emir [8] and AKTiveMedia [9], evolved. Most of the work on ontology-based annotation proceeds from the assumption that, before annotation starts, an appropriate ontology has to be created or assigned as a description schema (top-down approach). If this is left to the users, modality and sense of annotations depend on their intention which is even more difficult for non-ontology engineers.

Projects concerned with the problem of *automatic generation and maintenance* of semantic metadata are either targeting the automatic extraction and modeling of knowledge from documents (*Ontology Population* or *Ontology Instantiation*) based on NLP-techniques (e. g., KIM [10], ArtEquAKT [11], MediaCampaign [12]) or at the development of a so-called "Semantic Desktop". The NEPOMUK project [13] dealt with the development of a standardized, conceptual framework for "Semantic Desktops" which includes information extraction and wrapping from heterogeneous data sources, based on the Open Source project Aperture [14] (a reference implementation is Gnowsis [15]). Other projects that can be named in relation to "Semantic Desktop" are e. g., D-BIN [16], IRIS [17] and Haystack [18]. The latter turned out to be too non-restrictive to prevent data from being corrupted by the user.

At present, ontology-based solutions for multimedia document management are results of projects like aceMedia, BOEMIE or X-Media. They are either focused on automated annotation of image and video content [19], multimedia information extraction for ontology evolution [20] or large scale methodologies and techniques for knowledge management [21]. Adequate support for private users often means that a well-balanced compromise between manual and automatic annotation must be found. Presently, there is no integrated approach for ontology-based personal multimedia document management – from the content analysis to valid semantic metadata – accounting for existing context information and so-called "world knowledge". In particular, most of the existing approaches do not explicitly focus on controllability and long-term maintenance regarding data integrity and consistency, as well as *document life cycle management*. Furthermore, from a developer's point of view, the domain-specific customization and configuration (i. e., substitution of the used ontology model) is not explicitly supported.

III. Automatic Generation of Semantic Descriptions for Multimedia Documents

The prevalent uncertainty and ambiguity of interpretation and interrelation of information sources and the various application scenarios led us to the concept of a stepwise information instantiation process [22]. Figure 1 broadly depicts the generation process, showing a sequence of distinguishable stages of data modeling which will be described in more detail in the following.

A. Document Analysis

A multimedia document is processed and analyzed by a specific *Analyzer* component, depending on its media type and file format. Available Analyzers register dynamically at runtime and are thereupon considered as providers of specific information about a certain document type. They perform the task of document pre-processing, i. e., the file format specific processing and extraction of embedded metadata and raw data (content), and the format-independent analysis of the extracted (multimedia) content. Irrespective of the type of document, each analyzing process starts with the following steps: (a) identification of the file format, (b) extraction of embedded metadata, and (c) extraction of the raw data.

The correct identification of the file format is most important for the further processing and interpretation of the content. Even if the file has an appropriate filename extension, it can not be assured that it really complies with the corresponding format (e. g., because of multiple use of filename extensions). The primary decision criterion (whether the file can be processed and analyzed by the component) must be provided by the component itself to guarantee that the content can be analyzed correctly. Of course, embedded metadata can only be extracted and analyzed if the way it is stored (or embedded) complies with a certain standard or de facto standard. The same applies to the actual raw data of the document.

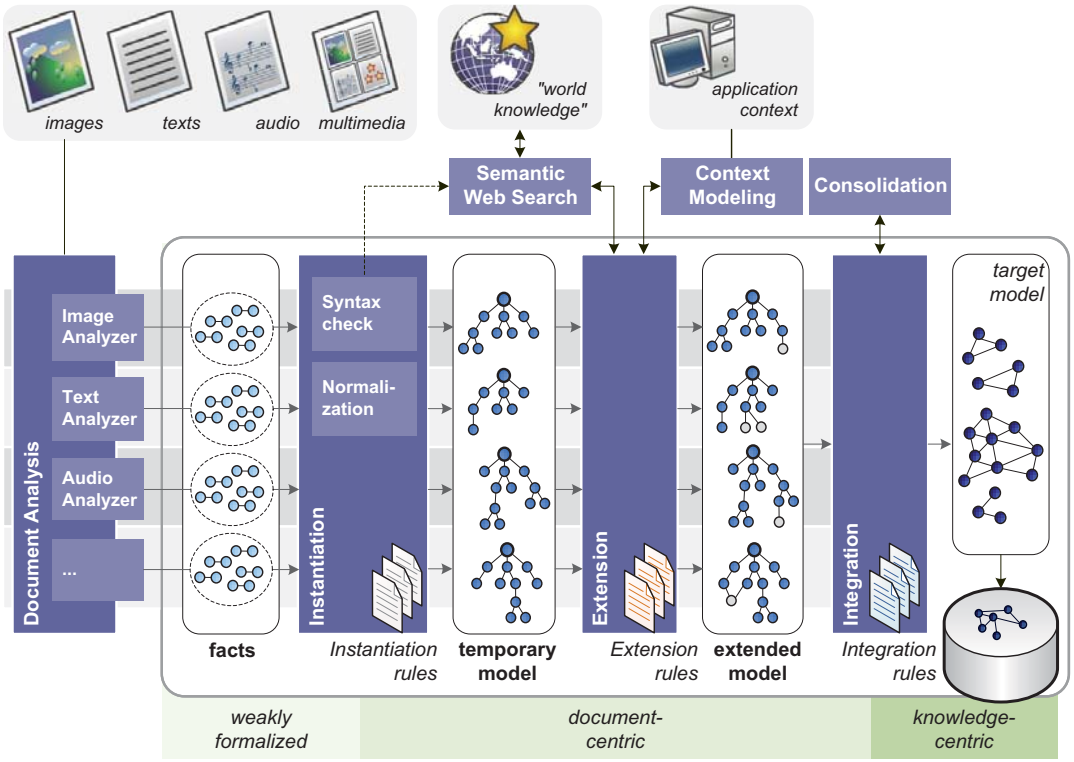


Figure 1. Data Modeling Process

The content-based exploitation and analysis of the raw data mainly comprises two subtasks: the segmentation or decomposition of the content to obtain logical parts (which can be further processed), and the determination of low- and high-level features. Following the principle of "divide-and-conquer", certain tasks of the content-based analysis of multimedia content are delegated to specialized subcomponents, allowing for reuse and substitution of particular solutions. Thus, an image which is part of a text document is extracted (by decomposition within a *Text Analyzer*) and passed to an appropriate *Image Analyzer*. Depending on the media type and file format, different techniques must be applied to extract low- and high-level features. Deducing high-level information from low-level features requires certain background information and user participation (e.g., to train classifiers for pattern recognition). For rather general fields of application, automated techniques are still missing and will hardly ever be on-hand without tight relationship to a user's context and conceptualization. Extent and complexity of the extracted data depends on how much background knowledge (rules and facts, or training data for the classification of low-level features) is available. The capabilities of an *Analyzer* component might be limited to the mere extraction of certain embedded metadata. Thus, it is possible to apply a combination of multiple *Analyzers* of different specialization to one document type.

B. Information Instantiation

As we can not predict the extent and quality of available information about multimedia documents, we need a flexible and extensible schema for the input data of the instantiation process. The most efficient way to specify descriptive information is in the form of attribute-value pairs (name and value of features or properties, like creator, modification date, but also color layout, sound intensity, etc.). A reduction to minimum structure allows a compact and uniform presentation of different sources and schemes. Furthermore, the list of attribute-value-pairs can grow dynamically. Thus, using this simple schema, an arbitrary number of Analyzer components can act as data providers. However, as the schema itself offers no validation ability, the passed input data might be incomplete or faulty, or contains redundancies or inconsistencies. To achieve an adequate level of data quality, the input data is evaluated within this process of *instantiation* (depicted in Figure 2) as follows:

1) **Filtering:** First of all, extracted data is filtered according to the requirements of the application domain. Filter criteria are defined in an editable configuration file. In case of redundant data a selection is made.
2) **Syntax check:** Syntactic errors occur if Analyzer components extract faulty data because of coding errors or problems with character sets. Examples are improper, supernumerary or missing characters, or the exceeding

of the range of values. The erroneous data is corrected or, if no automatic solution can be applied, excluded from further processing.

3) **Normalization:** Values originating from different data sources can be syntactically correct, but specified in different data formats (e. g., "`2008-01-13`" or "`01/13/2008`"). The transformation to a uniform, consistent data format is an important premise for further processing of the data.
4) **Transformation:** Finally, the filtered and normalized data is "translated" to the internal ontology model using a set of instantiation rules. In doing so, a decision is made regarding the interpretation of the mere syntactic input data by the semantic target model.

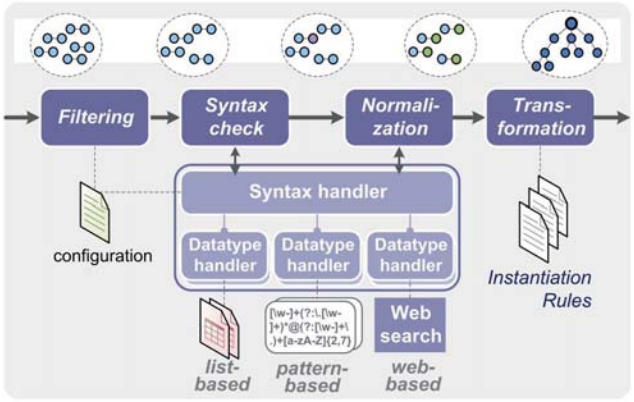


Figure 2. Information instantiation process

The process of syntax check and normalization is described in [22]. It is designed in a modular way, so that algorithms, sources and result format can be substituted and configured easily. The datatype handlers are either based on regular expressions (*pattern-based*), dictionaries (*list-based*) or web search results (*web-based*).

As for the process of *transformation*, we apply the following procedure: Each attribute-value-pair represents per se a *statement* and can be specified in RDF using the utility property `rdf:value` as predicate and a unique key as subject identifier. Hence, the resulting RDF model can be passed to a reasoner component to apply a set of configurable *Instantiation Rules*, allowing for appropriate customization.

C. Extension and Integration

In the next step, the resulting model is extended with additional data, i. e., with semantic information found on the Internet provided by a *Semantic Web Search Component*, described in more detail in [23], and current context information provided by a *Context Modeling Component*, presented in [24]. We assume that these services provide data in OWL over a standardized interface (using SPARQL) and that the ontologies are publicly available. An example for context information and a sample query is given in [24]. To allow for dynamic query composition, we introduced a template mechanism to specify SPARQL queries with the help of placeholders. An example (describing context information about an email transmission) is given in the following:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cm: <http://mmt.inf.tu-dresden.de/crocoon/
       context-mail.owl#>
PREFIX cu: <http://mmt.inf.tu-dresden.de/crocoon/
       context-upper.owl#>

SELECT ?mail ?property ?value
WHERE
{
  ?mail rdf:type cm:Email.
  ?mail cm:hasAttachment ?d.
  ?d cu:uniqueID [[SHA1_content]].
  ?mail ?property ?value.
}
```

The placeholders, tagged with squared brackets, are replaced at runtime with adequate attributes – in this case the SHA1 hash of the document's content, e. g.,

```
?d cu:uniqueID "d07149922d9f84c097f7ccf6ed5c7b658c4229d0".
```

The result of the query can be used to extend the existing semantic information about the document. Thus, a collection of configurable *Extension Rules* is applied to the temporary data. An example of an Extension Rule (in Jena Rules syntax [25], `person` and `core` are namespaces of the target ontology model) could be:

```
[foaf1: (?P rdf:type person:Person),
        (?F rdf:type foaf:Person),
        (?F foaf:page ?homepage),
        (?F foaf:depiction ?img)
        -> (?P person:homepage ?homepage),
           (?P core:imgLink ?img)]
```

Finally, the resulting temporary model can be inserted into the system's RDF repository. The concluding step of *Integration* (cf. Figure 1) performs two tasks:

- the exploitation of interrelations within the temporary model, and
- the verification and consolidation of the new information – both in isolation as well as in context of already existing information in the repository.

At first, a set of *Integration Rules* is applied to the temporary model to deduce interrelations between instances, e. g.,

```
[html1: (?H rdf:type ex:HTMLDocument),
        (?H ex:containsURL ?img),
        (?P rdf:type ex:Image),
        (?P ex:filepath ?fp),
        equal (?fp, ?img)
        -> (?H ex:contains ?P) ]
```

Secondly, the consolidation process is invoked, which is later on described in Section IV-A. Afterwards, the generation process is completed.

IV. MAINTENANCE

Due to the unsupervised analysis and extraction of document descriptions and context information, inserted data is likely to be of inferior quality in terms of consistency, accuracy, and redundancy. Furthermore, generated semantic descriptions become obsolete if documents are modified. In this section we describe our approach for maintaining the semantic data model, regarding consolidation and document life cycle management.

A. Data Validation and Consolidation

The semantic consolidation process within our system considers data in the context of the whole knowledge base. In general, consolidation is necessary whenever the knowledge base has been changed or extended by any automated process. It is invoked by the above-mentioned *Integration Component*. Our *Consolidation Component* is composed of three subcomponents: the *Semantic Conflicts Handler*, the *Duplication Handler*, and the *Incompletion Handler*. Their interrelation and the overall process of consolidation is described in detail in [22]. By now, the *Consolidation Component* provides high configuration ability: depending on the actual application context, the set of rules for the detection of semantic conflicts and incompletion, as well as the metrics and threshold for duplication detection can easily be adjusted or replaced. The following example rules (Jena Rules syntax) should illustrate the mode of action:

```
[rule1: (?P rdf:type ex:Person),
        (?P ex:bornOn ?T1),
        (?P ex:authorOf ?X),
        (?X ex:createdOn ?T2),
        greaterThan(?T1, ?T2)
        -> reportConflict(?P, ?X, '...description') ]

[label: (?O rdf:type ?C),
        noValue(?O rdfs:label)
        -> reportIncompletion(?O, 'Missing label...') ]

[fname: (?P rdf:type ex:Person),
        noValue(?P ex:familyName)
        -> reportIncompletion(?P,
               'Person has no surname...') ]
```

The first rule is an example for the detection of a semantic conflict (*"If a person, born on BirthEvent T1, is author of a Document X, created on AuthoringEvent T2, then T1 must have happened before T2.*). The second example is a domain-independent rule to detect incompletion (missing label), whereas the last one is a domain-dependent incompletion rule (missing attribute).

If a decision for conflict or duplication resolution can be made automatically, the user does not need to intervene. If a clear decision cannot be assured, the user must be involved for case-related judging. To minimize additional effort whilst providing the user with a high degree of control, it is necessary to find a compromise between fully automatic, semi-automatic and manual solution of the above-mentioned data problems. We propose two different approaches:

- Detected problems which can not be clearly solved are reported to the user (with a proposal for solution), leaving the active decision to him.
- All problems are solved automatically. Every decision is logged, in such a way that it can be undone.

Both approaches are supported by our solution as it provides machine-readable as well as human-readable problem description, according to a purpose-made ontology. These problem descriptions are produced by the above-mentioned handlers and passed to a central management component which performs the task of storage and provision, as well as solution and deletion of the conflicts depending on user feedback.

B. Document Life Cycle Management

A document's life cycle comprises all stages of a document: from its creation, processing, storage, and usage, to its disposal. In the context of personal document management, these stages are not clearly separable. Nevertheless, information about development stages of a document are quite valuable if they are related to the user's activities and events. In order to deal with the document life cycle, we specified a *Document Life Cycle Management* (DLCM) process, performing the task of modeling information about document activities which are either

1) activities which affect the document itself (creation, editing, and deletion) and according semantic information within the database needs to be updated, or
2) usage and management activities which do not affect the document itself (rendering, printing, publishing, sharing, retrieval, annotation, etc.).

An overview of the workflow is given in Figure 3. Document modifying activities invoke the application of *Update* and *Extension Rules* to the knowledge model.

In accordance to the instantiation process described in Section III-B, the modeling of metadata about the activities themselves is also a configurable, rule-based transformation process to provide flexibility and allow for substitution of the used ontology model. The input data for this modeling process should comply with a determined ontology which we call *Document Life Cycle Ontology (DLC)* – a purpose-built ontology to describe the above-mentioned document activities.

The DLC instantiation is triggered by an event handler which receives data from file system events, available context providers, and the management system itself (cf. Figure 3). To retrieve context information about a document's usage, we integrated the generic *Context Modeling Component* [24], already mentioned in Section III-B, which gathers and models cross-application context data from available context providers (e.g., from desktop applications, like e-mail clients, authoring tools, etc.).

Unlike the process of generating new semantic descriptions, as described in Section III, the update process of the

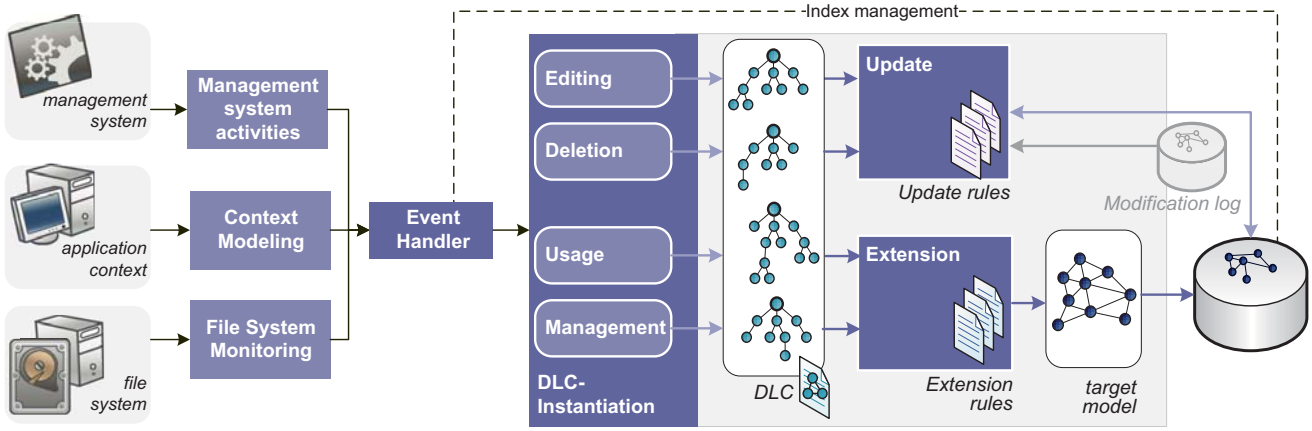


Figure 3. Document life cycle modeling process

DLC modeling process modifies existing semantic descriptions. This also means that user-created descriptions, i. e., descriptions of particular importance, might be altered unintentionally. To avoid this, we integrated a logging mechanism which records all user-driven activities in the repository (i. e., manual instance or statement creation, deletion, or editing) using a special logging ontology. The DLC modeling process accounts for the existing *modification log* (cf. Figure 3) by accomplishing only deletion and modification rules which do not effect user-generated content. To limit the size of the modification log it is advisable to apply suitable replacement policies, e. g., deleting log entries according to their timestamp (FIFO).

V. *K-IMM:* Architecture and Implementation

Based on the concepts described in Sections III and IV, we developed a prototypical personal multimedia document management system, designed as a service platform which autonomously manages documents stored on the local file system of the user. The import and indexing of multimedia assets (of different type) is performed by background tasks. An overview of the system is depicted in Figure 4. Three layers can be distinguished and are marked in the figure accordingly: (I) multimedia document indexing and analysis, (II) semantic data modeling and storage, and (III) a domain-dependent application layer.

(I) The document analyzing components extract specific properties and features (as described in Section III-A).

(II) The extracted data is passed to the *Semantic Modeling and Consolidation* component which provides the subcomponents for information instantiation and propagation, data validation and consolidation of the knowledge base, as well as the described document life cycle management. As mentioned in Sections III-C and IV-B, a *Semantic Web Search* component [23] and a *Context Management* component are connected to it to allow for the semi-automatic extension of semantic descriptions. The results of modeling and consolidation processes are stored in a persistent RDF/OWL repository using a third-party RDF/OWL API. The *Model Management and Processing* component provides an abstraction layer which allows for the substitution of applied RDF/OWL processing frameworks on the data layer. A component for *User and Rights Management*, described in more detail in [26], allows for sharing semantic descriptions with other users (on the local computer or via a remote RDF server).

(III) The topmost component (*Data Interface*) provides access to the modeled information for miscellaneous front-end applications for personal document management. On this level, application developers can configure or replace the used domain-specific ontology model and the corresponding rule sets (more details are given in Section V-A).

The overall architecture of the K-IMM system is realized in Java based on the OSGi [27] execution environment *Equinox* [28]. The diverse system components (described above) are implemented as OSGi *Service Bundles* which makes it possible to install, register, and start services (e. g., for multimedia analysis or user interface components) at runtime and on demand. Currently, there are three prototypical document analyzing components: an *ImageAnalyzer* for digital photographs, a *TextAnalyzer* for text documents, and an *AudioAnalyzer* for music files. RDF and OWL processing and storage is based on the Jena Semantic Web Framework [29], including its inference support for the application of rules and reasoning services. Particularly, we employ Jena's general purpose rule engine, its rule syntax and the concept of *Builtin primitives* [25] to pass data to corresponding Java modules, especially for evaluation and weighting algorithms within the process of data consolidation.

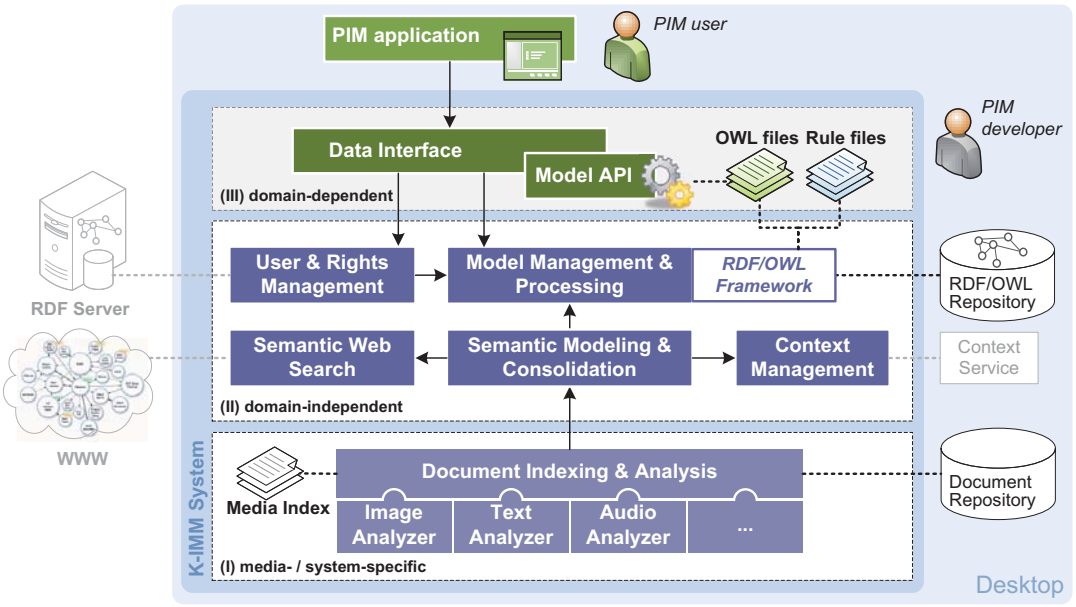


Figure 4. K-IMM Architecture (overview)

A. Domain-dependent Application Layer and Authoring Process

The Model API within the topmost layer of the architecture enables modeling, storing, and processing of instance data of the used ontology model. It provides an object-oriented access to the ontology-based dataset which is very useful for external software components (e. g., PIM applications) to access and edit information in an object-oriented way. Thus, external applications can create, modify or delete instances (e. g., if a user manually edits semantic descriptions) which is mapped by the Model API to appropriate operations on the RDF-based data layer. The Model API is *dynamically generated*, mapping OWL concepts to object-oriented classes with adequate methods for relations, with the help of an automated build-process. Thereby, at design time, the Model API is most flexible, allowing for substitution or modifications of the ontology model. Moreover, domain-specific rules used for the generation and consolidation processes, described in Sections III and IV, are kept in this layer. Thus, the developer is able to control these processes in order to meet the demands of the PIM application.

The intended authoring process comprises the following four steps:

(1) Building the application ontology in OWL (e. g., using Protégé [30]),
(2) Specifying the configuration settings for
- the instantiation process (with regard to the documentation of available Analyzer bundles),
- duplication handling (similarity metrics), and
- the compilation of SPARQL queries for the acquisition of context information and semantic web search results,

(3) Specifying rule sets for
- consolidation (semantic conflicts, incompletion),
- updating the document descriptions (DLC),
- extension (regarding available context information and "world knowledge"),
- integration (establishing relationships between documents),

(4) Implementing the front end application based on the object-oriented data interface.

A graphical representation is depicted in Figure 5.

The benefit of the system and its application layer is the separation of concerns: a declarative configuration of the application domain and its "business logic", and the imperative programming of the front end application. Developing suitable authoring tools, based on a linear, guided authoring process, is obviously worthwhile. Furthermore, it would also be possible to introduce distinct authoring roles, e. g., the *ontology designer* (a domain-expert, responsible for steps (1) and (3)), the *process designer* (responsible for step (2)) or the *user interface developer* (responsible for step (4), in general the most laborious task).

In the following we present an application example which shows the feasibility of our approach.

B. A K-IMM-based Desktop Application

Based on the exemplary implementation of the K-IMM System, we set up a desktop application based on the

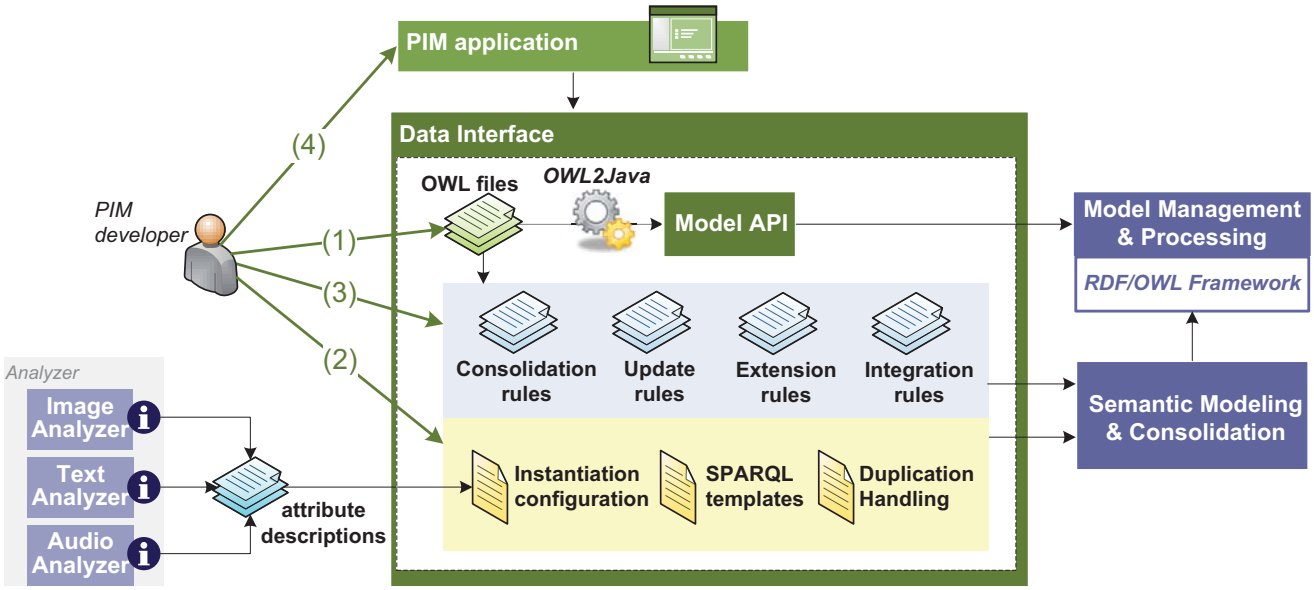


Figure 5. Authoring process of the PIM developer

Eclipse Rich Client Platform (RCP) [31]. The application uses the Data Interface and Model API to retrieve semantic information managed by the K-IMM System, modeled and managed as described in Sections III and IV. The used ontology is a purpose-built PIM ontology, consisting of 87 concepts and 173 properties. The necessary configuration and Jena Rule files include about 500 lines of code (LOC), whilst in contrast, the sophisticated graphical user interface (GUI) expectedly comprises more than 20.000 LOC in Java. A screenshot of the application is shown in Figure 6.

Figure 6. Screenshot of the demo application

The GUI provides several views to present and edit the available instances. Resources which have relations to spatial or temporal information (e. g., creation date/place) are visualized as pictograms in a *geographical view* (on the right), based on the Google Maps API [32], and in a *time-line view* (in the middle) which can be zoomed smoothly for different levels of detail [33]. The application allows for the unrestrained edition and creation of semantic descriptions, providing dynamically generated dialogs with appropriate data type verification.

The GUI also contains a so-called *Inspector* view in the bottom right corner. It presents currently existing and automatically detected problems with a human-readable description and proposed solution (cf. Section IV-A). Thus, the user can solve a conflict or delete the problem record with just one mouse click.

Furthermore, the application features a Semantic Web search widget, depicted in Figure 7. It illustrates the application of semi-automated gathering and integration of "world knowledge" found on the Semantic Web. As an example, the figure shows the extension of the person instance "Peter Jackson" using information from DBpedia.org [34] – with one mouse click.

VI. Conclusions and Future Work

In this paper we presented an integrated approach for ontology-based personal multimedia document management which covers the whole process of automated modeling of semantic descriptions for multimedia objects: document analysis, information instantiation, context-aware extension and integration, data consolidation, and observance of the document's life cycle. These aspects have been described in Sections III and IV. They provide the basis for the design

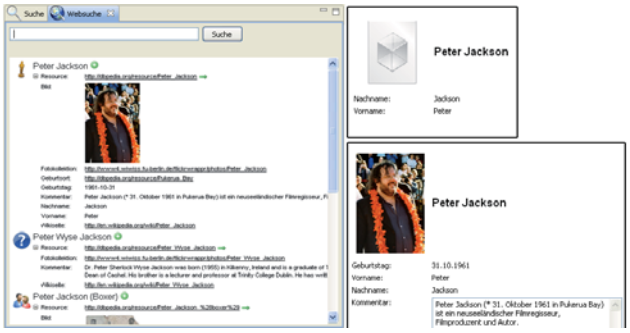


Figure 7. Detail of the demo application showing the widget for Semantic Web search

of a personal multimedia document management system, presented in Section V. It consists of three layers, separating *system-specific document analysis*, *domain-independent semantic data modeling and storage*, and the *domain-dependent application interface*. Thus, our solution allows for the domain-specific customization and substitution of the used ontology model and the corresponding modeling rules and configurations. The proposed approach is perfectly flexible regarding domain-specific alterations done by application developers, or regarding future document or metadata formats. We have proven the practicability of our concept by a prototypical implementation of the K-IMM system as an OSGi-based service platform for personal information management applications. As a demo application we created a comprehensive RCP front end, described in Section V-B.

We hope that this approach helps researchers and developers who pursue similar objectives. Of course, the benefit of this concept heavily depends on the usability and "added value" of suitable applications for personal information management. However, the rich, ontology-based datasets, which are automatically generated, consolidated and managed by the K-IMM system, form a proper basis to create advanced and lightweight front ends.

In the near future we will concentrate on more detailed performance and usability evaluation within our K-IMM-based personal desktop application for ontology-based multimedia document management. Additionally, as we have already adopted our system in other application scenarios (e. g., within the professional domain of construction process management addressed in the project BauVOGrid [35]), we would like to evaluate its feasibility within further application domains. Finally, to ameliorate and simplify the development process it would be reasonable to work on a convenient authoring tool which supports ontology design, specification of rules and configurations, as well as several authoring roles.

References

[1] A. Mitschick and K. Meißner, "Generation and maintenance of semantic metadata for personal multimedia document management," in *Proceedings of the First International Conference on Advances in Multimedia (MMEDIA 2009)*, Colmar, France, July 2009, pp. 74–79.

[2] D. R. Karger and W. Jones, "Data Unification in Personal Information Management," *Communications of the ACM, Special Issue: Personal information management*, vol. 49, no. 1, pp. 77–82, Jan. 2006.

[3] K-IMM Project, Chair of Multimedia Technology, TU Dresden, URL: http://mmt.inf.tu-dresden.de/K-IMM/, last accessed: 06/16/2010.

[4] V. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna, "Semantic annotation for knowledge management: Requirements and a survey of the state of the art," *Journal of Web Semantics*, vol. 4, no. 1, 2006.

[5] J. Kahan and M.-R. Koivunen, "Annotea: an open RDF infrastructure for shared Web annotations," in *WWW '01: Proceedings of the 10th international conference on World Wide Web*. New York, NY, USA: ACM, 2001, pp. 623–632.

[6] A. Kalyanpur, J. Golbeck, J. Hendler, and B. Parsia, "SMORE - Semantic Markup, Ontology, and RDF," Mindswap, Tech. Rep., Nov. 2002.

[7] S. Handschuh and S. Staab, "Authoring and Annotation of Web Pages in CREAM." in *Proceedings of the Eleventh International World Wide Web Conference, WWW2002*, 2002, pp. 462–473.

[8] M. Lux, "Semantische Metadaten: Ein Modell fr den Bereich zwischen Metadaten und Ontologien," Ph.D. dissertation, Graz University of Technology, 2006.

[9] A. Chakravarthy, F. Ciravegna, and V. Lanfranchi, "Cross-media Document Annotation and Enrichment," in *Proceedings of the 1st Semantic Authoring and Annotation Workshop (SAAW2006)*, 2006. [Online]. Available: http://www.dcs.shef.ac.uk/ ajay/publications/paper-camera-workshop.pdf

[10] B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, and M. Goranov, "KIM - Semantic Annotation Platform," in *International Semantic Web Conference*, ser. Lecture Notes in Computer Science, D. Fensel, K. P. Sycara, and J. Mylopoulos, Eds., vol. 2870. Springer, 2003, pp. 834–849.

[11] M. J. Weal, H. Alani, S. Kim, P. H. Lewis, D. E. Millard, P. A. S. Sinclair, D. C. De Roure, and N. R. Shadbolt, "Ontologies as facilitators for repurposing web documents," *International Journal of Human Computer Studies*, vol. 65, pp. 537–562, 2007.

[12] M. Yankova, H. Saggion, and H. Cunningham, "A Framework for Identity Resolution and Merging for Multi-source Information Extraction," in *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odjik, S. Piperidis, and D. Tapias, Eds. Marrakech, Morocco: European Language Resources Association (ELRA), May 2008.

[13] T. Groza, S. Handschuh, K. Moeller, G. Grimnes, L. Sauermann, E. Minack, C. Mesnage, M. Jazayeri, G. Reif, and R. Gudjonsdottir, "The NEPOMUK Project - On the way to the Social Semantic Desktop," in *Proc. of I-Semantics' 07*. JUCS, 2007, pp. pp. 201–211. [Online]. Available: http://www.dfki.uni-kl.de/ sauermann/papers/groza+2007a.pdf

[14] Aperture, Aduna Software, URL: http://www.aduna-software.com/technology/aperture, last accessed: 06/16/2010.

[15] L. Sauermann, G. A. Grimnes, M. Kiesel, C. Fluit, H. Maus, D. Heim, D. Nadeem, B. Horak, and A. Dengel, "Semantic Desktop 2.0: The Gnowsis Experience," in *Proc. of the ISWC Conference*, Nov 2006, pp. 887–900.

[16] G. Tummarello, C. Morbidoni, and M. Nucci, "Enabling Semantic Web Communities with DBin: An Overview." in *Proc. of ISWC 2006*, ser. Lecture Notes in Computer Science. Athens, GA, USA: Springer, 2006.

[17] A. Cheyer, J. Park, and R. Giuli, "IRIS: Integrate. Relate. Infer. Share." in *Proc. of 1st Workshop on The Semantic Desktop. 4th International Semantic Web Conference*, Nov. 2005, p. 15.

[18] D. R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha, "Haystack: A Customizable General-Purpose Information Management Tool for End Users of Semistructured Data," in *In CIDR*, 2005.

[19] S. Dasiopoulou, C. Saathoff, P. Mylonas, Y. Avrithis, Y. Kompatsiaris, S. Staab, and M. G. Strintzis, "Introducing context and reasoning in visual content analysis: An ontology-based framework," in *Semantic Multimedia and Ontologies*. Springer Verlag, 2008.

[20] S. Castano, A. Ferrara, S. Montanelli, and D. Lorusso, "Instance matching for ontology population," in *Proc. of the 16th Italian Symposium on Advanced Database Systems, 22-25 June 2008, Mondello, Italy*, 2008, pp. 121–132.

[21] P. Buitelaar, P. Cimiano, A. Frank, M. Hartung, and S. Racioppa, "Ontology-based information extraction and integration from heterogeneous data sources," *Int. Journal of Human-Computer Studies*, vol. 66, no. 11, pp. 759 – 788, 2008.

[22] A. Mitschick and K. Meißner, "Metadata Generation and Consolidation within an Ontology-based Document Management System," *Int. Journal of Metadata, Semantics and Ontologies*, vol. 3, no. 4, pp. 249–259, 2008.

[23] A. Mitschick, R. Winkler, and K. Meißner, "Searching Community-built Semantic Web Resources to Support Personal Media Annotation," in *Proc. of SemNet 2007, Int. Workshop located at ESWC2007*, Innsbruck, Austria, 2007.

[24] S. Pietschmann, A. Mitschick, R. Winkler, and K. Meißner, "CroCo: Ontology-Based, Cross-Application Context Management," in *Proc. of SMAP 2008*. Prague, CZ: IEEE Computer Society, December 2008.

[25] Jena Rules Documentation, Jena Semantic Web Framework, URL: http://jena.sourceforge.net/inference/, last accessed: 06/16/2010.

[26] A. Mitschick and R. Fritzsche, "Publishing and Sharing Ontology-Based Information in a Collaborative Multimedia Document Management System," in *WISE Workshops*, Nancy, France, Dec. 2007, pp. 79–90.

[27] D. Marples and P. Kriens, "The Open Services Gateway Initiative: An Introductory Overview." 2001. [Online]. Available: http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html

[28] Equinox Project, Eclipse Foundation, URL: http://www.eclipse.org/equinox, last accessed: 06/16/2010.

[29] J. J. Carroll, I. Dickinson, C. Dollin, A. Seaborne, K. Wilkinson, and D. Reynolds, "Jena: Implementing the semantic web recommendations," in *Proc. of WWW Alt. '04*, New York, USA, 2004.

[30] Protégé Ontology Editor and Knowledge Acquisition System, URL: http://protege.stanford.edu/, last accessed: 06/16/2010.

[31] Eclipse Rich Client Platform (RCP), Eclipse Foundation, URL: http://www.eclipse.org/rcp, last accessed: 06/16/2010.

[32] Google Maps API, Google Code, URL: http://code.google.com/apis/maps/, last accessed: 06/16/2010.

[33] R. Dachselt and M. Weiland, "TimeZoom: A Flexible Detail and Context Timeline," in *Proc. of CHI '06*. New York, NY, USA: ACM, 2006, pp. 682–687.

[34] DBpedia Knowledge Base, Linking Open Data community project, URL: http://dbpedia.org/, last accessed: 06/16/2010.

[35] A. Gehre, P. Katranuschkov, and R. Scherer, "Semantic Support for Construction Process Management in Virtual Organisation Environments," in *ECPPM 2008 - eWork and eBusiness in Architecture, Engineering and Construction - Proc. of the 7th European Conference on Product and Process Modelling (ECPPM)*, S. R. Zarli A., Ed. Taylor & Francis Group, Netherlands, Sep. 2008.

Relying on Testability Concepts to ease Validation and Verification activities of AIRBUS Systems

Fassely Doumbia, Odile Laurent
Systems Design
AIRBUS France
Toulouse, France
{Fassely.Doumbia, Odile.Laurent}@airbus.com

Chantal Robach, Michel Delaunay
Systems Design & Test
LCIS – Grenoble Institute of Technology
Valence, France
Chantal.Robach@esisar.grenoble-inp.fr

***Abstract*—The experiments, carried on AIRBUS systems, show that testability analysis can ease system formal detailed specifications validation activities. Indeed, testability information can highlight testing efforts, guide functional tests definition, facilitate detailed specification coverage analysis against system requirements, and support tests coverage analysis against formal detailed specification. This paper highlights the assessment result of testability concepts on AIRBUS systems.**

Keywords- requirement; data-flow design; testability flows; testability measures; testing strategy; test; coverage analysis

I. INTRODUCTION

Development of avionics systems must comply with the DO-178B standard [17]. The Validation and Verification (V&V) process is very demanding and contributes to high development costs. Therefore, innovative methods and tools that can alleviate and efficiently support V&V activities are of great interest for aeronautics domain. Functional testing is the most commonly used technique for systems requirements V&V. But, testing methods present some limits: exhaustive test data generation is most often unselected because of the size and the complexity of the systems. In this way, controlling testing effort is major, but systems quality demands are so big. Indeed, testing effort characterizes much as test scenarios and test data definition as error identification after fault detection during the diagnosis.

In this context, the testability analysis methodology proposed in this paper can offer useful methods to support the validation of systems formal detailed specification. Our testability approach deals with detailed specification relying on data-flow languages like SCADE [5]. The testability analysis proposed is based on SATAN [3, 7] (System's Automatic Testability ANalysis) technology. This approach determines testability flows and metrics that can be helpful information for system designers and system design validation engineers.

A significant number of research projects have dealt with software testability and a set of complexity metrics have been proposed. Each of these metrics address to either black-box or white-box testing technique. The black-box testing consists in considering the system under test as a box of which we know only the specification. Test data are generated from this specification which is most often formalized. The black box testing does not consider the internal structure of the system and defined test data are independent of the implementation. Freedman [6] and Voas and Miller [11] defined some testability approaches related to the black box testing technique. Freedman introduced the domain testability of software components based on controllability and observability. A component becomes observable when it produces distinct outputs from different inputs. A component is controllable when its specified output field corresponds to its produced output. Voas and Miller proposed the DRR (Domain / Range Ratio) metric, which exhibits fault-hiding tendencies of software subcomponents considering the input and the output field cardinality. This technique can be used to predict a subcomponent's ability to cause program failure if it contains a fault.

The white box testing technique is based on the internal structure of the system under test to define test data and coverage criteria. This technique assumes the program under test is available as well as the system specification. McCabe [8], Nejmeh [9] and Do, Le Traon and Robach [2, 3, 7] proposed some approaches based on the internal structure of a program. McCabe defined the Cyclomatic number which measures the number of linearly independent paths through the control graph built by using a program source code. Nejmeh introduced the Npath metric which computes the number of possible execution paths through a function. Do, Le Traon and Robach proposed a testability measurement applicable to data-flow designs. The study presented in this paper is based on the same approach.

This paper proceeds as follows: Section II introduces SATAN technology and the associated testability analysis concepts. Section III describes the AIRBUS flight control systems validation process. Section IV proposes some adaptation techniques for Airbus systems analysis. The defined methodology which aims at enhancing systems validation process is presented in Section V. Experiments results are depicted in Section VI. Finally, conclusion and perspectives are given in section VII.

II. TESTABILITY ANALYSIS

Fig. 1 gives a simple view of testability analysis, proposed by SATAN technology. This approach has been initially applied to hardware systems [3]. Further studies [2, 7] demonstrated that this approach could be used for analyzing the testability of data-flow designs. This method

provides metrics that allow the identification of critical parts (which can be hard to test). It also identifies testability flows which are a set of operators involved in the computation of one output from relevant inputs. They can be used to guide tests definition in the system design validation cycle.

The SATAN approach is based on a testability model that represents the transfer of information into the system. This model is called *ITM* "Information Transfer Model" (Section A). Testability flows are identified from this ITM (Section B). Measures quantified the testability of system components are calculated from these testability flows (Section C). Test strategies (Section D) can be applied to select relevant testability flows to help the tests generation process.

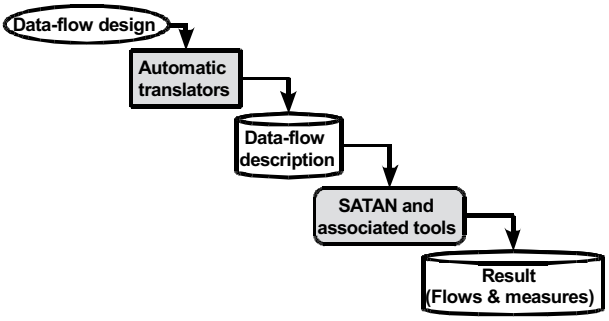


Figure 1. Testability analysis process.

A. Information Transfer Model

The testability model is a graph defined by a set of places, transitions and arrows. Places represent inputs, constants, functional modules, outputs and test injection points specified in the system. Transitions express information transfer modes between places. Arrows connecting places and transitions represent information media throughout the system.

Three different information transfer modes Fig. 2 are used in an ITM:

- Junction mode: the destination place needs information from all source places;
- Attribution mode: the destination place needs information from one of several source places;
- Selection mode: the same information is sent from the source place to some destination places.

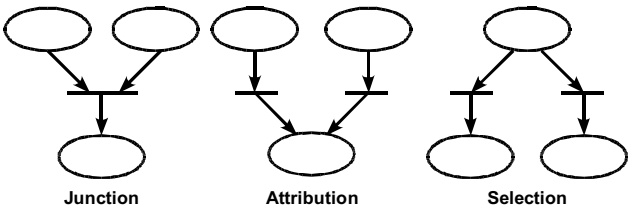


Figure 2. Information transfer modes.

A data-flow description of a system is hierarchically composed of operators. Each operator has an elementary ITM. This model corresponds to the data-flow representation of the operator. According to the level of testability analysis, this elementary model can be more (or less) detailed. The basic representation of an operator associates a functional module to each output. Graphic representations Fig. 3 illustrate the notion of elementary ITM.

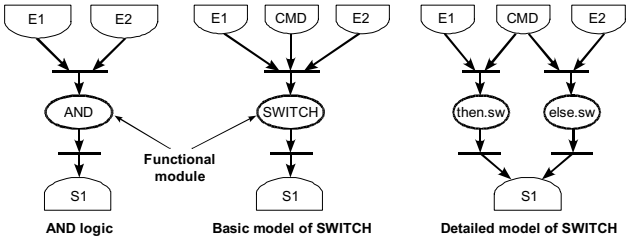


Figure 3. Elementary ITM of AND logic and SWITCH operators.

In this representation, inputs and outputs are depicted by semicircles, modules by circles, and transitions by bars.

The system ITM is obtained by concatenating the elementary ITMs.

B. Testability flows

SATAN technology identifies flows from a system ITM. A testability flow is an information path that carries information from one or several inputs, through modules and transitions, to one output. Several testability flows can be associated to an output.

The system specification represented in Fig. 4 contains sixteen testability flows. Two testability flows (F_1 and F_5) are depicted using bold lines. These flows are described below by a set of modules and output.

F_1 = {NOT_2, then.SWITCH_2, OR_2 | O2}

F_2 = {NOT_2, then.SWITCH_2, OR_2, then.SWITCH_4, OR_3, else.SWITCH_5 | O1}

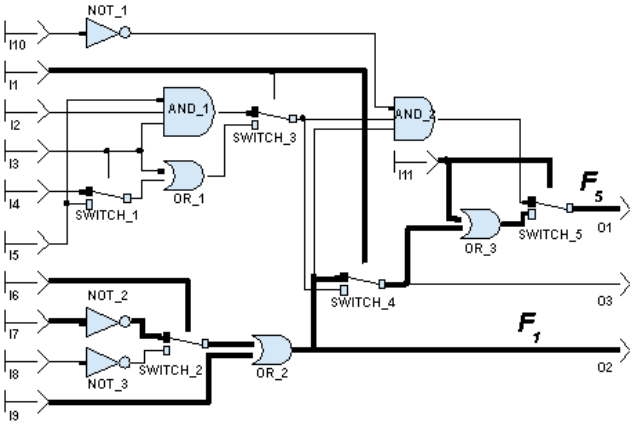


Figure 4. Graphical representation of a testability flow

C. Testability measures

Two different measures are defined to characterize the testability of a component using SATAN approach: the controllability and the observability. The controllability expresses how ease the input values of an internal component can be controlled through the input values of the system. The observability expresses how ease the results of an internal component can be observed at the outputs of the system. Fig.5 illustrates these measures.

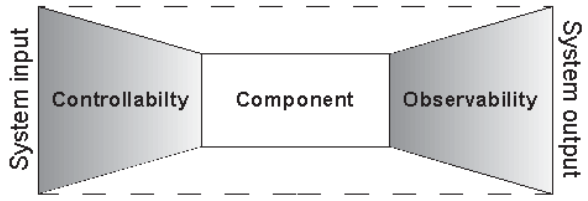


Figure 5. Controllability and observability of a component

These measures can be calculated for each defined testability flow. SATAN associates these measures to all the system ITM modules. Therefore, the testability measures of a component are deduced from modules which represent it in the ITM. Testability measures computation is based on information theory. Let F be a testability flow and M a module activated by F, we can define the following variables:

- I_F represents the sources of F;
- O_F is the output of F;
- I_M represents the input of M;
- O_M is the output of M.

The controllability of a module M is computed as follows: if M is isolated, all the possible combinations of its input values can be produced. We denote $C(I_M)$ the input capacity of M corresponding to the maximum information quantity on its inputs when it is isolated. If M belong to the testability flow F, the maximum information quantity that can be brought to its inputs is denoted $T(I_F;I_M)$. The controllability of M is expressed as follows:

$$Cont_F(M) = \frac{T(I_F;I_M)}{C(I_M)}$$

In the same way, the observability of a module M is computed as follows: the maximum information quantity produced by M is $C(O_M)$ when it is isolated. The maximum information quantity which can be delivered to the output of F from O_M is denoted $T(O_M;O_F)$. The following equation defines the observability of M:

$$Obs_F(M) = \frac{T(O_M;O_F)}{C(O_M)}$$

To determine the information quantities $T(I_F;I_M)$ and $T(O_M;O_F)$, the approach uses the Information transfer Net (ITN) to simulate the transfer of information in the system. The ITN is a weighted ITM, whose elements are associated with their information capacities. These capacities quantify the information carried by the ITM elements using bits as information unit.

- The information capacity of the ITM sources are determined from their data types. The capacity is equal to 1 for Boolean, 8 for Integer (when we reduce the domain to 2^8 elements), etc.
- The capacity of a module corresponds to the information quantity of its output. Indeed, we suppose that the output takes its value in a finite set. A probability is associated with the occurrence of each element of this set at the output. In this context, the capacity is equal to the entropy of the module output variable. The Information Loss Coefficient (ILC) of a module is determined from its capacity. It expresses the intrinsic loss of information of each ITM module. The ILC corresponds to the ratio between the effective capacity and the maximum capacity of the module [12, 13].
- The capacity of an arrow leaving from a place is equal to the capacity of that place.
- The capacity of an arrow arriving to a place is equal to the capacity of this place.
- In the junction information transfer mode, the capacity of the arrow that leaves the transition is the sum of capacities of all arrows arriving at the transition.
- In the selection mode, the capacity of each arrow leaving from the source place corresponds to the capacity of this source place.
- In the attribution mode, the capacity of the destination place is equal to the maximum value of capacities associated with arrows arriving at this place.

As to build the ITM, the ITN is obtained from elementary ITNs of system operators. “Fig 6” presents the ITM (a) and the ITN (b) of the AND operator.

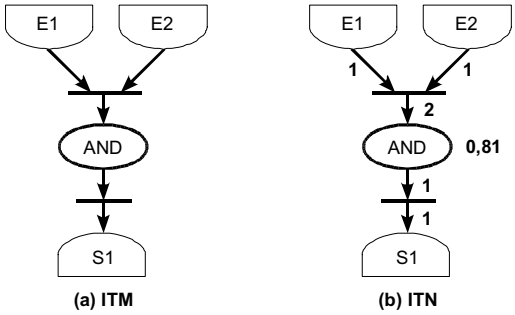


Figure 6. ITM and ITN of the operator AND logic

As previously mentioned, testability measures can be calculated for each testability flow. The approach described in this paper considers only selected flows after the application of a test strategy in order to alleviate the testability analysis process.

D. Test strategies

A test strategy defines the way to conduct test activities and to analyze test results. Test strategies allow us to select a set of relevant testability flows for testing purpose. Flows are chosen according to the following criterion: every place in the graph must be activated at least once in order to ensure the coverage of all operators. SATAN supports three strategies: *Start-Small* (progressive structural strategy) [10] suitable for the progressive detection of faults during system validation, *Multiple-Clue* (cross-checking strategy) [10] suitable for diagnosis during maintenance and *All-paths* [10] which chooses all flows contained in the ITM. We will focus on the Start-Small strategy in this paper.

The Start-Small strategy gradually covers the modules by choosing flows with an increasing number of covered modules. The main idea of this strategy is to minimize test and diagnosis efforts. The first testability flow to be tested contains the minimal number of modules. The next one contains a minimal number of modules that are not activated yet. In this strategy, a new flow is tested only if all faults detected in previous flows are corrected, as depicted in Fig. 5. Tests are defined for each selected flow.

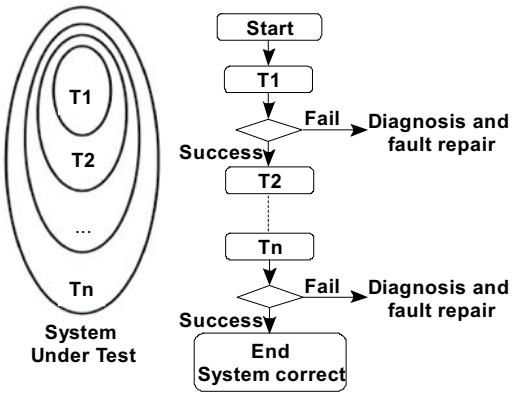


Figure 7. Start-Small strategy illustration

Start-Small strategy selects eight relevant testability flows instead of the sixteen determined for the system specification depicted in "Fig 4".

Choosing a relevant testing strategy depends on industrial practices and system development stage. In the next section, we present the AIRBUS system validation and verification process in order to point out the detailed specification validation activities.

III. AIRBUS VALIDATION AND VERIFICATION (V&V) PROCESS

In this section, we focus on AIRBUS flight control and Auto flight systems V&V activities. Flight control systems allow the pilot to control the aircraft in flight. Auto flight systems allow maintaining the flight path defined by the crew. They also control the aircraft and the engines. The V&V cycle of these reactive systems relies on a generic V-cycle process for which validation activities are added due to the modelling process at system level Fig. 6.

Three main levels can be identified in this V&V process.

- The "*Aircraft level*" is composed of aircraft level requirements definition, aircraft simulation, ground and flight tests activities.
- The *"System level*" represents the system specifications and design definition stage. The system specification validation activities are led in the phase.
- The "*Equipment level"* corresponds to the implementation of system specifications and designs in real equipment.

We will focus on systems validation activities in the rest of the paper.

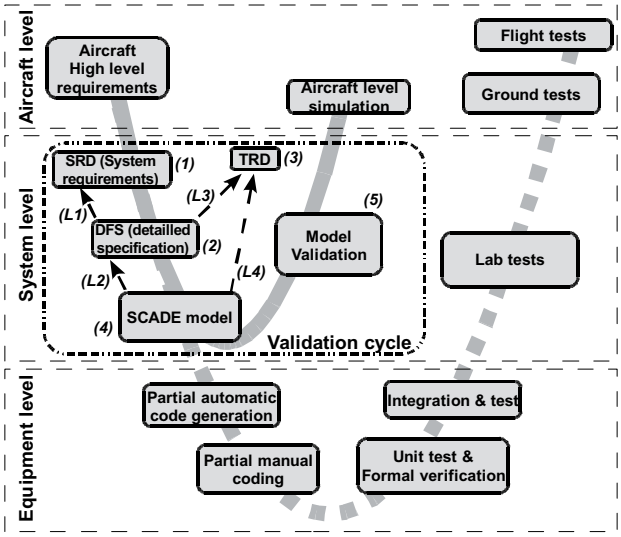


Figure 8. AIRBUS Flight control systems V&V process

A. System validation cycle

System validation activities are mainly based on testing and traceability activities. The model validation activities (5) consist in executing tests on a desktop simulator dedicated to flight control system. This simulator embeds system functions code automatically generated from the SCADE model. The traceability process relies on a documents cascade and on a SCADE model managed by the DOORS tool [5]:

- *SRD* (System Requirements Document) (1) specifying system requirements refined from aircraft requirements.
- *DFS* (Detailed Functional Specification) (2) document describing the system detailed functions that meet the system requirements.
- *TRD* (Test Requirements Document) (3) defining for each DFS function the tests data (functional tests description, tests vectors and expected tests results).
- SCADE model (or detailed specification) (4) corresponding to a formal implementation of the detailed functions and from which the embedded code is automatically generated.

The traceability activities, based on DOORS tools, consist in checking that:

- All the system requirements described in the SRD are considered in the system functions (L1);
- The SCADE model implements only once all the systems functions specified in the DFS (L2) (neither under-specification nor over-specification);
- The tests defined in TRD cover all the functions of the DFS (L3) and the functions of the SCADE model (L4) (no missing tests).

In order to alleviate drastically the coverage analysis activities described above, we propose to rely on the testability flows determined by SATAN on the SCADE model.

Indeed, if we can demonstrate that there is a clear link between testability flows and DFS requirements, the relationship between SCADE model and DFS becomes obvious. Another important issue of this approach in the validation process is the definition of relevant set of tests ensuring the coverage of the SCADE model. Enough tests have to be identified to cover all the system requirements, but redundant tests must be avoided for cost-efficiency reasons.

The next Section IV presents developed methods in order to take AIRBUS systems specific characteristics into account during testability analysis.

IV. ADAPTING THE TESTABILITY ANALYSIS TO THE AIRBUS CONTEXT

Appling the testability analysis on AIRBUS systems needs the development of new testability concepts. Indeed, systems specification use specific operators defined in the AIRBUS in-house libraries in addition to those predefined in the SCADE environment. The testability analysis must build ITM for each specific operator and integrate new approaches in order to comply with system validation functional tests definition process. We first describe the proposed technique for modeling AIRBUS specific operators ITM and ITN (Section A). Indeed, previous research works highlight the interest of testability measures on some systems [12, 15]. In this paper, we will check the relevance of these measures in the AIRBUS systems functional testing context. Then, we define testability flows classification method relying on the different phase of systems operation (Section B). Finally, we present defined testability approaches for systems validation (Section C).

A. AIRBUS specific operators ITM and ITN

The in-house libraries operators can be mainly split in two categories: combinative and temporal operators. Indeed, SCADE uses the synchronous approach for systems specification [14]. In this approach, the time is divided into discrete instants (cycles) defined by a global clock. At instant t, the system receives input i_t from its external environment, and computes output o_t. Fig. 9 illustrates the synchronous approach principles.

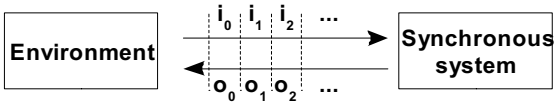


Figure 9. Synchronous mechanism

The synchronous hypothesis expresses that the computation of the output values is made instantaneously at the same instant t.

The combinative operators compute the output considering information of the current cycle. Arithmetic and logical and switch operators belong to the category of combinative operators. SATAN already proposes techniques to model these operators.

Temporal operators compute the output considering information of previous cycles (operators memory) in addition of the current one. These operators implement the past linear temporal logic. The flip-flop, state change confirmation and state change stabilizer operators belong to the category of temporal operators. This temporal aspect is not currently taken into account by the testability analysis. We use the flip-flop with priority to reset (BASCR) to describe the ITM and ITN modeling techniques in order to integrate the temporal aspect into the testability analysis proposed by SATAN.

Fig. 10 highlights the graphical representation of BASCR. Four different inputs and one output can be identified for this operator.

- E_S is the "Set" input (boolean);
- E_R is the "Reset" input (boolean);
- $Init$ represents the initialization value (boolean);
- B_Init corresponds to the initialization boolean;
- $S1$ is the output (boolean) of the operator.

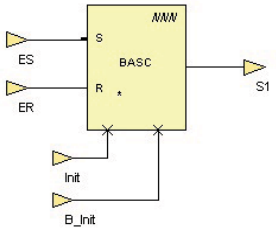


Figure 10. SCADE representation of BASCR

The algorithm highlighting the behavior of this operator is described below. We denote $E(k)$ as the value of the flot E at the cycle k.

Initialization

If $B\_Init(k) = True$ *Then* $S1(k) = Init(k)$

Else Calculation

Calculation

If $E_R(k) = True$ *Then* $S1(k) = False$

Else If $E_S(k) = True$ *Then* $S1(k) = True$

Else $S1(k) = S1(k-1)$

1) ITM of BASCR operator

The data-flow modeling of BASCR for testability analysis is represented using two modules (see Fig. 11).

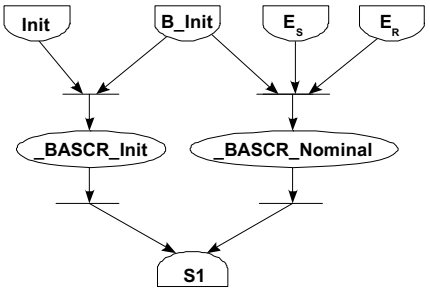


Figure 11. ITM of the temporal operator BASCR

The first module (_BASCR_Init) computes the value of $S1$ when B_Init is *True* from the input *Init* in the initialization cycle. The module "_BASCR_Nominal" computes $S1$ from E_S, E_R and the value of $S1$ in previous cycle when B_Init is *False*.

2) ITN of BASCR operator

The ITN modeling of BASCR operator returns to calculate the capacity of its modules ("_BASCR_Init" and "_BASCR_Nominal").

The basic hypothesis is to consider the occurrence of an operator inputs value as independent events from a cycle to another one. Indeed, it is difficult to make a correlation between these values according to the complexity of their production and the use of the operator in the system specification. We also suppose that these events comply with the Bernoulli distribution [16].

The "_BASCR_Init" module produces *Null* when B_Init is equal to *False or* 0 and calculates $S1$ when B_Init is equal to *True or* 1. As result, three different values (*Null*, 0 and 1) can be observed on the output of this module. Therefore, the maximum capacity of the output is equal to $Q = \log_2(3)$. We denote P_B the probability of the event $B\_Init = 1$. Let $P_{i\_m}(A)$ be the occurrence probability of A at the output of "_BASCR_Init".

$$P_{i\_m}(Null) = P(B\_Init = 0) = 1 - P_B$$

$$P_{i\_m}(0) = P(B\_Init = 1 \text{ and } Init = 0) = P_B \times P(Init = 0)$$

$$P_{i\_m}(1) = P(B\_Init = 1 \text{ and } Init = 1)$$

$$= P_B \times P(Init = 1) = P_B \times (1 - P(Init = 1))$$

The capacity of this module can be expressed as follows:

$$Capa(\_BASCR\_Init) = -\sum_{j=\{Null,0,1\}} P_{i\_m}(j) \times \log_2(P_{i\_m}(j))$$

In addition, the ILC (Information Loss Coefficient) of this module is defined in the following expression.

$$ILC(\_BASCR\_Init) = \frac{Capa(\_BASCR\_Init)}{\log_2(3)}$$

Considering the module "_BASCR_Nominal", it produces *Null* when $B\_Init = 1$ and calculates $S1$ when $B\_Init = 0$. As result, three different values (*Null*, 0 and 1) can be observed on the output of this module. Therefore, the maximum capacity of the output is equal to $Q = \log_2(3)$. Let $P_{n\_m}(A)$ be the occurrence probability of A at the output of "_BASCR_Nominal". We denote p the probability of the events $P(E_R(k) = 1)$ or $P(E_S(k) = 1)$.

$$P_{n\_m}(Null) = P(B\_Init = 1) = P_B$$

$$P_{n\_m}(1) = P(B\_Init = 0 \text{ and } S1(k) = 1)$$

$$= (1 - P_B) \times P(S1(k) = 1)$$

When $B\_Init = 0$, the probability of the event $S1(k) = 1$ can be expressed using the following table.

$E_R(k)$	$E_S(k)$	$S1(k)$
0	0	$S1(k-1)$
1	0	1
0	1	0
1	1	0

$$P(S1(k) = 1) = P(E_R = 1 \text{ and } E_S = 0)$$

$$\text{or } P(E_R = 0 \text{ and } E_S = 0 \text{ and } S1(k-1) = 1)$$

$$\Rightarrow P(S1(k) = 1) = (p \times p) + ((1-p) \times (1-p) \times P(S1(k) = 1))$$

$$P(S1(k) = 1) = \frac{1-p}{2-p}$$

Therefore, $P_{n\_m}(1) = (1 - P_B) \times \frac{1-p}{2-p}$

$$P_{n\_m}(0) = P(B\_Init = 0 \text{ and } S1(k) = 0)$$

$$= (1 - P_B) \times (1 - P(S1(k) = 1))$$

$$P_{n\_m}(0) = (1 - P_B) \times \frac{1}{2-p}$$

The capacity of this module is:

$$Capa(\_BASCR\_NoMinal) = -\sum_{j=\{Null,0,1\}} P_{n\_m}(j) \times \log_2(P_{n\_m}(j))$$

The ILC is defined in the following expression.

$$ILC(\_BASCR\_NoMinal) = \frac{Capa(\_BASCR\_NoMinal)}{\log_2(3)}$$

These capacities depend on the probabilities P_B and p. Fig. 12 presents an ITN of BASCR with $P_B = 0{,}5$ $p = 0{,}5$ and $P(Init = 1) = 0{,}5$.

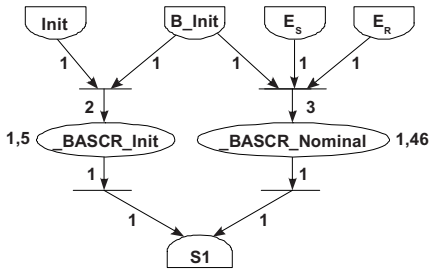


Figure 12. ITN of the temporal operator BASCR

This modeling technique can be applied to other temporal operators. It introduces new aspects (the state of memory and the initialization phase of operators) in the testability measures computing process.

B. Testability flows classification

Testability flows are the basic element of the proposed testability analysis. Indeed, they represent system elementary functions. The initialization phase modeling for temporal operators ITM leads to the identification of two different categories of testability flows. These categories reflect the normal operation of reactive systems (initialization and nominal phases).

- The initialization phase corresponds to the program memory initialization in order to ensure a

deterministic behavior. However, the memories of all the used temporal operators represent the memory of the program. In this instance, this phase returns to the initialization of these operators.

- The nominal phase represents the cyclic operation of the program. This periodic operation is composed of three main parts: the inputs reading, the outputs computing and the memory updating. In a synchronous paradigm, this period corresponds to a cycle.

In order to make the testability analysis more representative of systems operation, we define two classes of testability flows corresponding to each operation phase. The method of testability flows classification is illustrated in aid of the Fig. 13.

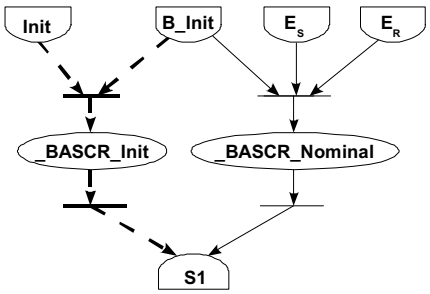


Figure 13. An ITM representing the two classes of flows

As previously described, the ITM of the BASCR operator is composed of two modules representing respectively the initialization and the nominal phases. Considering this ITM, one flow coming from "_BASCR_Init" or "_BASCR_Nominal" is required to activate *S*1. Indeed, it corresponds to the attribution mode (see Fig. 2). Therefore, the classification method consists in separating testability flows which contain initialization modules from the other testability flows. Referring to Fig. 13, the dotted line represents the flow which activates "_BASCR_Init". This technique can be applied to other temporal operators and information transfer modes. It allows the identification of testability flows (functions) which are activated in the both phases. This can be interesting for test definition issue.

C. Testability analysis approahces

Testability analysis information is useful only if it reflects the development process. In this section, we present three testability approaches defined for AIRBUS needs.

1) Output variable approach

This first approach consists in identifying all the SCADE output variables involved in the DFS. The part of the SCADE model related to each output variable is extracted. Thereby, testability flows are determined for each extracted part of the SCADE model. This approach allows a local testability view of the system for all output variables.

2) Set of output variables approach

This second approach consists in identifying the set of output variables per function. The part of the SCADE model related to this set of output variables is extracted and testability flows are determined. It allows the testability assessment for each function defined in the system.

3) Component approach

In this third approach, we propose to split the SCADE model into independent parts from a data-flow point of view (called hereafter components). This approach consists in extracting each component and performing the testability analysis. It allows the analysis of system independent parts separately.

To illustrate testability approaches, we use the system specification depicted in Fig. 4. "Tab I" below summarizes the result when applying these methods. We assume O_1 performs a function and O_2 and O_3 perform another function.

We observe that the number of selected testability flows decreases from (1) to (3). Indeed, the description of some output variables can share SCADE model parts.

- In approach (1), these common parts are analyzed for each output variable. Thereby, an important number of testability flows is determined. Fig. 14 highlights the notion of common parts. The part (A) computes the output variable O_2 and (B) performing O_3 involves (A). Using this approach, testability flows determined for (A) are also considered during flows identification for (B). (A) is then a common part of the model.
- Regarding the approach (2), common parts used by a set of output variables related to a function are analyzed once during testability analysis. Common parts related to different functions are analyzed several times. So, the number of testability flow decreases compared to (1) but is still high. The part (A) performs O_2 and O_3 output variables; (B) performing O_1 involves (A) (see Fig.14).

TABLE I. TESTABILITY APPROACHES ILLUSTRATION

	Methods		
	Output variables (1)	Set of output variables (2)	Component (3)
Number of selected flows	**12** (6 for O_1, 2 for O_2 and 4 for O_3)	**11** (6 for the *1st function* and 5 the *2nd function*)	**8** (5 for O_1, 2 for O_2 and 1 for O_3)

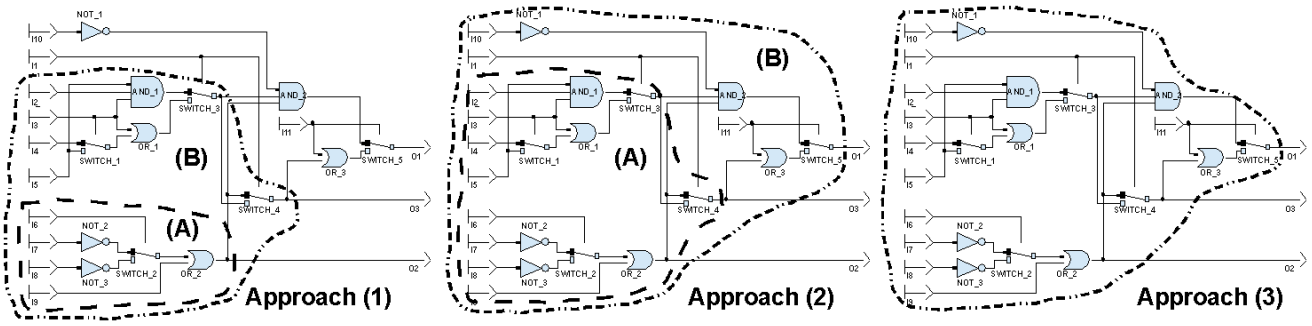


Figure 14. Specification common parts illustration

- In approach (3), all SCADE model related parts are analyzed together. Each part of the specification is explored only once during testability analysis. As a result, testability flows determined by this approach represent the lowest number compared to the two others. The specification depicted in Fig. 14 contains only one component.

We present, in this section, the main developed methods for AIRBUS systems testability analysis. The next section (Section V) defines a methodology based on testability analysis to support system coverage analysis.

V. TESTABILITY ANALYSIS METHODOLOGY

The methodology defined for AIRBUS systems is based on DFS document, TRD document and SCADE model. It is composed of three main activities: a testability approach application, the coverage assessment of the requirements against the SCADE model, and the test cases against the SCADE model.

A. Testability approach application

The testability analysis stage is depicted in Fig. 15. It consists in applying one of the three testability approaches defined in the previous section. Relevant testability flows are selected using the Start-Small strategy (Section II). These flows are exploited for coverage activities. Testability measures are determined from selected flows. The relevance of these measures is checked during experiments (Section VI).

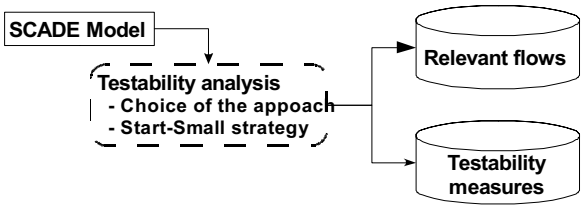


Figure 15. Testability approach application stage

B. SCADE model coverage against system requirements

Systems functional requirements have to be specified into a formal model for their validation. This coverage analysis uses relevant testability flows selected in the previous section to give information about the completeness of the formal specification. Fig. 16 presents the SCADE model coverage analysis process.

This coverage analysis is mainly composed of three activities.

a) Requirements output variables identification: This activity consists in using the DFS document to identify SCADE ouput variables involved in each system requirement definition. The identification is performed manually.

b) SCADE model coverage analysis: Testability flows are associated automatically with output variables in this step. This coverage activity allows highlighting a link between requirements and testability flows. It also points out the presence of:

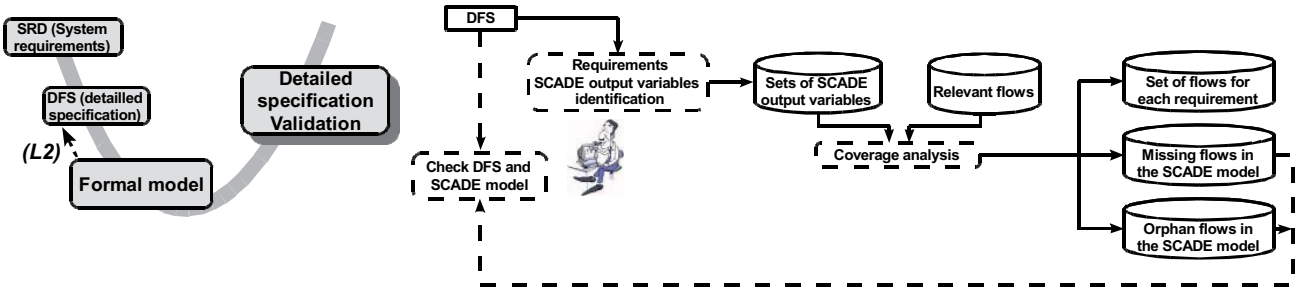


Figure 16. SCADE model coverage analysis against requirements process

- *Orphan flows*: Is called orphan a flow which has not any association with a requirement. The identification of these flows highlights the presence of SCADE output variables which are not defined in the DFS. The presence of these flows points out either a possible over-specification of the SCADE model or the implementation of derived requirements in the SCADE model.
- *Missing flows*: This situation is due to the absence of flows associated with some output variables. The detection of these outputs means that no SCADE model part corresponds to these output variables. It highlights the SCADE model is incomplete: the implementation of some DFS requirements is missing.

In addition, for each DFS requirement, the minimum number of tests to be defined can be deduced from the number of testability flows: at least one test per flow. It allows evaluating the testing effort.

c) SCADE model check: It consists in analyzing orphan and missing flows data and may lead to modifications in the SCADE model: suppress the SCADE part related to orphan flows; add the SCADE part related to the unimplemented DFS requirements highlighted by the missing flows. This checking activity needs human intervention.

C. Tests coverage against SCADE model

Testing is the dynamic verification technique led on the simulated code of the system in order to verify the completeness and the correctness of implanted requirements. This coverage analysis uses testability flows associated with system requirements in the previous section to give information about defined test cases for validation. Fig. 17 presents the tests coverage analysis process.

This tests coverage analysis is mainly composed of three activities.

a) Requirements test cases identification: This activity consists in using the TRD to identify test cases associated with each requirement described in the DFS. The identification is performed manually.

b) Tests coverage analysis: This step highlights the link, for each requirement, between relevant testability flows and identified test cases. When a relevant flow associated with a requirement cannot be linked to a described test in the TRD, a missing test is identified. When that a same flow is linked to different tests, potential redundant tests are identified. It is also pointed out that the number of determined flows is dependant of the ITM modeling (Section II). In the current study, the ITM modeling principle is based on branch and decision coverage criteria.

c) TRD document check: This third activity consists in analyzing missing and potential redundant tests data and may lead to modifications in the TRD. These modifications could be: the definition additional tests related to testability flows which are not associated with any test and the deletion tests when analysis shows they are functionally redundant.

Otherwise, the relation between relevant flows and test cases can be used to define tests scheduling. Indeed, Start-Small strategy (Section II) proposes an order of execution of the tests related to the selected flows. This order aims at minimizing diagnosis effort.

In the following section, we apply our methodology to an academic example and present the results of its use for two industrial case studies.

VI. CASE STUDIES

In this section, we first depict two experiments on coverage analyses: the pumping system which is academic and two Auto-flight systems which are AIRBUS operational systems. Then we give some conclusion on the relevance of the testability measures in the AIRBUS systems functional testing process.

A. Coverage analyses

For interpretation purpose: (1) means the "Output variable approach", (2) means the "Set of output variables approach" and (3) means the "Component approach". We will focus on nominal tests coverage analysis in this paper.

1) A house pumping system

This academic case study controls the water supply of a house. Two pumps are used to specify this system: the first one brings up water from a well to fill a tank; the second pump supplies water to the residence. Three main functions can be defined for controlling this system: one manages the first pump; another controls the second pump and the last one performs the system's global status. These functions can be described by the following requirements:

- R1: the system shall actuate the first pump when the water-level in the tank decreases and reaches the "filling level";

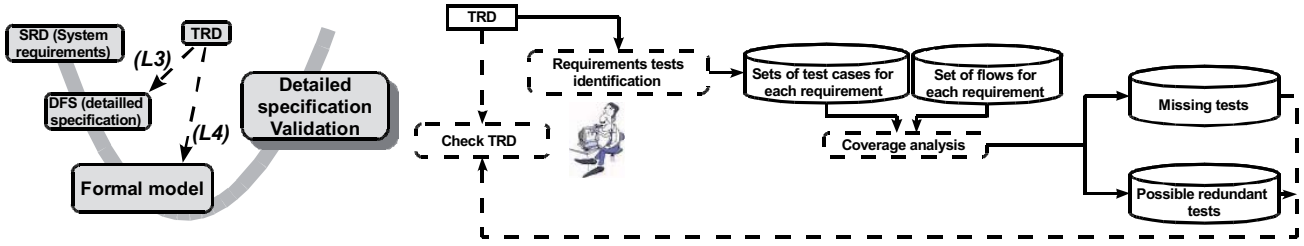


Figure 17. Tests coverage analysis against SCADE model process

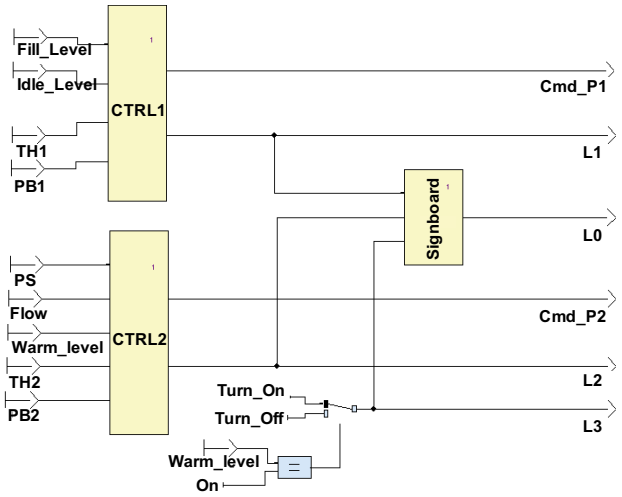


Figure 18. Formal specification of pumping system

- R2: The pumping shall stop when the water-level in the tank raises and reaches the "pumping idle level";
- R3: The first pump shall stop running when the temperature of the pump is abnormally high (only a push button can actuate it).
- R4: The second pump shall be actuated by the decrease in pressure due to the turning on of a tap in the residence. Its shutdown is provoked by the turning off all the taps in the residence;
- R5: the system shall warn and idle the second pump when water reaches the "warning level" (the water-level is lower than the "filling level");
- R6: The pump shall be idled when the water flow rate is too low;
- R7: The pumping shall stop when its temperature is abnormally high (only a push button can actuate it).
- R8: the system shall indicate its global status by taking into account the two pumps.

The following figure Fig. 18 shows pumping system SCADE model.

The "Tab. II" gives the result of testability analysis process on this case study. It shows that the number of tests (48) is higher than the number of testability flows (45, 45 or 27) selected using respectively (1), (2) or (3).

We observe that the number of selected testability flows is stable from using (1) to (2) in this case study. This is mainly due to the structure of the SCADE model. Indeed, each output variable corresponds to a function description. The number of selected flows decreases from using (2) to (3).

The requirement R8 is verified in combination with the seven other requirements. Indeed, L0 represents the state that is checked what ether the pump behavior.

An example of functional tests defined for requirement R1 is described as follow: *"the water-level reaches the filling level (Fill_level = true and Idle_level = false); the pump temperature is normal (TH1 = false and PB1 = false); as result the first pump is activated (Cmd_P1 = true)"*.

The experiment result analysis outlines interesting points:

- For this academic example, we found no orphan testability flows; neither missing implemented requirement in the SCADE model. Indeed, this specification is a final version which has been tuned;
- In order to introduce missing test problem, we deliberately omitted to define tests verifying the second pump is running without output rate (R5). As a result, three testability flows have not been linked to any test;
- For redundant tests, we identify several tests verifying the second pump behavior when its temperature is too high (R7). These tests are linked to the same testability flows. Consequently, they can be considered redundant because they all address the same pump behavior.

TABLE II. PUMPING SYSTEM ANALYSIS RESULT

Functional requirements	Output variables	Number: Defined tests	Number: Selected nominal flows (1)	(2)	(3)	Number: *Selected initialization flows* (1)	(2)	(3)	Input variables
R1 R2	Cmd_P1	6	**5**	**5**	**5**	*5*	*5*	*5*	*Fill_level*; *Idle_level*; *TH1*; *PB1*.
R3	Cmd_P1 L1	6	**6**	**6**	**4**	*6*	*6*	*4*	*Fill_level*; *Idle_level*; *TH1*; *PB1*.
R4	Cmd_P2	6	**4**	**4**	**4**	*4*	*4*	*4*	*PS*; *Flow*; *Warn_level*; *TH2*; *PB2*.
R5	Cmd_P2 L3	6	**3**	**3**	**3**	*2*	*2*	*2*	*PS*; *Flow*; *Warn_level*; *TH2*; *PB2*.
R6 R7	Cmd_P2 L2	24	**13**	**13**	**9**	*13*	*13*	*9*	*PS*; *Flow*; *Warn_level* *TH2*; *PB2*.
R8	L0	48	**14**	**14**	**2**	*14*	*14*	*3*	*Fill_level*; *Idle_level*; *Warn_level*; *TH1*; *PB1;TH2; PB2*.

1) Industrial examples

Our testability analysis process has been experimented with two systems LM1 and LM2 provided by AIRBUS. These examples are extracted from an AIRBUS program Auto Flight systems. LM1 and LM2 contain respectively 69 and 146 nodes.

The following table "Tab. III" exposes the result of the defined testability analysis methodology application on these systems.

We observe that the number of selected testability flows decreases from (1) to the (3). The "Section IV" describing the different testability approaches gives explications about the mechanism. Indeed, the description of some output variables can share SCADE model parts.

In (1), these common parts are analyzed for each output variable. Thereby, an important number of testability flows is determined using this approach.

Regarding (2), common parts using by a set of output variables related to a function are analyzed once during testability analysis. Common parts related to different functions are analyzed several times. So, the number of testability flow decreases compared to (1) but is still high.

In (3), all SCADE model related parts are analyzed together. Each common part is explored only once during testability analysis. As a result, testability flows determined by this approach represent the lowest number compared to the two others.

Considering these approaches, the principle of (3) sticks to the AIRBUS testing process of system validation.

The experiment result analysis of LM1 and LM2 outlines the following points:

- No orphan testability flows has been detected after running our testability analysis process. Neither missing flow is identified in the detailed specification.
- Considering (3), for LM1 (resp. LM2), the number of tests (70 (resp. 114)) is lower than the number of flows (84 (resp. 127)). At least, 14 (resp. 13) tests are missing.
- During tests coverage analysis process, we identified that several tests are linked to the same testability flows. Consequently, they can be considered redundant.

These experiments show the testability analysis methodology described in Section IV can support efficiently the AIRBUS system detailed specification validation process. Nevertheless, the applicability of the method in term of scalability must be confirmed on larger operational systems.

B. Testability measures assessment

This assessment aims at checking testability measures relevance in the AIRBUS systems validation context. Indeed, these measures should give information about the complexity introduced by some operators during test definition. The main idea is the highlighting of a possible correlation between predictive controllability and observability measurements and the test effort during the validation of the system specification. The results show that it is difficult to build this relation in AIRBUS context based on functional test. We use the diagram of operators described below Fig. 19 to illustrate this difficulty. It is extracted from the system LM1. This diagram is composed of temporal operators (PULSE1, PULSE2 and PREV) and logic operators (AND and NOT). PULSE1 (resp. PULSE2) outputs pulses on the rising (resp. falling) steps of a Boolean signal. PREV delays a Boolean signal. I_1, I_2, I_3, I_4 are the inputs of the diagram and O_1, O_2 represent the outputs the diagram of operators.

The "Tab. IV" exposes the testability measures associated with each operator used in the diagram above.

1) Controllability measurement

The "Tab. IV" highlights a difference of about 13% between the controllability measure associated with AND_1 (1.0) and AND_2 (0.8754). This difference is due to the lost of information caused by AND_1 on AND_2 input. This reduces the effective information quantity available on AND_2. Therefore, the controllability measurement of AND_2 becomes lower than the AND_1.

TABLE III. LM1 AND LM2 TESTABILITY ANALYSIS RESULT

	Number								
	Functional requirements	Output variables	Defined tests	Selected nominal flows			Selected initialization flows		
				(1)	(2)	(3)	(1)	(2)	(3)
LM1	34	58	70	**615**	**198**	**84**	*155*	*83*	*73*
LM2	57	92	114	**534**	**202**	**127**	*203*	*132*	*112*

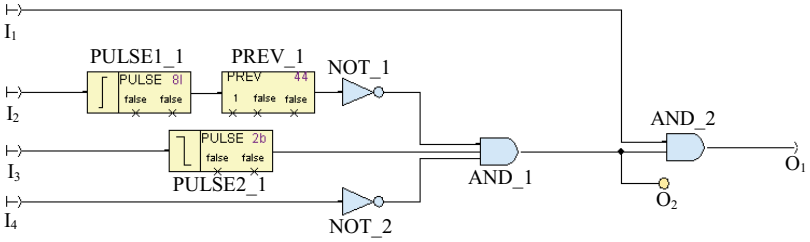


Figure 19. Diagram of operators extracted from the system LM1

TABLE IV. TESTABILITY MEASURES

	PULSE1_1	PREV_1	PULSE2_1	NOT_1	NOT_2	AND_1	AND_2
Controllability	1.0	0.9539	1.0	0.9443	1.0	**1.0**	**0.8754**
Observability	0.9273	0.9271	0.9177	**0.9271**	**0.8754**	1.0	1.0

In a functional test context, the activation effort of AND_2 is at the most equal to the needed effort to activate AND_1. The coverage effort of AND_2 is limited to the production of "true" and "false" by AND_1. Indeed, the second input of AND_2 corresponds to a system input (I_1). Considering this example, we can conclude that it is difficult to construe the controllability measurement (defined as such) in terms of the effort of an operator or a component.

2) Observability measurement

The observability measures associated to operators presented on "Tab. IV" show a difference of about 5% between NOT_1 (0.9271) and NOT_2 (0.8754). Indeed, the effective information quantity available on the output NOT_1 is lower than NOT_2 one. Then, the information loss from NOT_1 is less important than NOT_2 from the both operators to the outputs (O_1 and O_2).

In a functional test context, the observation effort of the output of NOT_1 is equal to NOT_2 one. Indeed, the information flows produced by the both operators are treated in the same way to the outputs (O_1 and O_2). Considering this case, we can conclude that it is difficult to interpret the observability measurement (defined as such) in terms of the effort of an operator or a component.

VII. CONCLUSION AND FUTURE WORK

This paper deals with the definition of a methodology based on testability principles to support traceability activities and test design in the system validation process.

We propose methods highlighting links between the detailed specification, the SCADE model, the test data and the testability flows. AIRBUS systems testability analysis required the extension of SATAN (System's Automatic Testability ANalysis) technology principles. This extension concerns the definition of Airbus specific operators modeling techniques.

We also introduce the temporal aspect in the testability analysis process by proposing the information loss assessment technique considering several execution cycles of the system. A testability flows classification technique has been defined. It splits testability flows in two categories: the initial flows related to system initialization phase; and the nominal flows. Such an approach allows facilitating coverage analysis activities and tests design. The validation phase is thus shortened generating important cost and effort reduction. However, we demonstrate that testability measures, such as defined, do not provide relevant information about the complexity introduced by some system parts during test definition in the functional testing context.

In the future, our work will focus on two main objectives. The first one consists in leading new experiments on a larger number of systems in order to enhance tools supporting the systems specification coverage analysis methodologies. This allows their consolidation for their deployment in operational conditions.

The second objective of our future work aims at adapting testability measures to a functional testing context. This requires the consideration of value and constraints related to information flows in the system specification during information loss coefficient assessment of operators.

REFERENCES

[1] F. Doumbia, O. Laurent, C. Robach, and M. Delaunay, "Using the Testability Analysis Methodology for the Validation of AIRBUS Systems," VALID 2009, First International Conference on Advances in System Testing and Validation Lifecycle, pp.86-91, September 2009.

[2] H. V. Do, M. Delaunay, and C. Robach, "Integrating testability into the development process of reactive systems", *IASTED SE 2007*, Innsbruck, Austria, February 2007.

[3] C. Robach: "Test et testabilité de systèmes informatique", *PhD Thesis*, 1979.

[4] N.Halbwachs, P. Caspi, P. Raymond, and D. Pilaud: "The synchronous dataflow programing language LUSTRE", *Proceedings of the IEEE*, 79(9): 1305-1320, September 1991.

[5] Esterel Technologies SA. *SCADE Technical Manual*, 2005.

[6] R. S. Freedman. Testability of Software Components. *IEEE Transactions on Software Engineering*, 17(6):553–564, Jun 1991.

[7] Y. Le Traon and C. Robach. Testability Measurements for Data Flow Design. *In Proceedings of the Fourth International Software Metrics Symposium*, pages 91–98, Albuquerque, New Mexico, Nov 1997.

[8] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[9] B. A. Nejmeh. Npath: A complexity measure of execution path complexity and its applications. *Communications of the ACM*, 31(2):188–200, February 1988.

[10] C. Robach and P. Wodey. Linking design and test tools: an implementation. *IEEE Transactions on Industrial Electronics*, 36:286–295, 1989.

[11] J. M. Voas and K. W. Miller. Software testability: The new verification. *IEEE Software*, 12(3):17–28, May 1995.

[12] H. V. Do, C. Robach, M. Delaunay, and J. S. Cruz. Testability Analysis for Grapically Describec Algorithms of Reactive Systems. *EMBEDDED REAL TIME SOFTWARE (ERTS) 2006*, Toulouse, France, January 2006.

[13] A. Dammak: "Etude de Mesures de Testabilité de Systèmes Logiques", PhD Thesis, 1985.

[14] A. Benveniste and G. Berry. The Synchronous Approach to Reactive and Real-Time Systems. *Proceedings of the IEEE*, 79(9), 1991.

[15] C. Robach, H. V. Do, and M. Delaunay. Testability as a component of CASE tools. *International Conference on Degradation, Damage, Fatigue and Accelerated Life Models in Reliability Testing*, Angers, France, May 2006.

[16] Bermoulli Distribution. *http://en.wikipedia.org/wiki/Bernoulli_distribution*.

[17] RTCA/DO-178B, "*Software Considerations in Airborne Systems and Equipment Certification*", December 1992.

Towards a Deterministic Business Process Modelling Method based on Normalized Systems Theory

Dieter Van Nuffel, Herwig Mannaert, Carlos De Backer, Jan Verelst
Department of Management Information Systems
University of Antwerp
Antwerp, Belgium
dieter.vannuffel;herwig.mannaert;carlos.debacker;jan.verelst@ua.ac.be

***Abstract*—Normalized Systems theory has recently been proposed to engineer evolvable information systems. In order to build information systems according to this theory, a method to identify the Normalized Systems' primitives has to be constructed. Because business processes are currently receiving more attention as process-centric representations of an enterprise, the method should be able to translate business process models into the Normalized Systems primitives. In this paper, a preliminary mapping method based on proven software engineering principles, is discussed. The proposed method adheres to the Normalized Systems' viewpoint of business processes being normalized production lines. In this sense, business process production lines are identified as workflow elements operating on a single type of data element. These process lines are operated as state machines, triggering action elements on the specified data element. The mapping method is illustrated using an example of a realistic business process flow. Preliminary guidelines and conclusions on the method construction are presented.**

***Keywords*—Normalized Systems, Business Process Engineering, Business Process Modelling, BPMN**

I. Introduction

Contemporary information systems are confronted with higher demands of evolvability, i.e. able to be swiftly adapted to the changing business environment. The required business agility needs to be translated towards the supporting software, which makes software change inevitable. However, due to the invasiveness and frequency of these changes and because most IT infrastructures are poorly architected, organizations severely suffer from the number and nature of their complications [26]. Most of the time, these adaptations happen during the mature life cycle stage of an information systems and are thus coined as software maintenance [25]. Software maintenance is therefore regarded as the most expensive phase of the software life cycle, and often leads to an increase of architectural complexity and a decrease of software quality [9]. This phenomenon is known as Lehman's law of increasing complexity [14], expressing the degradation of information systems' structure over time. To accomplish the required agility within information systems, the Normalized Systems theory has recently been established [16]. Based on the systems theoretic concept of stability, a software engineering theory is proposed to engineer evolvable information systems. Although the theory has already been used to design global mission-critical information systems [15], a systematic way to derive primitives underlying Normalized Systems from organizational requirements is not yet completely determined. For this purpose, different approaches to describe organizational requirements are available, but business processes are recently receiving more attention as process-centric representations of an enterprise. Whereas earlier, mostly data-driven approaches have been pursued as a starting point for information systems modelling, there is currently a tendency to apply process-driven requirements engineering [24].

A relatively large number of notations, languages

and tools exist to model business processes. These existing business process languages however have some limitations, e.g., absence of formal semantics, limited potential for verification, message-oriented approach and multi-party collaborations modelling [4]. Nevertheless, they are currently adopted within numerous organizations, and especially the Business Process Modelling Notation (BPMN) is one of the most applied notations [18]. Although the constructs of BPMN are rather ambiguously defined, the notation seems to be quite intuitive, and easy to understand and learn [7]. It is even argued that BPMN has become the de facto process modelling standard, being more widely adopted and supported than other business process modelling languages such as Event-Driven Process Chains (EPC) [21]. Therefore, BPMN models are chosen to represent organizational requirements in our research. The contribution of this paper is thus aimed at mapping, in a systematic way, the organizational requirements represented as BPMN models, to the primitives of Normalized Systems exhibiting proven evolvability. In this sense, the paper provides a way to derive stable information systems from contemporarily widely applied process-centric requirements representations, offering a potential answer to the problems earlier stated. This paper extends the method presented in [1] by adding a number of theory-grounded guidelines and illustrating their applicability on an expanded case study.

The remainder of the paper is organized as follows. In Section 2, the Normalized Systems theory will be discussed. In addition, it will elaborate on the different types of the Normalized Systems primitives, and how these primitives can enable business processes. A third section provides insights on a systematic way to map business processes onto these primitives of Normalized Systems. Finally, conclusions and future research are discussed.

II. Normalized Information Systems

Manny Lehman's law of increasing complexity [13], [14] expresses the degradation of information systems' structure over time. Normalized Systems Theory has been proposed to design and implement information systems that defy this law. In a first section, a brief summary of this theory is presented. A second section explains the implications of Normalized Systems Theory on the automation of business processes.

A. From Stability to Evolvable Elements

In this section, we present a brief overview of Normalized Systems theory. Starting from the systems theoretic concept of stability, both software design theorems and evolvable software elements are deduced.

1) Stability and Combinatorial Effects: The basic assumption of Normalized Systems theory is that information systems should be able to evolve over time, and should be designed to accommodate change. Therefore, the software architecture should not only satisfy the current requirements, but should also support future requirements. Although this is an important concern for all information systems, it is particularly important for large-scale information systems and even more important for Software Product Lines, as future applications are sometimes hard to predict [15], [17].

In order to support these changes, Normalized Systems Theory states that an essential characteristic of an information system is its *stability*. In systems theory, stability refers to a system in which a bounded input function results in bounded output values, even as $t \rightarrow \infty$. When applied to information systems, this implies that there should be no combinatorial or change propagation effects in the system. This means that applying a specific change to the information system should require the same effort, irrespective of the size of the information system or the point in time at which the change is applied. This implies that such systems defy Manny Lehman's Law of Increasing Complexity, which states that as time goes by, the structure of software will degrade and become more complex as changes are applied to it, causing the impact of a given change to increase over time [13], [14].

Normalized Systems are defined as information systems exhibiting stability with respect to a defined set of changes [15]. In this sense, evolvability is operationalised as a number of anticipated changes that occur to software systems during their life cycle

[16]. The existence of changes that are dependent on the size of the system, pose a serious threat to stability, and are called *combinatorial effects* [15], [16].

2) Design Theorems for Software Stability: To contain these combinatorial effects, a sound architectural approach is required, following a set of design rules as called for by Baldwin and Clark [2]. In Normalized Systems Theory, a set of four *design theorems* is deduced that act as design rules to identify most combinatorial effects [15], [16]. Essentially, these theorems identify, in very clear and specific terms, places in the software architecture where high coupling is threatening evolvability.

The first theorem, *separation of concerns*, implies that every change driver or concern should be separated from other concerns. This theorem allows for the isolation of the impact of each change driver. This principle was informally described by Parnas already in 1972 [19] as what was later called *design for change*. This theorem implies that each module can contain only one submodular task (which is defined as a change driver), but also that workflows should be separated from functional submodular tasks. Any violation automatically results in a combinatorial effect: for instance, consider a function F consisting of task A with a single version and a second task B with N versions; thus leading to N versions of function F. The introduction of a mandatory version upgrade of the task A will not only require the creation of the additional task version of A, but also the insertion of this new version in the N existing versions of function F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect.

The second theorem, *data version transparency*, implies that data should be communicated in version transparent ways between components. This requires that this data can be changed (e.g., additional data can be sent between components), without having an impact on the components and their interfaces. For instance, consider a data structure D passed through N versions of a function F. If an update of the data structure is not version transparent, it will also demand the adaptation of the code that accesses this data structure. Therefore, it will require new versions of the N existing processing functions F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. This principle can, for example, be accomplished by appropriate and systematic use of web services instead of using binary transfer of parameters. This also implies that most external APIs cannot be used directly, since they use an enumeration of primitive data types in their interface. As a result, such interface is not data version transparent.

The third theorem, *action version transparency*, implies that a component can be upgraded without impacting the calling components. Consider, for instance, a processing function P that is called by N other processing functions F. If a version upgrade of the processing function P is not version transparent, it will, besides upgrading P, also cause the adaptation of the code that calls P in the various functions F. Therefore, it will require new versions of the N existing processing functions F. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. This principle can be accomplished by appropriate and systematic use of, for example, polymorphism or a facade pattern. In practice, it can often be observed that upgrading a component can have an impact on the rest of the system. A possible reason could be that they are not used in an action version transparent way.

The fourth theorem, *separation of states*, implies that actions or steps in a workflow should be separated from each other in time by keeping state after every action or step. For instance, consider a processing function P that is called by N other processing functions F. Suppose the calling of the function P does not exhibit state keeping. The introduction of a new version of P, possibly with a new error state, would force the N functions F to handle this error, and would therefore lead to N distinct code changes. The number N is clearly dependent on the size of the system, and thus implies a combinatorial effect. This theorem suggests an asynchronous and stateful way of calling other components. Synchronous calls—resulting in pipelines of objects calling other objects, which are

typical for object-oriented development—result in combinatorial effects.

It needs to be emphasized that each of these theorems is not completely new, and even relates to the heuristic knowledge of developers. However, formulating this knowledge as theorems that cause combinatorial effects, supports systematic identification of these combinatorial effects so that systems can be built with minimal combinatorial effects.

3) Encapsulations for Evolvable Elements: The design theorems show that software constructs, such as functions and classes, by themselves offer no mechanisms to accommodate anticipated changes in a stable manner. Therefore, Normalized Systems Theory proposes to encapsulate software constructs in a set of five higher-level software elements, modular structures that adhere to the design theorems, in order to provide the required stability with respect to the anticipated changes [15].

The second and third theorem imply that the basic software constructs, representing data and actions, need to be encapsulated in order to build stable information systems. This leads to the following encapsulations or elements:

- *Data Encapsulation*, the composition of software constructs to encapsulate a data construct into a data element, implies that data elements have get- and set-methods for data version transparency, So-called cross-cutting concerns—such as remote access and persistence— can be added to the element in separate constructs.
- *Action Encapsulation*, the composition of software constructs to encapsulate an action construct into an action element, implies that the core action construct can only contain a single functional task, not multiple tasks, and that workflow has to be separated from these elements. Arguments and parameters need to be encapsulated as data elements, and so-called cross-cutting concerns—such as remote access, logging and access control— can be added to the action element in separate constructs.

The first and fourth theorem, dealing with aggregations of tasks, imply that workflow must be separated from other action elements, actions must be separated or isolated by intermediate states, and information systems must be able to follow up and react on states and/or error states. This leads to additional encapsulations:

- *Workflow Encapsulation*, the composition of software constructs to create an encapsulated workflow element, implies that workflow elements cannot contain other functional tasks, and that they must be stateful. This state is required for every instance of use of the action element, and therefore needs to be part of, or linked to, the instance of the data element that serves as argument.
- *Trigger Encapsulation*, the composition of software constructs to create an encapsulated trigger element, implies that trigger elements need to control the separated—both error and non-error—states, and check whether an action element has to be triggered. So-called cross-cutting concerns—such as controlling the trigger and its time interval—can be added to the element in separate constructs.
- *Connector Encapsulation*, the composition of software constructs to create an encapsulated connector element, implies that connector elements must ensure that external systems can interact with data elements, but that they cannot call an action element in a stateless way. So-called cross-cutting concerns—such as setting up network listeners—can be added to the element in separate constructs.

B. Business Process Production Lines

In this section, the viewpoint of the Normalized Systems theory on business processes will be first discussed. The subsequent subsections will describe the different elements of Normalized Systems relevant to enable business processes.

1) Normalized Production Lines: Automated manufacturing is based on so-called production or assembly lines, where products are assembled as they pass through the production line. At every step or position of the assembly line, a specific and dedicated operation is performed on the product that is being created. Though production lines seem highly integrated at first sight, they actually exhibit

loose coupling. Though every single processing step requires the completion of the previous steps on that instance of the product that is being created, it neither requires any knowledge of the previous processing steps, nor of the subsequent steps. Moreover, they do not have to be aware of the timing of the other steps. Any step can be performed on thousands of product instances that have been prepared hours, or even days, earlier.

It is this proven model or metaphor for automated production in the industrial world that we propose to apply to the automated execution of business processes by information systems. We translate the concept of a production line, that assembles instances of a specific product that is being created, to a business process flow, that performs operations on instances of a specific target data argument. The software primitives of these production lines are the *elements* of Normalized Systems theory. These elements are encapsulated software entities that exhibit stability with respect to a defined set of basic changes, and that are able to take care of a number of so-called cross-cutting concerns, such as persistency and remote access. Software entities are defined as instantiations of programming constructs, for instance `Java` or `C#` classes.

2) Data and Action Elements: Based on the laws of *separation of concerns* and *separation of states*, we propose that every flow is concerned with one, and precisely one, type of data element. Due to *separation of states*, every flow should be divided in its constituent actions to isolate the different functional tasks and to sequence the state transitions. As such, the artefact whose state is being altered by the subsequent functional tasks should be uniquely defined. Complementing this insight with the metaphor of the normalized production line, it is clear that the artefact underlying a flow, is a data element. Based on *separation of concerns*, this artefact cannot represent more than one concern; thus a flow is concerned with one, and precisely one data element. This type of data element is called the life cycle object of the flow, and corresponds to the type of product that is created on an assembly line. Every instance of this data element goes through the life cycle of this flow, and a dedicated state attribute stores the state of this *product* instance. In this way, the state of a product instance is available to the outside world, resulting in the required loose coupling between operations, both in features and time. These data elements or life cycle objects basically correspond to the nouns of the business processes. Indeed, as every data element is built around a single data entity, and cross-cutting concerns such as persistency and remote access are integrated into the element, this long time promise of object-oriented software can finally be realized. In object-oriented software, nouns could only be implemented in plain classes if all cross-cutting concerns would be part of the same class.

We propose that every operation in a flow consists of one, and precisely one, action element. Once again, this is made possible by the fact that every action element is built around a single action entity and therefore task, and that cross-cutting concerns like remote control, logging, and possible access control are integrated into the element. In our opinion, the following types of action elements are distinguished in business process flows:

- *Standard actions:* the information system performs an actual action, e.g., sending an e-mail, checking the availability of a part, deciding on a type of procedure, sending an invoice, confirming an order, etcetera.
- *Bridge actions:* the information system creates another type of life cycle data element that will be processed in its own state machine flow; e.g. creating an order upon an approved offer, creating a number of parts to be reserved upon an accepted order, creating an invoice after an order has been delivered, etcetera.
- *Manual actions:* a human user is required to perform the action, and to set the state of the life cycle data element through a user interface, e.g. approving an expense report, granting a holiday, checking whether a payment has been made, etcetera.
- *External actions:* another process, possibly belonging to another information system, is assumed to perform the action, and to set the state of the life cycle data element, e.g. reporting the state of another system, trigger-

ing an alarm, acknowledging the receipt of a transmission, etcetera.

3) Flows, Tasks, and Timers: As mentioned in paragraph II-A, a workflow element is responsible for executing the process flow for every instance of the target life cycle data element. Because it should be possible to (re-)define and to (re-)configure workflows in a dynamic way, the definition of the workflow should not be programmed or hard-coded. According to the theorem of *separation of concerns*, a particular workflow description language like BPEL should not drive the workflow as this combines the process flow with a specific technology. Workflows should therefore be defined using a neutral representation consisting of data elements. To apply descriptions like BPEL, a connector element should be used. As such, the concern of a specific technology and the concern of workflow execution are separated. Thus, in order to specify a process flow according to the Normalized Systems theory, the following data elements are defined:

- *Flows:* An instance of a `Flow` data element represents a process flow operating on a single life cycle data element, e.g. an `InvoiceFlow`, an `OrderFlow`, etcetera. Such a flow consists of multiple `tasks` on the target data element. It is possible to have different flows operating on the same life cycle data element, e.g. multiple invoice flows.
- *Tasks:* An instance of a `Task` data element represents a task operating on a single life cycle data element. Such a task identifies a specific action element operating on the data element, a parameter data element, a begin or trigger state, a success state, and a failure state. This failure state allows the flow to branch from the so-called golden path for specific instances of the data element. Tasks are grouped using flows.
- *Timers:* An instance of a `Timer` data element represents a timing constraint operating on a single life cycle data element. Such a timer specifies a maximum allowed period between two states or anchor points in a flow. The timer may identify a specific action element to be executed in case the timer expires, and/or a new state that needs to be set in any instance of the data element for which the timer expires.

The need for the first two elements can be straightforwardly derived, the third element is introduced because of its omnipresence in contemporary business processes. The control flow model of such a process flow is based on the following three primitives:

- *Trigger states:* Every instance of the target or argument data element needs to have a state field or attribute. This persistent field will always keep track of the current position of that data element instance in the control flow. It represents which operations have already been performed on that instance of the data element that is being processed.
- *State transitions:* A processing step or operation on an instance of a data element is specified as a state transition. Performing an operation is represented in the control flow model as a transition from one state—a value of the described state field—to another. In order to allow *branching*, such a state transition can in general have two outcomes.
- *Transition actions:* A state transition corresponds to the execution of an actual operation on that instance of the data element. Such transition action performs a real operation, and is implemented in a so-called action element.

III. Mapping Method Construction

The construction of the mapping method will be illustrated using a generic business process example of a make to order producing company. Figure 1 represents the process description modelled in BPMN. It should be mentioned that the business process example has a rather restricted scope, and can thus only be considered as a proof-of-concept to exemplify the applicability of the proposed mapping method.

> A small company manufactures customized bicycles. When an order for a customized bicycle is received by the organization, it is first evaluated by the sales department. If the order is rejected, the customer is notified and the process

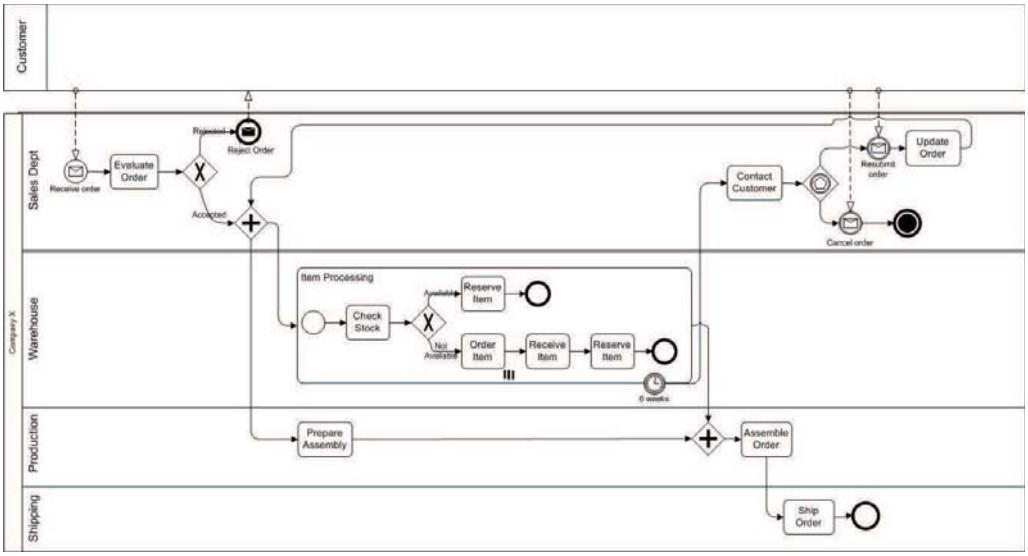


Figure 1. BPMN business process model

ends. If it is accepted, both the warehouse and production are informed. Production is commanded to plan and prepare the assembly. The warehouse processes the required items by checking the availability of the parts. If the items are available in-house, they are reserved; otherwise the items are back-ordered. Upon arrival of the ordered items, they will be received and reserved. When all required items are available and the assembly is prepared, the order is produced. After production, the order is passed to the shipping department that will ship the bicycle to the customer. When the order has not been completed within six weeks, the customer will be contacted. This option is incorporated to offer the customer the opportunity to cancel or change and resubmit the order.

In a first subsection, it will be discussed how data elements will be derived from the model. In a second subsection, the identification of flows, constituent tasks and action elements will be described. A third subsection will provide a summary on the proposed way of working, and thus constitutes a first draft of the resulting method.

A. Data Elements

Based upon the theorems of *separation of concerns* and *data version transparency*, business entities and business actors are analyzed if they correspond to separate data elements. For every noun represented in the BPMN model, expressing an business entity or actor, it should be decided whether it indicates an entity, an instance of an existing entity, or an attribute describing an existing entity. In this way, the identification is rather similar as searching for business objects within object-orientation. The choice is however not irreversible as introducing a new separate data element and an additional flow regarding this data element, is a functional change that can be translated in a set of anticipated changes for which Normalized Systems exhibit proven evolvability [15]. Determining data elements representing the life cycle objects on which the process flow is executed, can however be considered to be more concise. As business processes symbolize a sequence of activities on one or more business entities, the life cycle entities are rec-

ognized as the elementary artefacts, i.e. *concerns*, passing through the different states. Because of the relatedness to the information-centric business process modelling approach, research results from that domain can, to some extent, be incorporated in our work. As such, three conditions to determine whether an information entity is a business entity were identified [3, p.290]:

- business entities are records storing information pertinent to a given business context;
- business entities have their own, distinct life cycle from creation to completion;
- business entities have a unique identifier within the organization.

Applying our insights complemented with the aforementioned guidelines leads to the identification of the following data elements. A first data element is of course the `User`: all actors in the description need to be defined as users of the information system. In general, this data element is already in existence for other business processes that have previously been automated, as a user symbolizes a generic concern in all information systems. Also the criteria mentioned by the related research apply, as a `User` has a distinct life cycle and a unique identifier. This also accounts for a `Customer` data element, probably linked to the `User` element, as a Customer has its distinct life cycle and unique identifier within the organization. It should be mentioned that, according to Normalized Systems theory, the introduction of one or more additional data attributes to an existing data element can be done with a limited impact [16].

There are two main life cycle data elements in the described business process or flow: the `Order` and the `Part`. Although the business process in Figure 1 is described as a unified process of Order and Part, the actions on these entities cannot be represented by the same flow because they can clearly evolve independently from each other, and are thus different concerns. Moreover, the two elements have obviously independent life cycles. An instance of the `Order` data element is created by the customer and goes through the various processing steps that have been described, ending with the shipment or delivery of the order. For every part that makes up the specific instance of a product, an instance of the `Part` data element is created, and linked to the instance of the respective `Order` data element. Every individual instance of the `Part` data element goes through its own life cycle of reservation, reception, and so on. The observations that different information is stored and that different identifiers are used to pinpoint an `Order` and a `Part`, add to the rationale to identify these data elements.

Several tasks that are described in the process involve some kind of notification. Such a notification clearly consists of two concerns: the extraction of the information that makes up the message's content on the one hand, and the actual sending of the message on the other. This means that, in accordance with *separation of concerns* and *separation of states*, they have to be separated into two different tasks or action elements. The second task is actually quite a generic one: the sending of a notification message, such as an e-mail or SMS. Therefore, it should operate on a corresponding generic target data element `Notifier`, in a corresponding separate flow. This also implies that the first task of such a composed notification task will be implemented as a *bridge action*. Based on the appropriate information extracted from the order state, an instance of the `Notifier` data element will be created.

B. Flows, Tasks and Action Elements

This paragraph discusses how flows, tasks and the diverse kinds of action elements will be detected. A number of recommendations and guidelines with their rationale will be provided, each of them illustrated by the representative examples within the scope of our case example. It should be repeated that based upon the four Normalized Systems' theorems, workflow elements are represented by state transition diagrams of a single data element; and action elements will contain only one functional task resulting in a state transition of the life cycle data element driving the flow. The resulting state transition diagrams of the elementary life cycle data elements `Part` and `Order` can be found in Figures 2 and 3 respectively.

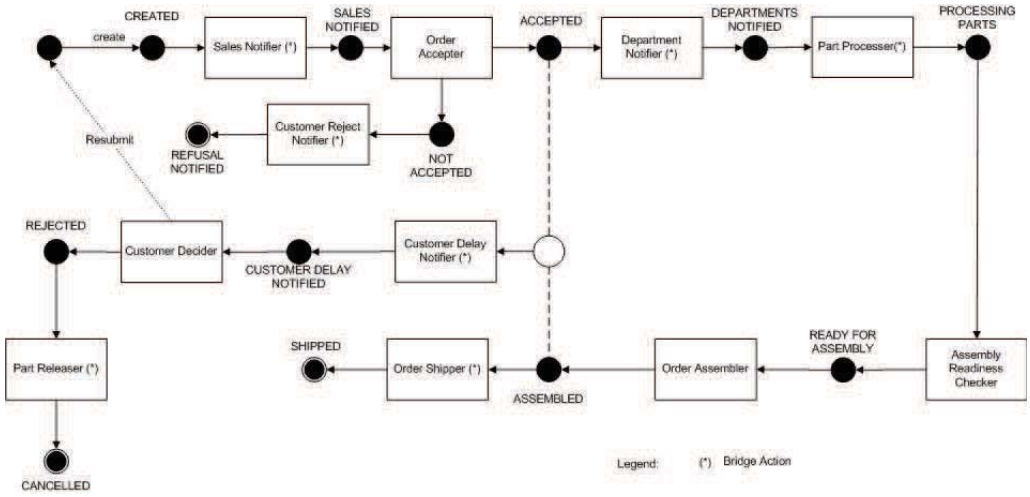


Figure 3. Schematic representation of the Order state transition diagram.

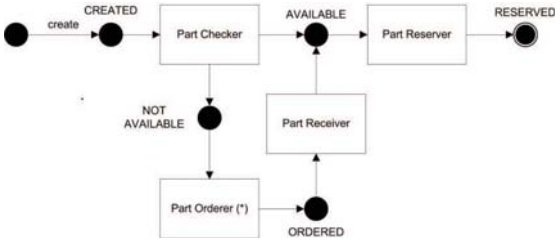


Figure 2. Schematic representation of the Part state transition diagram.

1) State labelling: The figures illustrate that due to *separation of states*, the different states must be explicitly defined. Moreover, this state definition has to be very concise because every defined state has to be unique for the life cycle object driving the flow. Otherwise, the action element triggered by the respective state can not be determined as it is not clear which state the life cycle object has. For example, an order can be rejected at multiple points in the flow, either by the sales department when receiving the order, or by the customer when the order takes longer than six weeks to complete. If these rejection states would be both labelled `rejected`, it would not be possible to distinguish between the two different notifications that have to be sent. A best practice is therefore to provide each state with a distinctive, self-explanatory label.

2) Interacting life cycle data elements: Certain life cycle data elements will be created as a result or as a consequence of actions performed by other life cycle data elements. Based on the Normalized Systems theorems *separation of concerns* and *separation of states*, these two life cycle data elements are not the same concern and both life cycles should be separately managed. Within paragraph II-B2, a bridge action was mentioned as one of the types of action elements. This action element will be used when a second life cycle data element has to be initiated. Following examples will exemplify the way of working.

Concerning the `Part Processor`, it was mentioned in Section III-A that the `Part` data element is identified as a separate life cycle element, and therefore `Part Processor` is a bridge action because at this point in the process, an instance of the `Part` life cycle data element is created for every single part of the order. Handling the *Item Processing* subprocess exemplified in Figure 1 is clearly another concern than handling the complete order. It should also be mentioned that abstraction is made of the *Prepare Assembly* activity mentioned in Figure 1. This ac-

tivity probably consists of planning the production, reserving the needed resources, etcetera. In this sense, it is argued that this activity is most likely a bridge action to an additional workflow element, for instance a `ProductionPlanningFlow`. The final step of the `Order` flow consists of shipping the order to the customer: a bridge action will create a shipment data element that will go through the shipping process not further explained in this paper.

The `Part Orderer` within Figure 2 is a bridge action to another flow or information system, as purchase orders will probably be handled by a `PurchaseFlow`. Based on the Normalized Systems theorems *separation of concerns* and *separation of states*, the issue of purchase orders being delivered on time, with the correct amount, etcetera. is not an issue of the `PartFlow`, but of a `PurchaseFlow`.

3) Notifying stakeholders: Like already mentioned at the end of Section III-A, business processes often require activities to notify certain stakeholders. This requirement is actually a particular case of interacting life cycle data elements, as one is always an instance of the `Notifier` data element. Based upon the theorem *separation of concerns*, sending notifications to diverse stakeholders is considered a separate concern. Delivering a message in the correct format to the intended recipients at the right time, with the related fault handling, does not concern other data elements. Our solution consists of implementing a bridge action that will trigger the creation of a `Notifier` data element. Of course these bridge actions will differ from each other, as the semantics of the message that has to be communicated, vary depending on the situation. Therefore, the bridge action will pass a set of parameters defining the message's content and format. Additional tasks that should be performed to send the notifications, can be designed using workflow elements defined upon the `Notifier` data element.

In the example, notifiers can be found at diverse points within the Order workflow represented by Figure 3. First, when receiving an order, a notification is sent to a sales representative in order to evaluate the order. The `Sales Notifier` bridge action will thus result in the creation of a `Notifier` data element that sends an e-mail to a sales representative stating that a particular order has to be evaluated. Second, the `Department Notifier` bridge action creates a notification or assembly preparation request that is sent to the manufacturing department, and the individual parts are created. Third, when the six weeks timer elapses, a `Delay Notifier` bridge action is triggered that will create a notification sent to the customer to inform her about the delay and to request the wanted action. The multiple notifiers illustrate the usefulness of isolating a change driver in its designated data element to obtain true reusability as notification functionality can thus be reused by applying a bridge action.

4) Communicating life cycle data elements: Different life cycle data elements sometimes need to communicate their state to one another in order to trigger further execution of the flow. Although in many cases an external action can implement this, like the `Order Assembler` action in the `OrderFlow` or the `Part Receiver` action in the `PartFlow`, this will not suffice in the particular case when a life cycle data element A triggers multiple instances of another life cycle data element B, and its flow can only continue when all these instances have reached a particular state. In this case, an action element on the triggering life cycle data element A has to be implemented that will verify the state of the initiated instances of B. When all the initiated instances of B reach the target state, the action will set off the state transition on A. If one of the triggered instances of B has not yet arrived at the target state, A's state will not be altered. As such, the action element will be initiated until all instances of B attain the target state.

An example from the case will exemplify and further ground the proposed solution: when all parts are created in the `OrderFlow`, the order has to wait until it can be produced. This implies that the manufacturing department is ready to start the assembly, and that all parts are reserved and available in stock. Actually, both these conditions will become available in the instance

of the Order data element, either through a data attribute or data links. Therefore, though this information will be entered by another flow (e.g. the PartFlow) and due to the *separation of concerns* theorem not allowing one flow to actively interfere with another one; a simple standard action, Assembly Readiness Checker, is needed to check the appropriate information on a regular basis. A flow element actively interfering with another flow element can be considered a so-called *GOTO statement*. Although most contemporary business process descriptions and languages do not inhibit this behaviour, Normalized Systems theory does not allow their presence in accordance with the seminal work of Dijkstra [8]. The reason why Assembly Readiness Checker is positioned in the OrderFlow, and not within the PartFlow, is quite straightforward as the Order instance "knows" through its data links, which Parts are created. Therefore, an action in the PartFlow communicating that everything has completed correctly for a particular part can not be implemented as there are multiple parts for a single order and the individual Part instances are not aware of the existence of the other instances.

5) Human tasks: Whether a task is executed by a human or an information system does actually not matter, as it is the encapsulated functional task representing the change driver that should be isolated within its designated action element. When defining the action element within paragraph II-A, it was derived that an action element encapsulates one and only one functional task. The way in which this task is performed, manually or automated, is just a matter of implementation and should, adhering to the *action version transparency theorem*, be kept hidden from the action or flow calling the respective action element. For instance, it is obvious that Order Evaluator within Figure 3 is a manual action: the sales manager verifies the order and takes a decision whether or not to accept the order. This task is not identified as a separate element because of being performed manually, but because it represents a separate concern, namely an elementary functional task, and because the task triggers a state transition relevant to the life cycle of the underlying Order data element.

6) Timer functionality: Contemporary business processes very often contain time constraints, e.g. stakeholders only have a given period to answer, a management reporting process should be run every morning at 7 AM, etcetera. Because of its omnipresence and its clear concern, a time constraint, a timer element was introduced in paragraph II-B3 to represent this functionality. Adhering to the Normalized Systems' theorems, such a timer element can only operate on a single life cycle data element and will specify a maximum allowed period between two states of the associated data element. As such, the start state will probably trigger two elements: the action element that is described within the flow element, and the timer element. When the timer expires, either a new state is set in an instance of the data element, or a specific action element is triggered.

The description of the OrderFlow process specifies a timer of the second kind. When an order takes longer than six weeks to be completely processed, the customer has to be contacted. This individual timer element, schematically represented in Figure 3 by the open circle and described in Table I, has an allowed time window of six weeks between start state accepted and target state assembled before the Customer Delay Notifier bridge action is potentially triggered. The start state accepted thus triggers both the timer element Customer Delay Timer and the bridge action Department Notifier.

7) Cancellation Pattern: Within the business process, a customer contact was provided to offer the customer the opportunity to cancel his order. However, one should be very cautious with possible cancellations. If the customer decides to cancel the order, the state cannot simply be set to cancelled and thus disregard everything that has already been done. This would very quickly lead to an infinite amount of parts in stock as these parts will be kept reserved for an already cancelled order. If it is absolutely necessary to offer the customer the pos-

Start state	Target state	Timer Element Name	Time elapse	Action Element Triggered
accepted	assembled	Customer Delay Timer	6 weeks	Customer Delay Notifier

Table I
TIMER ELEMENT OF ORDER FLOW

sibility to cancel the order, an entire branch should be added to the process flow of the order, where the assembly request is withdrawn and the various reserved parts are released. Within our example, only the `Part Releaser` action element was modelled, because no details were provided what should be performed when the customer cancels its order. The `Part Releaser` is a standard action sending a release request to the `Part` data element, which will handle the request accordingly.

However, contemporary organizations need to provide the customer an opportunity to cancel an order during a part of or even the whole process. By consequence, cancel requests can arrive not only on specified moments like discussed above, but also during the regular flow execution. Like mentioned in paragraph III-B4, the Normalized Systems' theorems do not allow to directly interrupt a flow and to change the state. Due to transactional integrity reasons, flows can only be routed by elementary action elements resulting in state transitions of the underlying data element. To support this cancellation functionality, we suggest to add a `cancelRequest` data attribute to every data element representing a business artefact that might be have to cancelled. The following cancellation pattern describes how to handle such a cancel request:

- Capture cancel request by updating `cancelRequest` data attribute. Conditions that check whether the cancel request is valid, are designed in the update operation of this data attribute.
- The engine supporting the flow element then checks the `cancelRequest` data attribute, equivalent to the way it verifies the different states of the different data elements to trigger the correct action element.
- If the `cancelRequest` data attribute's state is true, the regular state field is updated to a value such that the regular flow does not continue; and the state field in which the data element was before cancelling, will be stored in another state attribute of the respective data element. This second state attribute will be referenced in our approach as a *parking state field*. The value attributed to the regular state attribute must be the same for every data element, as this will uniquely define the situation and can thus be recognized within all data elements; as such we suggest to label it `cancel requested`. The second state attribute (see also paragraph III-B8) is necessary to trigger the correct actions to handle the cancellation. The way a cancellation will be handled, evidently varies according to the data element's life cycle state. For instance, cancelling an accepted order will be totally different compared to cancelling an already produced order.
- Finally, an action element will be triggered that based upon the value of the parking state field will decide which cancellation flow should be triggered as the scenario will differ according to the actions already executed upon the data element that was requested to be cancelled. The output of this action element is attributing a value to the regular state field of the particular data element that uniquely describes which action should be triggered to initiate the correct cancellation flow.

If the customer decides to confirm the order, the process can simply continue, and should of course not be restarted, nor should it re-create the various parts. This latter option will be discussed in the next paragraph.

8) Pausing flows: When the customer is contacted to either cancel or resubmit the order, it can be argued that a third option is missing: the

customer still wants its initially ordered bicycle to be delivered. In this case, the order has to continue its regular flow. In the BPMN model however, this behaviour is difficult to model as neither the interrupting nor the non-interrupting intermediate events added in the latest proposed BPMN standard version 2.0 [18], exhibit the wanted behaviour of "pausing" the flow until the customer answer is received. Interrupting intermediate events break off the flow, and non-interrupting intermediate events let the flow continue.

The Normalized Systems' theorems and derived primitives do however enable this desired behaviour in a quite straightforward and easy to comprehend way: preserve the life cycle data element's state in the parking state data attribute. The way of working will be explained by applying it to the example case, in particular when the `Customer Delay Notifier` discussed in paragraph III-B6 elapses and the customer will be notified. As a result, the `Order` data element's state will become `customer delay notified`. The state in which the `Order` data element was before the timer elapsed should however not be neglected as it might be possible that the customer requests the continuation of the initial order. Therefore, this state should be made be persistent by storing it in the designated parking state data attribute. In this way, the flow will be paused and can be reinstated upon request by retrieving the state from the parking state data attribute and updating the regular state field with this value.

The fact that this requirement can be solved in a rather simple way is due to the deterministic nature of the Normalized Systems' theory. First, the *separation of concerns* theorem prescribes that atomic functional tasks should be separated in different action elements. Second, the *separation of states* theorem adds the need of defining action states in order to isolate these individual tasks. Third, combining the two theorems leads to our proposition that business processes should be implemented as state machines operating on a single data element. Fourth, adding the state labelling guideline discussed in paragraph III-B1 to these characteristics realizes that any business process state will be uniquely and unambiguously defined by the state field of the life cycle data element going through the process flow. Fifth, the deterministic pattern expansion used to design and implement Normalized Systems' elements makes it possible to introduce such an additional state field in a standard way to any instance of the respective data elements. Sixth, as such the initial life cycle data element's state can persistently be stored and retrieved upon request without interfering with the prerequisite of transactional integrity.

C. Method Overview

To summarize the results when obeying to the guidelines discussed above, Tables II and III represent the flow elements driving the business process.

In the first section, the limited potential for verification was mentioned as one of the drawbacks of contemporary business process languages. When comparing the business process represented as Normalized Systems elements in Figures 2 and 3, to the BPMN of Figure 1, it can be noticed that the former representations offer better support for verification as process states are explicitly modelled, and can thus be compared with the allowed state transitions of the underlying data element. We also claim in accordance with Kumaran et al. [12, p.41] that representing processes as state machines of life cycle data elements (or business entities) increase the understandability of these models.

It can be concluded that applying the Normalized Systems' theorems on business processes already provides some principles to assess these business processes. In this sense, the following preliminary guidelines are proposed in this article:

- Business processes should be separated in workflow elements driven by the persistent state field of a single life cycle data element.
- These life cycle data elements are identified as the elementary artefacts passing through the different states during business process execution, e.g. `Order`. Useful conditions to identify such life cycle data elements are found in the work regarding business entities [3, p.290].
- Time constraints should be isolated in separate timer elements, e.g. a six weeks timer.

Start State	End State	Action Element Name	Action Element Type
created	sales-notified	Sales Notifier	Bridge
sales-notified	accepted XOR not-accepted	Order Accepter	Manual
not-accepted	refusal-notified	Customer Reject Notifier	Bridge
accepted	departments-notified	Department Notifier	Bridge
departments-notified	processing-parts	Part Processor	Bridge
processing-parts	ready-for-assembly	Assembly Readiness Checker	Standard
ready-for-assembly	assembled	Order Assembler	External
assembled	shipped	Order Shipper	Bridge
customer-delay-notified	rejected	Customer Decider	Manual
rejected	cancelled	Part Releaser	Bridge

Table II
STATE TRANSITIONS DESCRIBING ORDER WORKFLOW ELEMENT

Start State	End State	Action Element Name	Action Element Type
created	available XOR not-available	Part Checker	Standard
not-available	ordered	Part Orderer	Bridge
ordered	available	Part Receiver	External
available	reserved	Part Reserver	Standard

Table III
STATE TRANSITIONS DESCRIBING PART FLOW ELEMENT

- When the creation of a life cycle data element is dependent on actions performed by another life cycle data element, the interaction between the two elements has to be implemented using a bridge action.
- Frequently required generic functionality like notifying people, should be isolated in separate workflow elements driven by a generic data element, e.g. `Notifier`.
- Separating the activities of a business process into different tasks and action elements can be done in a structured way by dividing different concerns, representing different change drivers, into different Normalized Systems primitives. For instance, the identification of the action element `Assembly Readiness Checker` based upon the communicating life cycle data elements guideline, exemplifies the fact of only allowing one functional task in one single action element: some designers might be tempted to implement this task either in the workflow element itself, or in the `PartsCreator` action element.
- To cancel processes, a Cancellation Pattern is proposed needing a `cancelRequest` and a `parking state field` data attribute that are by default provided in every life cycle data element.
- To enable processes to be paused, a pattern is proposed again using the by default provided `parking state field` data attribute.

IV. RELATED WORK

Our work relates to research in three areas. First, it is related to research on modularity and stability. Modularity expresses the idea to decompose a system in loosely coupled building blocks. In software engineering, modularity is used to decompose an information system in independent modules [19]. Stability refers to the systems theoretic notion that a bounded input results in bounded output. Although no precise definition exists in the context of information systems, most authors imply it to refer to software or information systems architectures designed to be resistant to change propagations [10].

Second, two business process theorems relate to

a certain extent to our viewpoint. First, the case handling paradigm also focuses on the role of data objects to drive the flow [22]. This orientation considers a case to be the central concept and describes it as a product, which is produced with structure and state. This structure and state are based on a collection of data objects representing valuable information about the case. As such, a process is defined as the recipe for handling cases of a certain type. The main differences with other workflow approaches are the focus on the whole case and not on single work-items; and the state of the case, rather than the control flow, that primarily determines which activities are enabled [23]. Second, our work is related to the information-centric approach on business process modelling where a business process is modelled as the interacting life cycles of information entities [12]. These information entities, also called business entities, are used to describe business processes operating as state machines where state transitions are caused by activities acting on the most important entity. Business processes are thus defined as the life cycles of the business entities from their initial to final state. In this sense, the approach is very closely related to ours.

Third, the mapping method presented in this paper relates to research in the Service-Oriented Architectures (SOA) domain. In this domain, a number of approaches exist that describe how to identify service operations based on business process models. These approaches originate from both practice, e.g., Mainstream SOA Methodology [5], and academia, e.g., [6]. A more comprehensive overview can be found in [11], [20]. Because our proposed method is based upon proven software engineering principles, it mainly relates to the principles-driven design approaches [20].

V. Conclusion and Future Work

When deriving Normalized Systems' primitives from business process models, the following initial conclusions can be drawn. Data elements are mostly only indirectly represented within business process models. Therefore, every noun should be systematically checked as a potential data element candidate. The identification of the elementary life cycle data elements is however considered relatively straightforward as they represent the business entities going through different business states during business process execution. Moreover business processes will potentially be enabled by multiple workflow elements as the Normalized Systems theorems propose that a workflow element should only relate to one and only one data element. In this sense, both `Order` and `Part` workflow elements were identified.

Due to the fact that business process models emphasize the flow of activities, the constituent tasks of these workflow elements can be deducted in a structured way. Also action elements are obtainable by merging the Normalized Systems' laws with the functionality exhibited by the activities within the business process model. For instance, timer elements are basic blocks of both business processes and Normalized Systems, and can therefore be mapped in a structured way. In addition, the case demonstrated how the omnipresent tasks of contacting diverse actors can be mapped to a generic `Notifier` data element on which workflows taking care of the requested notification functionality can be defined.

Our future work will be, next to executing more extended and additional case studies, targeted at formalizing the method proposed in this paper. The rather implicit rules must be translated into strict guidelines, providing an unambiguous way to derive the Normalized Systems elements from business process models. This will also include identifying the different concerns existing at the level of business processes, as they will vary from the concerns identified at the software level. Second, the mapping of other business process modelling languages and enterprise architecture descriptions to Normalized Systems primitives will be studied. Finally, research on the Normalized Systems theory itself will be extended. Key areas are the introduction of additional supporting tasks into the stable software elements, and porting the stable element patterns to supplementary software platforms.

REFERENCES

[1] D. Van Nuffel, H. Mannaert, C. De Backer and J. Verelst, "Deriving Normalized Systems elements from business process models," in *Proceedings of Fourth International Conference on Software Engineering Advances (ICSEA 2009)*, K. Boness, Ed. Los Alamitos, CA, USA: IEEE Computer Society, September 2009, pp. 27–32.

[2] C. Y. Baldwin, and K. B. Clark, "Design Rules: Vol. 1: The Power of Modularity," MIT Press, Cambridge, MA, USA, 2000.

[3] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su, "Towards formal analysis of artifact-centric business process models," in *BPM 2007*, ser. Lecture Notes in Computer Science, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Berlin Heidelberg: Springer-Verlag, 2007, pp. 288–304.

[4] M. De Backer, M. Snoeck, G. Monsieur, W. Lemahieu, and G. Dedene, "A scenario-based verification technique to assess the compatibility of collaborative business processes," *Data and Knowledge Engineering*, vol. 68, no. 6, pp. 531–551, June 2009.

[5] T. Erl, "SOA: Principles of Service Design," Prentice Hall, Upper Saddle River, NJ, USA, 2008.

[6] A. Erradi, S. Anand, and N. Kulkarni, "SOAF: An Architectural Framework for Service Definition and Realization," in *Proceedings of the IEEE International Conference on Services Computing (SCC'06)*. Los Alamitos, CA, USA: IEEE Computer Society, September 2006, pp. 151–158.

[7] R. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, November 2008.

[8] E. Dijkstra, "Go to statement considered harmful," *Communications of the ACM*, vol. 11, no. 3, pp. 147–148, 1968.

[9] S. G. Eick, T. L. Graves, A. F. Karr, J. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, January 2001.

[10] D. Kelly, "A study of design characteristics in evolving software using stability as a criterion," *IEEE Transactions on Software Engineering*, vol. 32, no. 5, pp. 315–329, May 2006.

[11] A. Kontogogos, and P. Avgeriou, "An Overview of Software Engineering Approaches to Service Oriented Architectures in Various Fields," in *Proceedings of the 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, S. M. Reddy, Eds., Los Alamitos, CA, USA: IEEE Computer Society, July 2009, pp. 254–259.

[12] S. Kumaran, R. Liu, and F. Y. Wu, "On the duality of information-centric and activity-centric models of business processes," in *20th International Conference on Advanced Information Systems Engineering, CAiSE 2008*, ser. Lecture Notes in Computer Science, Z. Bellahsene and M. Leonard, Eds., vol. 5074. Berlin Heidelberg: Springer-Verlag, June 2008, pp. 32–47.

[13] M. M. Lehman, "Programs, life cycles, and laws of software evolution," in *Proceedings of the IEEE*, Vol. 68, pp. 1060–1076, September 1980.

[14] M. M. Lehman and J. F. Ramil, "Rules and tools for software evolution planning and management," *Annals of Software Engineering*, vol. 11, no. 1, pp. 15–44, November 2001.

[15] H. Mannaert and J. Verelst, *Normalized Systems: Recreating Information Technology Based on Laws for Software Evolvability*. Hasselt: Koppa, March 2009.

[16] H. Mannaert, J. Verelst, and K. Ven, "Exploring the concept of systems theoretic stability as a starting point for a unified theory on software engineering," in *Proceedings of Third International Conference on Software Engineering Advances (ICSEA 2008)*, H. Mannaert, T. Ohta, C. Dini, and R. Pellerin, Eds. Los Alamitos, CA, USA: IEEE Computer Society, October 2008, pp. 360–366.

[17] H. Mannaert, J. Verelst, and K. Ven, "Design theorems for avoiding combinatorial effects in integrating open source software components in software product lines.," in *Proceedings of the Joint Workshop on Quality and Architectural Concerns in Open Source Software (QACOS) and Open Source Software and Product Lines (OSSPL)*, Babar, Muhammad Ali et al. Eds. Skövde, Skövde University, 2009, p. 20-27.

[18] Object Management Group, "Business Process Modeling and Notation, v2.0 Beta 1 OMG Adopted Beta specification," online available at: http://www.omg.org/cgi-bin/doc?dtc/09-08-14, August 2009.

[19] D. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," in *Communications of the ACM*, Vol. 15, Nr. 12, pp.1053–1058, 1972.

[20] S. Patig, "Cases of Software Services Design in Practice," in *ICSOFT 2009 - Proceedings of the 4th International Conference on Software and Data Technologies*, B. Shishkov, J. Cordeiro, and A. Ranchordas, Eds. Setubal, Portugal: INSTICC Press, July 2009, pp. 376–383.

[21] J. Recker, "Opportunities and constraints: the current struggle with BPMN," *Business Process Management Journal*, vol. 16, no. 1, pp. 181–201, 2010.

[22] H. A. Reijers, J. H. M. Rigter, and W. M. van der Aalst, "The case handling case," *International Journal of Cooperative Information Systems*, vol. 12, no. 3, pp. 365–391, September 2003.

[23] W. M. P. van der Aalst, M. Weske, D. Grünbauer, "Case handling: a new paradigm for business process support," *Data and Knowledge Engineering*, vol. 53, no. 2, pp. 129–162, May 2005.

[24] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, 2007.

[25] B. J. Williams and J. C. Carver, "Characterizing software architecture changes: A systematic review," *Information and Software Technology*, vol. 52, no. 1, pp. 31–51, January 2010.

[26] J. L. Zhao, M. Tanniru, and L.-J. Zhang, "Services computing as the foundation of enterprise agility: Overview of recent advances and introduction to the special issue," *Information Systems Frontiers*, vol. 9, no. 1, pp. 1–8, March 2007.

Adaptive Object-Models: a Research Roadmap

Hugo Sereno Ferreira
INESC Porto
Faculdade de Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 Porto, Portugal
hugo.sereno@fe.up.pt

Filipe Figueiredo Correia
Faculdade de Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 Porto, Portugal
filipe.correia@fe.up.pt

Ademar Aguiar
INESC Porto
Faculdade de Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 Porto, Portugal
ademar.aguiar@fe.up.pt

João Pascoal Faria
INESC Porto
Faculdade de Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 Porto, Portugal
jpf@fe.up.pt

***Abstract*—The Adaptive Object-Model (AOM) is a meta-architectural pattern of systems that expose an high-degree of runtime adaptability of their domain. Despite there being a class of software projects that would directly benefit by being built as AOMs, their usage is still very scarce. To address this topic, a wide scope of concepts surrounding to Adaptive Object-Models need to be understood, such as the role of *incompleteness* in software, and its effects on system variability and adaptability, as well as existing metamodeling and metaprogramming techniques and how do they relate to software construction. The inherent complexity, reduced literature and case-studies, lack of reusable framework components, and fundamental issues as those regarding evolution, frequently drive developers (and researchers) away from this topic. In this work, we provide an extensive review of the state-of-the-art in AOM, as well as a roadmap for empirical validation of research in this area, which underlying principles have the potential to alter the way software systems are perceived and designed.**

***Keywords*-Architectural Structures and Viewpoints, Design Patterns, Families of Programs and Frameworks.**

I. Introduction

The current demand for industrialization of software development is having a profound impact in the growth of software complexity and time-to-market [1]. Moreover, a lot of the effort in the development of software is repeatedly applied to the same tasks, despite all the effort in research for more effective reuse techniques and practices. Like in other areas of scientific research, the reaction has been to hide the inherent complexities of technological concerns by creating increasingly higher levels (and layers) of abstractions with the goal of facilitating reasoning, albeit often at the cost of widening the already existing gap between specification and implementation artifacts [2]. To make these abstractions useful beyond documentation and analytical and reasoning purposes [3], [4], higher-level models must be made executable, by systematic transformation [5] or interpretation [6] of problem-level abstractions into software implementations. The primary focus of model-driven engineering (MDE) is to find ways of automatically animating such models, often used simply to describe complex systems at multiple levels of abstraction and perspectives and therefore promoting them to first-class artifacts [2].

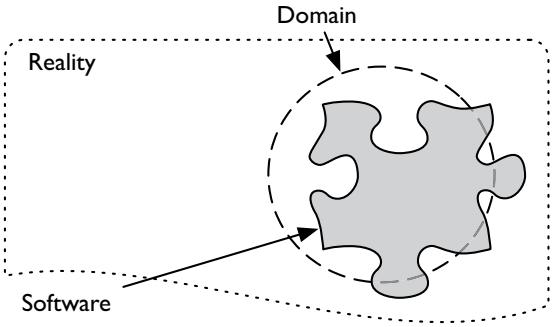


Figure 1. Software may be regarded as the crystallization of an abstraction that models a specific domain. Ideally, it would match the exact limits of that domain. But in practice: (i) those limits are fuzzy, (ii) software often imposes an artificial, unnaturally rigid structure, and (iii) reality itself keeps changing.

A. Incomplete by Design

A recurrent problem in software development is the difficulty of acquiring, inferring, capturing and formalizing requirements, particularly when designing systems where the process is highly-coupled with the stakeholders' perspective and the requirements often change faster than the implementation. This reality is well known in industrial environments, and is sometimes blamed upon incompleteness of the stakeholders' knowledge [7] — maintaining and evolving software is a knowledge intensive task that represents a significative amount of effort [8]. Consequently, once the analysis phase is finished and the implementation progresses, strong resistance to further change emerges, due to the mismatch between specification and implementation artifacts. Notwithstanding, from the stakeholder's perspective, some domains do rely on constant adaptation of their processes to an evolving reality, not to mention that new knowledge is continuously acquired, which lead to new insights of their own business and what support they expect from software.

Confronted with the above issues, some development methodologies (particularly those coined "agile") have intensified their focus on a highly iterative and incremental approach, accepting that *change* is, in fact, an invariant of software development [9]. For example, one of the most prominent books in agile methodologies — Kent Beck's "Extreme Programming Explained" [10] — use the phrase "embrace change" as its subtitle. This stance is in clear contrast with other practices that focus on *a priori*, time-consuming, rigorous design, considering continuous change as a luxurious (and somewhat dangerous) asset for the productivity and quality of software development.

Although the benefits of an up-front, correct and validated specification are undeniable — and have been particularly praised by formal methods of development, particularly when coping with critical systems — their approach is often recognized as impractical, particularly in environments characterized by continuous change. Likewise, the way developers currently cope with change often result in a BIG BALL OF MUD, where the systems will eventually face a total reconstruction, invariably impacting the economy [11]. Thus, software that is target of continuous change should be regarded as *incomplete by design*, or in other words, it needs to be constantly evolving and adapting itself to a new reality, and most attempts to freeze its requirements are probably doomed to fail (see Figure 1).

This notion, and the adequate infrastructure to support it, has roots that go as back as the history of Smalltalk and object-oriented programming in the dawn of personal computers [12]: *"Instead of at most a few thousand institutional mainframes in the world (...) and at most a few thousand users trained for each application, there would be millions of personal machines and users, mostly outside of direct institutional control. Where would the applications and training come from? Why should we expect an applications programmer to anticipate the specific needs of a particular one of the millions of potential users? An extensional system seemed to be called for in which the end-users would do most of the tailoring (and even some of the direct constructions) of their tools."*

B. Motivational Example

Figure 2 depicts small subset of the class diagram from real-world information system for a medical healthcare center. In summary, the medical center has `Patients` and specialized `Doctors`. Information about a patient, such as her personal information, `Contacts` and `Insurances`, are required to be stored. Patients go to the center to have `Appointments` with several `Doctors`, though they are normally followed by just one. During an appointment, several `Pathologies` may be identified, which are often addressed through the execution of medical `Procedures`.

In this example, we can begin to observe the *incompleteness* of these kind of information systems. For example, procedures, insurances, pathologies and contacts are depicted has having open-hierarchies (where each specialization may require different fields). Patients may not have all the relevant information recorded (e.g., critical health conditions) and foreseeing those missed formalizations, either the designer or the customer make extensive usage of an `observations` field. The system is also missing some domain notion, such that of Auxiliary Personnel, which would require a complete new entity. Maybe it will be revealed as relevant to store personal information of Doctors; actually, in the presence of this new requirements, a designer would probable make `Patients`, `Doctors` and `Auxiliary Personnel` inherits from a single abstraction (e.g., `Persons`). The healthcare center may also require the system to prevent doctors from performing procedures for which they are not qualified (e.g., through a specific constraint based on their specialization). In fact, it now seems evident that a doctor may have multiple specializations.

These are examples of requirements that could easily elude developers and stakeholders during the analysis process. What may seem a reasonable, realistic and useful system at some point, may quickly evolve beyond the original expectations, unfortunately after analysis is considered finished.

C. Accidental Complexity

Should the customer require the system to cope with these incomplete definitions, the designer would have to deliberately make the system *extensible* in appropriate points. Figure 3 shows the *refactored* elements of a particular solution that only addresses open inheritances and enumerations.

Compared to the initial design, the new one reveals itself as a much larger model. In fact, it is now more difficult to distinguish between elements that model the domain, from those that provide extensibility to the system. The result is an increase of what is defined as *accidental complexity* — complexity that arises in computer artifacts, or their development process, which is non-essential to the problem to be solved. In contract, the first model was much closer to that of *essential complexity* — inherent and unavoidable. This increase in accidental complexity was only caused by the specific approach chosen to solve the problem — in this case, recurrent usage of the TYPE-OBJECT pattern.

While sometimes accidental complexity can be due to mistakes such as ineffective planning, or low priority placed on a project, some accidental complexity always occurs as the side effect of solving any problem. For example, mechanisms to deal with out of memory errors are part of the accidental complexity of most implementations, although they occur just because one has decided to use a (von-neumann) computer to solve the problem. Because the minimization of accidental complexity is considered a good practice to any architecture, design, and implementation,

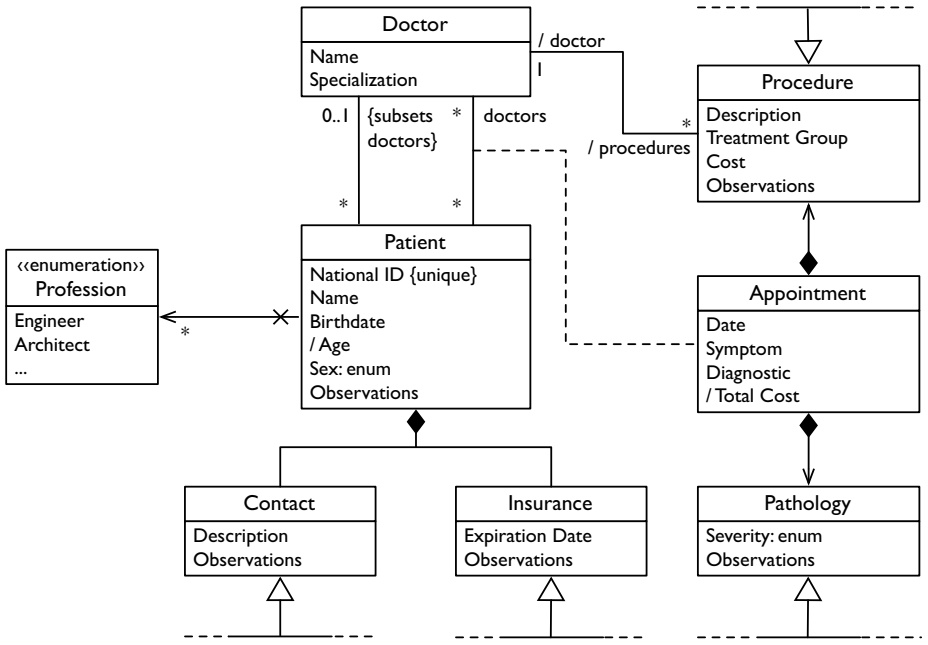


Figure 2. Example of a domain diagram of an information system for a medical center. The horizontal dashed lines denote open (incomplete) inheritances. The dots inside the enumeration also denotes incomplete knowledge which should be editable by the end-user.

excessive accidental complexity is a clear example of an anti-pattern.

D. Designing for Incompleteness

While newer software engineering methodologies struggle to increase the ability to adjust easily to change of both the process and the development team, they seem to generally have a certain agnosticism regarding the form of the produced software artifacts (probably an over-simplification since agile methodologies, for example, recommended the *simplest design that works*, which addresses form, in a certain way). This doesn't mean they are not aware of this "need to change". In fact, iterative means several cycles going from analysis to development, and back again. Some are also aware of the BIG BALL OF MUD pattern (or anti-pattern, depending on the perspective); the practice of *refactoring* after each iteration in order to cope with *design debt* is specifically included to address that [13]. But the problem seems to remain in the sense that the outcome — w.r.t. *form* — of each iteration is mostly synthesized as if it would be the last one (albeit knowing it isn't).

Yet, if these systems are accepted and regarded as being *incomplete by design*, it seems reasonable to assume benefits when actively *designing them for incompleteness*. If we shift the way we develop software to *embrace change*, it seems a natural decision to deliberately design that same software to best cope with change. Citing the work of Garud et al.: *"The traditional scientific approach to design extols the virtues of completeness. However, in environments characterized by continual change (new solutions) highlight a pragmatic approach to design in which incompleteness is harnessed in a generative manner. This suggests a change in the meaning of the word* design *itself – from one that separates the process of design from its outcome, to one that considers design as both the medium and outcome of action."* [7]

This is in particular dissonance with the current approaches to software engineering, where most processes attempt to establish a clear line between designing and developing, specifying and implementing. Though it seems that, should we wish to harness *continual change*, that distinction no longer suits our purposes: *design should become both the medium and outcome of action*. Consequently, we are thus looking forward not just for a process to be effective and agile, but to what *form* should agile software take.

E. Article Structure

The remaining of the article is divided into three main sections. We will first present a literature survey of the state-of-the-art regarding several concepts and techniques used to harness the specification and construction of these *incomplete by design* systems. In Section III, we'll use return to the motivational example and delve into the Adaptive Object-Model meta-architectural pattern to further detail its related concerns. Section IV aims to summarize the current known open issues in the field, and discuss issues on research design and empirical validation for assessing

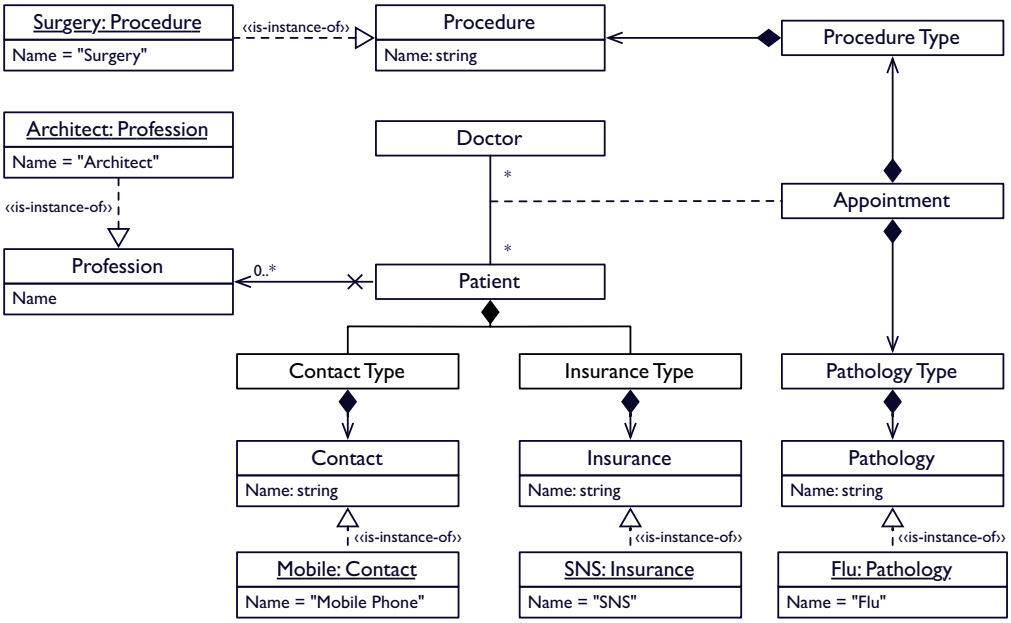


Figure 3. A *refactored* solution for the diagram in Figure 2, mainly depicting the elements that were changed/added for providing a mechanism to cope with open inheritance and enumerations. This example makes extensive use of the TYPE-OBJECT pattern (see Section III).

Adaptive Object-Models. We will finish this article by drafting some conclusions and pointing to future work.

II. STATE-OF-THE-ART

One way to design software able to cope with incompleteness is to encode the system's concepts into higher-level representations, which could then be systematically synthesized — desirably, in an automatic fashion — into executable artifacts, thus reducing the overall effort (and complexity) of changing it. An overview of the several concepts that will be approached in this section is shown in Figure 4.

A. Fundamentals

1) Variability: Software variability is the need of a software system or artifact to be changed, customized or configured for use in different contexts [14]. High variability means that the software may be used in a broader range of contexts (i.e., the software is more reusable). The degree of variability of a particular system is given by its *Variation Points*, or roughly the parts which support (re)configuration and consequently tailoring the product for different contexts or for different purposes. Variability is a well-known concept in Software Product Lines, which will be covered later.

2) Adaptability and Self-Adaptive Systems: While variability is given by context, the capability of software systems to react efficiently to changed circumstances is called Adaptability. The main difference relies on what has changed and what is being changed accordingly. The same software may be reconfigured to be used in different contexts (e.g., by recompiling with an additional component), and this provides Variability. The mechanisms that allow software to change its behavior (without recompiling) is called Adaptability. Adaptive systems may thus be defined as open systems which are able to fit their behavior according to either (or both) external or internal changes. The work by Andresen et al. [15] identifies some enabling criteria for adaptability in enterprise systems. Further work by Meso et al. [16] provides a insight into how agile software development practices can be used to improve adaptability.

Self-adaptation is a particular case of Adaptability, when software systems are empowered with the ability to adjust their own behavior themselves during run-time, in response to both their perception of the environment and itself [17]. Despite that in self-adaptation often (i) the change agent is the system itself in reaction to the external world, and (ii) the scope of adaptability is well-defined *a priori*, there is an extensive amount of research that still applies to systems that need adaptation, without having to adapt themselves.

B. Incompleteness

1) The Wiki Way: Earlier in 1995, Cunningham wrote a set of scripts that allowed collaborative editing of webpages inside the very same browser used to view them [18], and named this system *WikiWikiWeb*. He chose this word because of the analogy between its meaning (quick) and the underlying philosophy of its creation: a *quick-web*. Up until now, wikis have gradually become a popular tool on

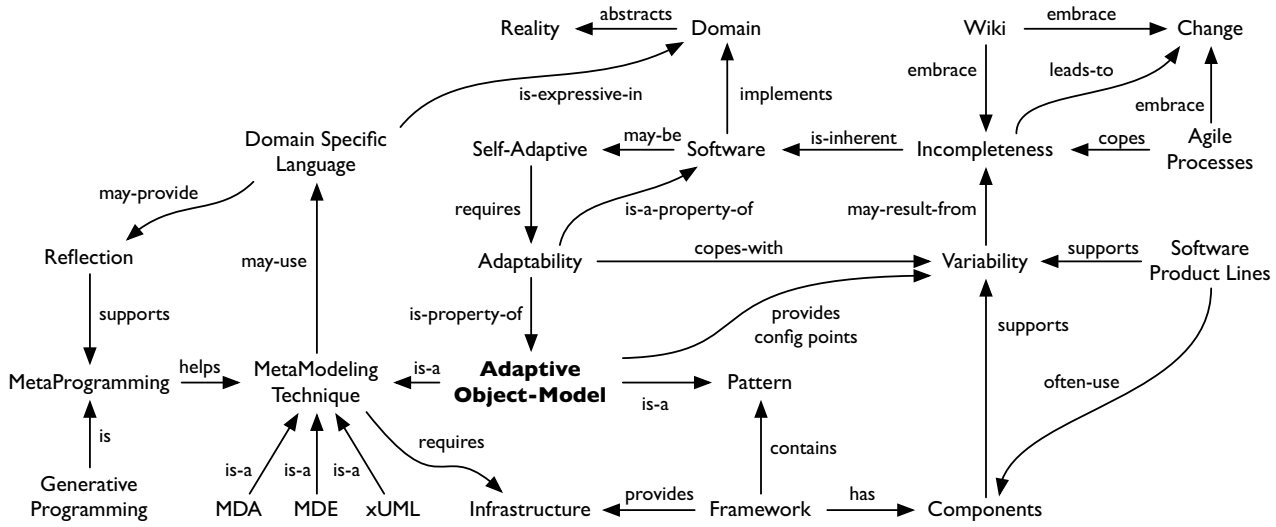


Figure 4. Concept map representing the relationship between several areas, concepts and techniques related to Adaptive Object-Models.

several domains, including that of software development (e.g., to assist the creation of lightweight software documentation [19]) — they ease collaboration, provide generalized availability of information, and allow to combine different types of content (e.g., text, models, code) into the same infrastructures.

2) Characteristics: The overall success of wikis may be due to a set of design principles drafted by Cunningham that overall tend to embrace *change* and *incompleteness*, namely (i) Open, (ii) Incremental, (iii) organic, (iv) Universal, (v) Overt, (vi) Tolerant, (vii) Observable and (viii) Convergent. Their original definition may be looked up in [20].

3) Wikis and Incomplete Systems: Likewise, we may also regard the usage (and development) of a software systems as a team work. The underlying design and the knowledge that lead to it (e.g., requirements, use-cases, models) is mostly devised and shared between developers and stakeholders. Data is collaboratively viewed and edited among users. Oddly, not only these circles seem to be disjoint, but emergent knowledge from this collaboration seem to start and end towards the artifacts themselves, despite the fact they have (and are) built incrementally.

Incompleteness, once again, seems to be the key. Instead of regarding it as a defect, we should embrace it as the means through which the system evolves, thus fulfilling its function. WikiWikiWeb seemed to have realized this fundamental ideal, and for us it is reasonable to conjecture about attempting the same underlying principles [20] to other computational systems as well:

- **Open.** Should a resource be found to be incomplete or poorly organized, the end-user can evolve it as they see fit.
- **Incremental.** Resources can link to other resources, including those who have not yet been brought into existence.
- **Organic.** Structure and content are open to editing and evolution. Evolution may be made more difficult if it is mandatory for information to strictly conform to a pre-established model.
- **Universal.** The same (or very similar) mechanisms for modifying data and model should be exposed by the system with no apparent distinction.
- **Overt.** End-user evolution should be made by non-programmers. The introduction of linguistic constructions (such as textual syntax) is usually required in order to provide formalization. However, such constructions may reveal unnatural, intrusive and complex to the end-user, thus model edition should be made as readily apparent (and transparent) as possible.
- **Tolerant.** Interpretable behavior is preferred to system halt.
- **Observable.** Activity should be exposed and reviewed by end-users, fomenting social collaboration.
- **Convergent.** Duplication is discouraged and removed by incremental restructuring and linking to similar or related content.

To reach expand these principles inherent to the *WikiWikiWeb* to other areas of software, we need a particular degree of adaptability — both to the developers and to the end-users — from the infrastructure which is neither commonly found, nor particularly easy to design.

C. Abstraction

Wikipedia [21] has several articles defining the concept of abstraction, depending on the scientific area: *"Conceptually, it is the process or result of generalization by reducing*

the information content of a concept or an observable phenomenon, typically to retain only information which is relevant for a particular purpose. In mathematics, it is the process of extracting the underlying essence of a mathematical concept, removing any dependence on real world objects with which it might originally have been connected, and generalizing it so that it has wider applications or matching among other abstract descriptions of equivalent phenomena. In computer science, it is the mechanism and practice of abstraction reduces and factors out details so that one can focus on a few concepts at a time."

All definitions share one factor in common, i.e., that *abstraction* involves relinquishing some property (e.g., detail) to gain or increase another property (e.g., simplicity). For example, a common known use of abstraction is the level of programming language. Assembly is often called low-level because it exposes the underlying mechanisms of the machine with an high degree of fidelity. On the other end, Haskell is an high-level language, struggling to hide as much as possible the underlying details of its execution. The latter trades execution *performance* in favor of *cross-platform* and *domain expressiveness*.

In this sense, abstractions are never to be considered *win-win* solutions. For example, Joel Spolsky [22] observes a recurrent phenomena in technological abstractions called *Leaky Abstractions*, which occurs when one technological abstraction tries to completely hide the concepts of another, lower-level technology, and sometimes the underlying concepts *"leak through"* those supposed invisible layers, rendering them visible. For example, an high-level language may try to hide the fact that the program is being executed at all by a *von-neumann* machine. In this sense, although two programs may be functionally equivalent, memory consumption and processor cycles may eventually draw a clear separation between them. Hence, the programmer may need to learn about the middle and lower-level components (i.e., processor, memory, compiler, etc.) for the purpose of *designing a program that executes in a reasonable time*, thus breaking the abstraction. He goes as far as hypothesizing that *"all non-trivial abstractions, to some degree, are leaky"*. Hence, good abstractions are specifically designed to express the exactly intended details in a specific context, while relinquishing what is considered unimportant.

1) Metaprogramming: Metaprogramming consists on writing programs that generate or manipulate either other programs, or themselves, by treating code as data [23]. Historically, it is divided into two languages: (i) the meta-language, in which the meta-program is written, and (ii) the object language which the metaprogram produces or manipulates. Nowadays, most programming languages use the same language for the two functions [24], either by being homoiconic (e.g., Lisp), dynamic (e.g., Python) or by exposing the internals of the runtime engine through APIs (e.g., Java and .NET). Claims about the economic benefits in terms of development and adaptability have been studied and published for more than twenty years [25], though its focus is on code-level manipulation and not on domain artifacts.

2) Meta-modeling: Supporting the use of models during runtime is an answer to high-variability systems [6], where the large semantic mismatch between specification and implementation artifacts in traditional systems can be reduced by the use of models, meta-models, and metadata in general. Metamodeling is thus the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modeling a predefined class of problems [26] (i.e., a model to specify models).

3) Reflection: Reflection is the property of a system that allows to observe and alter its own structure or behavior during its own execution. This is normally achieved through the usage and manipulation of (meta-)data representing the state of the program/system. There are two aspects of such manipulation: (i) introspection, i.e., to observe and reason about its own state, and (ii) intercession, i.e., to modify its own execution state (structure) or alter its own interpretation or meaning (semantics) [24]. Due to this properties, reflection is a key property for metaprogramming.

4) Domain Specific Languages: A domain-specific language (DSL) is a programming or specification language specifically designed to suit a particular problem domain, representation technique, and/or solution technique. They can be either visual diagramming languages, such as UML, programatic abstractions, such as the Eclipse Modeling Framework, or textual languages, such as SQL. The benefits of creating a DSL (along with the necessary infrastructure to support its interpretation or execution) may reveal considerable whenever the language allows a more clear expression of a particular type of problems or solutions than pre-existing languages would, and the type of problem in question reappears sufficiently often (i.e., recurrent, either in a specific project, like extensive usage of mathematical formulae, or global-wise, such as database querying).

The creation of a DSL can be supported by tools such as AntLR [27] or YACC [28], which take a formalized grammar (e.g., defined in a meta-syntax such as BNF), and generate parsers in a supported target language (e.g., Java). Recently, the term DSL has also been used to coin a particular type of syntactic construction within a general purpose language which tends to more naturally resemble a particular problem domain, but without actually extending or creating a new grammar. The Ruby community, for example, has been enthusiastic in applying this term to such syntactic sugar [29].

Domain-specific languages share common design goals that contrast with those of general-purpose languages, in the sense they are (i) less comprehensive, (ii) more expressive in their domain, and (iii) exhibit minimum redundancy. Language Oriented Programming [30] considers the creation of special-purpose languages for expressing problems as a

standard methodology of the problem solving process.

5) Meta-Architectures: We have already seen several techniques used to address systems with high-variability needs. There is, nonetheless, differences between them. For example, some do not parse or interpret the system definition (meta-data) while it is running: Generative Programming and Metamodeling rely on code generation done at compile time. Reflection is more of a property than a technique by itself, and the level at which it is typically available (i.e., programming language) is inconvenient to deal with domain-level changes. Domain Specific Languages are programming (or specification) languages created for a specific purpose. They are not generally tailored to deal with change (though they could), and they do require a specific infrastructure in order to be executed.

Meta-architectures, or reflective-architectures, are architectures that strongly rely on reflective properties, and may even dynamically adapt to new user requirement during runtime. Pure OO environments, and MOF-based systems are examples of such architectures, as they make use of meta-data to create different levels that sequentially comply to each other. The lowest level in this chain is called the data level, and all the levels above the meta-data levels, but the line that separates them is frequently blurred as both are data.

D. Approaches

1) Software Product Lines: A software product line (SPL) is a set of software systems which share a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [31]. Software product line development, encompasses software engineering methods, tools and techniques for supporting such approach. A characteristic that distinguishes SPL from previous efforts is predictive versus opportunistic software reuse. Rather than put general software components into a library in the hope that opportunities for reuse will arise, software product lines only call for software artifacts to be created when reuse is predicted in one or more products in a well defined product line.

2) Naked Objects: Naked Objects takes the automatic generation of graphical user interfaces for domain models to the extreme. This software architectural pattern, first described in Richard Pawson's PhD thesis [32] which work includes a thorough investigation on prior known uses and variants of this pattern, is defined by three principles:

1) All business logic should be encapsulated onto the domain objects, which directly reflect the principle of *encapsulation* common to object-oriented design.
2) The user interface should be a direct representation of the domain objects, where all user actions are essentially creation, retrieval and message send (or method invokation) of domain objects. It has been argued that this principle is a particular interpretation of an object-oriented user-interface (OOUI).
3) The user interface should be completely generated solely from the definition of the domain objects, by using several different technologies (e.g., source code generation or reflection).

The work of Pawson further contains some controversial information, namely a foreword by Trygve Reenskaug, who first formulated the model-view-controller (MVC) pattern, suggesting that Naked Objects is closer to the original MVC intent than many subsequent interpretations and implementations.

3) Domain-Driven Design: Domain-driven design (DDD) was coined by Eric Evans in his books of the same title [33]. It is an approach to developing software for complex needs by deeply connecting the implementation to an evolving model of the core business concepts, which encompasses a set of practices and terminology for making design decisions that focus and accelerate software projects dealing with complicated domains. The premise of domain-driven design is the following: (i) placing the project's primary focus on the core domain and domain logic, (ii) basing complex designs on a model, and (iii) initiating a creative collaboration between technical and domain experts to iteratively cut ever closer to the conceptual heart of the problem.

4) Model-Driven Engineering: Model-Driven Engineering (MDE) is a metamodeling technique that strives to close the gap between specification artifacts which are based upon high-level models, and concrete implementations. A conservative statement claims that MDE tries to reduce the effort of shortening (not completely closing) that gap by generating code, producing artifacts or otherwise interpreting models such that they become executable [2].

Proponents of MDE claim several advantages over traditional approaches: (i) shorter time-to-market, since users model the domain instead of implementing it, focusing on analysis instead of implementation details; (ii) increased reuse, because the inner workings are hidden from the user, avoiding to deal with the intricate details of frameworks or system components; (iii) fewer bugs, because once one is detected and corrected, it immediately affects all the system leading to increased coherence; (iv) easier-to-understand systems and up-to-date documentation, because the design is the implementation so they never fall out of sync [6]. One can argue if these advantages are exclusive of MDE or just a consequence of "raising the level of abstraction" (see *Domain Specific Languages*).

Downsides in typical generative MDE approaches include the delay between model change and model instance execution due to code generation, debugging difficulties, compilation, system restart, installation and configuration of the new system, which can take a substantial time and must take place within the development environment [6]. Once

again, it doesn't seem a particular downside of MDE, but a general property of normal deployment and evolution of typical systems.

More interesting counter-points to MDE adoption will be addressed in Section IV. It seems worthwhile to note that the prefix *Model-driven* seems to be currently serving as a kind of umbrella definition for several techniques.

5) Model-Driven Architecture: Model-driven architecture (MDA) in an approach to MDE proposed by the OMG, for the development of software systems [34]. It provides a set of guidelines to the conception and use of model-based specifications and may be regarded as a type of domain engineering. It bases itself on the Meta Object Facility (MOF) [35], which main purpose is to define a strict, closed metamodeling architecture for MOF-based systems and UML itself, provides four modeling levels (M_3, M_2, M_1 and M_0), each conforming to the upper one (M_3 is in conformance to itself).

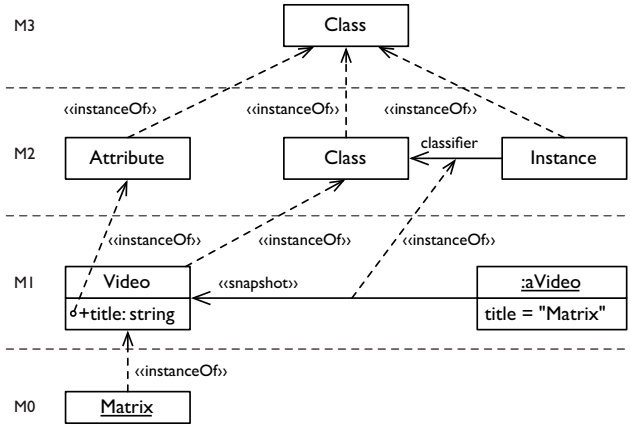


Figure 5. Layers of Abstraction in the Meta-Object Facility (MOF). Each level is in direct conformance to the upper level. The classes named *Class*, from both level 2 and 3, represent the same concept.

Like most of the MDE practices, MDA thrives within a complex ecosystem with specialized tools for performing specific actions. Moreover, MDA is typically oriented for generative approaches, using systematic offline transformation of high-level models into executable artifacts. for example, trying to answer MDA's objective of covering the entire gap between specification and implementation, xUML was developed. It is a UML profile which allows the graphical representation of a system, and takes an approach in which platform specific implementations are created automatically from platform-independent models, with the use of model compilers.

While some complex parts of MDA allow runtime adaptivity, developers seldom acquire enough knowledge and proficiency of the overall technologies to make them cost (and time) effective in medium-sized applications. Runtime adaptivity may be approached in different ways, including the use of reflection, and the interpretation of models at runtime [6], covering the concept of *Adaptive Object-Model* (see section II-E3).

E. Infrastructures

1) Frameworks: Object-oriented frameworks are reusable designs of all or part of a software system described by a set of abstract artifacts and the way they collaborate [36]. They aim to provide both an infrastructure, through a COMPONENT LIBRARY and pre-defined interconnections among them, which may all be (re-)configured and extended to address different problems in a given domain. Good frameworks are able to reduce the cost of development by an order of magnitude. It should be stressed that software frameworks are more than just a collection of reusable components (also known as LIBRARY); a framework usually makes use of the *Hollywood Principle*[1] to promote high cohesion and low coupling in object-oriented designs, by ensuring a THREAD OF CONTROL.

2) Generative Programming: One common approach to address variability and adaptability is the use of Generative Programming (GP) methods, which transform a description of a system (model) based in primitive structures [37], into either executable code or code skeleton. This code can then be further modified and/or refined and linked to other components [38]. Generative Programming deals with a wide range of possibilities including those from Aspect Oriented Programming [39], [40] and Intentional Programming [41].

Because GP approaches focus on the automatic generation of systems from high-level (or, at least, higher-level) descriptions, it is arguable whether those act like a meta-model of the generated system. Still, the resulting functionality is not directly produced by programmers but specified using domain-related constructs. In summary, GP works as an off-line code-producer and not as a run-time adaptive system [42].

This technique typically assumes that (i) changes are always introduced by developers (change agents), (ii) within the development environment, and (iii) that a full transformation (and most likely, compilation) cycle is affordable (i.e., no run-time reconfiguration). When these premises fail to hold, generative approaches are easily overwhelmed [6].

3) Adaptive Object-Models: In search for flexibility and run-time adaptability, many developers had systematically applied code and design reuse of particular domains, effectively constructing higher-levels representations (or abstractions). For example, some implementations have their data structure and domain rules extracted from the code and stored externally as modifiable parameters of well-known structures, or statements of a DSL. This strategy gradually transforms some components of the underlying system into

[1] A well-known cliché response given to amateurs auditioning in Hollywood: *"Don't call us, we'll call you"*.

an Interpreter or Virtual Machine whose run-time behavior is defined by the particular instantiation of the defined model. Changing this model data thus results on the system following a different business domain model. Whenever we apply these techniques based upon object-oriented principles and design, we are using an architectural style, known as an Adaptive Object-Model [43].

Shortly, it is (i) a class of systems' architectures, (ii) heavily based on Metamodeling and Object-Oriented design, (iii) which often uses a Domain Specific Language, (iv) usually have the property of being Reflective, and (v) with the intent of exposing its configuration to the end-user. Because we are abstracting a set of systems and techniques into a common underlying architecture heavily based on object-oriented metaprogramming/metamodelling, we categorize the AOM as a meta-architecture.

The evolution of the core aspects of an AOM can be observed by the broad nomenclature used in the literature in the past couple of decades (e.g., Type Instance, User Defined Product Architecture, Active Object-Models and Dynamic Object Models). The concept of Adaptive Object-Model is inherently coupled with that of an architectural pattern, as it is an effective, documented, and prescriptive solution to a recurrent problem.

It should therefore be noted that most AOMs emerge from a bottom-up process [43], resulting in systems that will likely use only a subset of these concepts and properties, and only when and where they are needed. This is in absolute contrast with top-down efforts of specific meta-modeling techniques (e.g., Model-Driven Architecture) where the whole infrastructure is specifically designed to be as generic as possible (if possible, to completely abstract the underlying level).

The concepts of End-User Programming and Confined Variability — the capability of allowing the system's users to introduce changes and thus control either part, or the entire system's behavior — are significative consequences of the AOM architecture which aren't easily reconcilable with other techniques such as Generative Programming.

III. Architecture and Design of Adaptive Object-Models

The basic architecture of an Adaptive Object-Model is divided into three parts, or levels, that roughly correspond to the levels presented by MOF: M_0 is the operational level, where system data is stored, M_1 is the knowledge level where information that defines the system (i.e., the model) is stored, and M_2 is the design of our supporting infrastructure. M_0 and M_1 are variants of our system. M_2, the meta-model, is usually invariant — should it need to change, we would have to transform M_0 and M_1 to be compliant with the new definition.

A. Making the Structure Agile

In Section I, we have shown a refactored example which made use of a small set of patterns to introduce the desired adaptability into the original system. As always, one key to good software design is two-fold: (i) recognize the things that will often change in a predictable way, and (ii) recognize what it rarely does not; it is the search for *patterns* and *invariants*.

B. The Type-Object Pattern

In the context Object-Oriented Programming and Analysis, the Type of an object is defined as a Class, and its Instance as an Object which conforms to its class. A typical implementation of our example system would hardcode into the program structure (i.e., source-code) every modeled entity (e.g., Patient). Would the system needed to be changed (e.g., to support a new entity) the source-code would have to be modified.

However, if one anticipate this change, objects can be generalized and their variation described as parameters, just like one can make a function that sums any two given numbers, regardless of their actual value. The Type Object pattern, depicted in Figure 7, identifies the practice of splitting such a class in two: a Type called EntityType, and its Instances, called Entity.

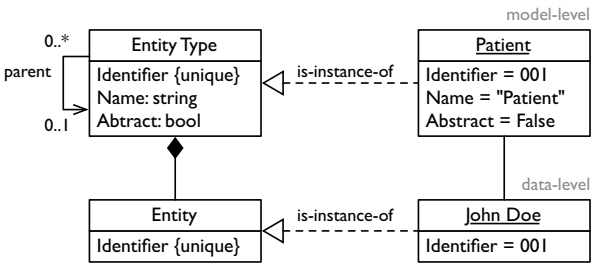


Figure 7. The Type-Object pattern.

Using this pattern, Patients, Doctors, Appointments, etc. all become instances of Entity Types, therefore meta-data. The actual system data, such as a the patient John Doe, are now represented as instances of Entity. Because data and meta-data are values beyond the program structure, they can be changed during run-time.

We'll often extend and customize the original design of patterns to further enhance the model semantics. For example, by supporting the notion of inheritance through an optional relation between EntityTypes, which do incidentally solve the problem of open-inheritance. Provided sufficient mechanisms exists to allow the end-user customization of the model-level, new specializations (e.g., Procedures, Pathologies, etc.) can be added without modifying the source-code.

C. The Property Pattern

Similarly to the Type-Object pattern, we face a similar problem with the attributes of an object, such as the Name and Age of a Patient, which are usually stored as values of fields of an object, which have been defined in its

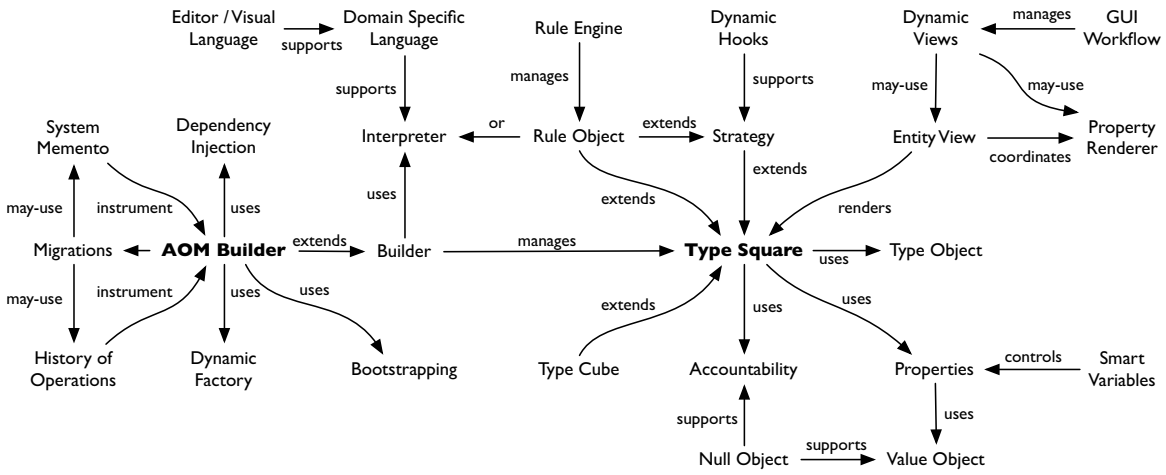


Figure 6. Pattern map of design patterns and concepts related to Adaptive Object-Models. Adapted from [44].

appropriate classes. Once again, anticipation of this change leads to the PROPERTY pattern; a similar bi-section between the definition of a property and its corresponding value as depicted in Figure 8.

Using this pattern, the Name, Age, Birthday, Profession, and several attributes of the domain's entities become instances of Property Types, and their particular values instances of Property. Again, this technique solves the problem of adding (or removing) more information to existing entities beyond those originally designed.

D. The Type-Square Pattern

The two previous patterns, type-object and property, are usually used in conjunction, resulting in what is known as the TYPE-SQUARE pattern, which poses the core of an AOM. If we add the instantiations of these classes, we get the diagram depicted in Figure 9.

In this picture, the objects represent both the systems' data and model, while the classes represent our static, abstract infrastructure. The ability to represent an increasing number of different — and useful — systems is directly dependent on the power of the underlying infrastructure.

E. Revisiting the PROPERTY Pattern

Literature in AOM used to describe a form to capture relationship between different objects by using the ACCOUNTABILITY pattern [45]. But what exactly is a field, a relation or a property? Object fields, in OOP, are used to store either values of native types (such as int or float in Java) or references to other objects. They can also be either scalars or collections. Some OO languages (e.g., smalltalk) treat everything as an object, and as such do not make any difference from native types to references. Some also discard scalar values and instead use singleton sets. We may borrow these notions to extend the PROPERTY pattern in order to support associations between entities, provided we are able to state properties such as cardinality, navigability, role, etc. The actual differentiation between what is a scalar property and a relation, becomes an implementation detail. Figure 10 depicts such design.

One hook introduced between the abstraction and the underlying language is the use of the Native Type property in Entity Type, to allow any custom Entity Type to be directly mapped into a native type of the underlying language (such

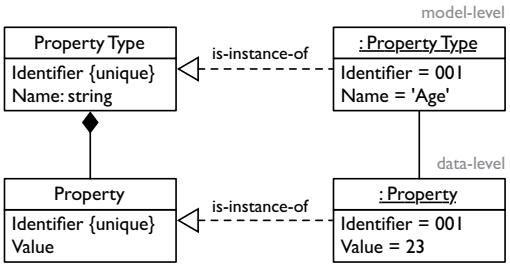


Figure 8. The PROPERTY pattern, which separates the definition of an object's property and its corresponding value.

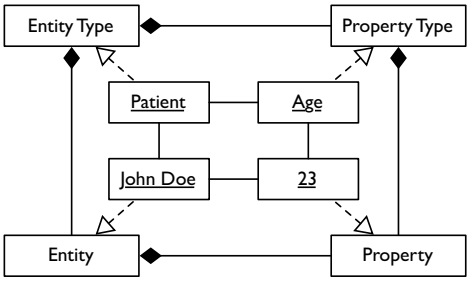


Figure 9. The TYPE-SQUARE pattern, which is a composition of the TYPE OBJECT and PROPERTY patterns.

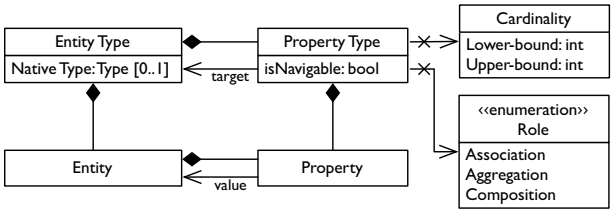


Figure 10. An extension of the TYPE-SQUARE pattern, using a variant of the ACCOUNTABILITY pattern.

as integers and strings).

There are also several logical restrictions related to the semantics of this design. For example, the lower and upper bound in cardinality should restrain the number of associations from a single Property to Entities. Likewise, Properties should only link to Entities which are of the same Entity Type as that defined in Property Type. The complete formalization of the semantics of the presented models is outside of the scope of this work.

F. Self-compliance

If our meta-model has enough expressivity, one can reach the point where the model can be represented inside itself. MOF is an example of such self-compliance by making M_3 a self-describing level. There are several reasons to make that design choice: (i) it makes a strict meta-modeling architecture, since every model element on every level is strictly in correspondence with a model element of above level, and (ii) the same mechanisms used to view, modify and evolve data can be reused for meta-data. A way to accomplish this is by introducing the notion of a Thing, that abstracts elements at any level. A Thing is thus an instance of another Thing, including of itself, such as depicted in Figure 11. This implies that during bootstrapping, the system would first need to load its own meta-model. When applicable, Entity Types are actually M_2 Entities, thus requiring a mechanism to inherently convert between them.

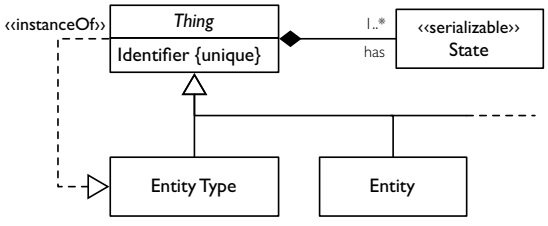


Figure 11. The THING pattern, which decouples the *Identity* of an object from its *State*.

G. Versioning

As is also shown by Figure 11, the identity of each instance is maintained as Things, while the respective details are maintained as States. The decoupling of these two concepts may be leveraged to provide auditability capabilities, thus answering a common request when building information systems. Auditability may be reached by allowing users to access past versions of an instance, and thus to understand how such instance has evolved.

Figure 12 depicts an example of this mechanism. Two distinct values exist for the same Property Type, corresponding to two different changes the instance has been through over time.

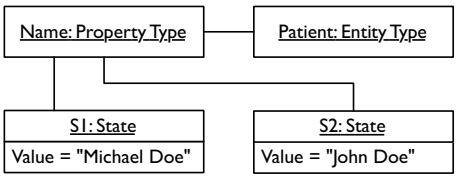


Figure 12. S1 and S2 are two different states — or snapshots — of the property name. Although a typical system usually stores only the most recent snapshot, some techniques rely on having the history of an object.

H. Making the Behavior Agile

In addition to structure, such as Entities, Properties and Associations, systems usefulness also relies on the ability to support rules and automatic behavior. Some examples of these include, but are not limited to, (i) Constraints, such as relationship cardinality, navigability, type redefinition, default values, pre-conditions, etc. (ii) Functional Rules, which include reactive logic such as triggers, events and actions, and (iii) Workflows.

During the instantiation of the object-model, after all types of objects and their respective attributes are created, there are some operations that can be applied to them. Some of these operations are simple Strategies, relatively immutable or otherwise parameterized, which can be easily described in the metadata as the method to be invoked along with the appropriate Strategy.

When the desired behavior reaches a certain level of complexity and dynamism, then a domain specific language can be designed using RULE-OBJECTS [46]. In this case, primitive rules are defined and composed together with logical objects, parsed into a tree structure to be interpreted during runtime. The use of patterns like SMART VARIABLES [47] and TABLE LOOKUP reveal useful for triggering automatic updates and validations of property values. More complex systems can make use of StateMachines and Workflows both for data and human-computer interaction.

A common problem that arises with the abstraction of rules, is that the developer may fall in the trap of creating (and then maintaining) a general-purpose programming language, which may result in increased complexity regarding the implementation and maintenance of the target application, far beyond what would be expected if those rules were

hardcoded [42]. It should be kept in mind that the goal is not to implement the whole application in a new language; just what changes.

I. Rules

Figure 13 depicts a design extending the RULE OBJECT pattern and presented in [48], which allows the definition of: (i) Entity Type invariants, which are predicates that must always hold, (ii) derivation rules for Property Types and Views, (iii) the body of Methods, (iv) guard-conditions of Operations, etc. Even some structural enforcement, such as the Cardinality and Uniqueness of a Property Type may be unified into conditions. Methods, which are used-defined Batches of Operations, may be invoked manually, or triggered by Events, thus providing enough expressivity to specify STATE MACHINES. This design ensures the capability of the system to enforce semantic integrity during normal usage and assisting model evolution.

J. Views

In the design presented so far, Views are regarded as Entity Types which have a derivation rule that returns a collection, and every property have a second derivation rule (often manipulating each item of that collection). They allow the existence of virtual Entity Types where their information is not stored by inferred, similar the querying mechanisms of relational databases (SQL). For example, we could consider a requirement for the system presented in Figure 2, that specifies a list of items composed by every doctor in the medical center, along with the number of high-risk procedures and their total cost. This would be represented by a new Entity Type ("High-Risk Treatments Income by Doctor") that iterates over Doctors and their Procedures. Since the latter is also a derived property, it should be specified as a rule of Doctor and so on.

K. Evolution

Has already discussed, structural integrity of the run-time model is asserted through rules stated in the meta-model. For example, Entities should conform to their specified Entity Type (e.g., they should only hold Properties to which its Property Type belong to the same Entity). Nonetheless, evolving the model may corrupt structural integrity. For example, when moving a mandatory Property Type to its superclass, if it doesn't have a default value, it can render some Entities structurally inconsistent. Some of these issues can be coped with, by foreseeing integrity violations and applying prior steps to avoid them (e.g., one could first introduce a default value before moving the Property Type to its superclass). Through careful annotation of model-level operations (e.g., by reducing the applicability scope through pre-conditions so that a well-formalized semantics is established), one can increase the confidence on maintaining structural integrity.

Another issue arrises when parts of a composite evolution violate model integrity, although the global result would be valid. For example if a Property Type is mandatory, one cannot delete its Properties without deleting itself and vice-versa. This problem is solved by the use of transactions or change-sets, and by suspending integrity check until the end.

Semantic Integrity of a particular evolution, on the other hand, is much harder to ensure since it is not encoded as rules in the meta-model. State-based comparison of models have already shown this problem, because it is not always possible to just compare the results of an arbitrary evolution and accurately infer the performed operations. In our example, consider the scenario where a patient's age was being stored directly, but we now realize that having the birthdate is far less problematic. For that, the following modifications to the model are made: (i) rename Age to Date of Birth, (ii) reverse calculate it according to the current date, and (iii) move it to a the superclass Person.

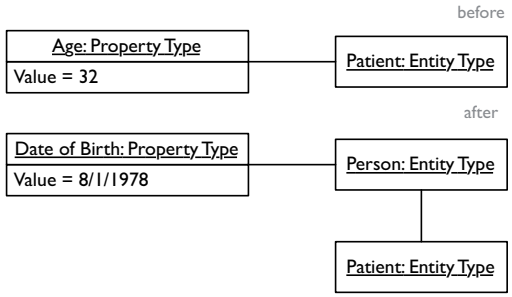


Figure 14. A model evolution example, where a property is both changed w.r.t. its semantics, and moved to the superclass.

Would we rely on the direct comparison of the initial and final models, a possible solution would be to delete the Age in Patient and create Date Of Birth in Person. Clearly, the original meaning of the intended evolution (e.g., that we wanted to transform ages to birth-dates) would be missed, and every data lost. Operation-based evolution solves this problem, and we can make use of the MIGRATION pattern [49] which express these changes through sequences of model-level operations that cascade into instance-level changes, as we'll see next.

L. (Co-)Evolution

Allowing (potentially collaborative) co-evolution of model and data by the end-user introduces a new set of concerns not usually found in classic systems. They are (i) how to preserve model and data integrity, (ii) how to reproduce previously introduced changes, (iii) how to access the state of the system at any arbitrary point in the past, and (iv) how to allow concurrent changes. All these concerns are directly related to traceability, reproducibility, auditability, disagreement and safety, and are commonly found, and coped with, on version-control systems.

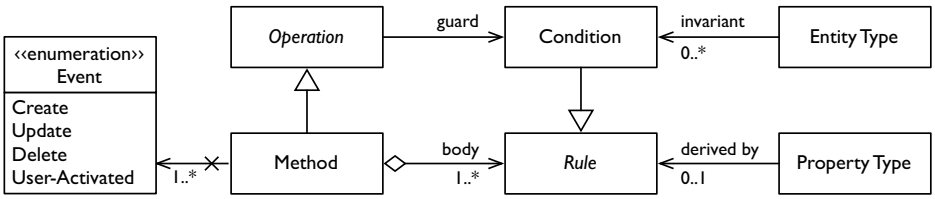


Figure 13. The dynamic core of an Adaptive Object-Model framework. This particular design supports method definition, guard conditions, class invariants, derived rules and event triggers.

Typically, Evolution is understood as the introduction of changes to the model. Yet, the presented design so far doesn't establish a difference between changing data or meta-data; both are regarded as evolutions of Things, expressed as Operations over their States, and performed by the same underlying mechanisms as depicted in Figure 15. To provide enough expressivity such that semantic integrity can be preserved during co-evolution, model-level Batches operate simultaneously over data and meta-data.

Sequences of Operations may be encapsulated as ChangeSets, following the HISTORY OF OPERATIONS pattern [49], along with meta-information such as user, date and time, base version, textual observations, and data-hashes, etc. Whenever the system validates or commits a ChangeSet, the Controller uses the merge mechanism depicted in Figure 15 (similarly to the SYSTEM MEMENTO pattern [49]) by (i) orderly applying each Operation to create a new State, (ii) dynamically overlaying the new States onto the base version, (iii) evaluating and ensuring behavioral rules, and finally (iv) producing a new version.

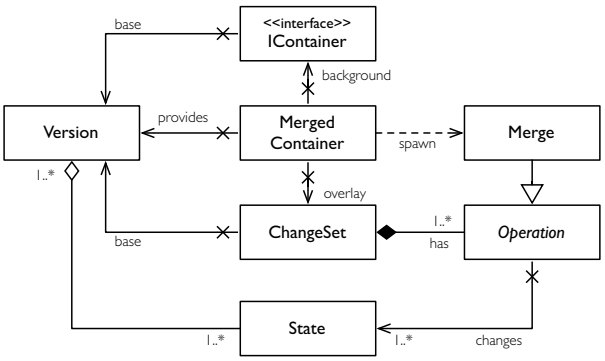


Figure 15. The merging mechanism of an AOM framework, which overlays a changeset to provide a merged view (i.e., a snapshot) of a system.

M. Enduser (Co-)Evolution

In order for the end-user to perform arbitrary model evolutions, a sufficiently large library of (probably composable) operations should be provided. If the sequences of operations over data and meta-data are preserved, it becomes possible to recover past states of the system. It also opens way to solve concurrent changes to the model, by allowing the existence of multiple branches of evolution, and provide disagreement and reconciliation mechanisms [48].

N. Warehousing

So far we have been incrementally empowering an infrastructure — or meta-model — to describe most of our example application. One issue remains though: how should this metadata be represented, accessed, and stored?

We should have in mind that, by describing both the structure and behavior of the system, it is to be interpreted in three distinct phases: (i) during system initialization, a process also known as bootstrapping, (ii) during the construction of objects, a phase typically known as "instantiating the object-model", and (iii) during the assessment and execution of rules.

Warehousing thus aim to hide the details of persistency from the rest of the system, exposing and consuming data and meta-data (i.e., Things), and managing versioning (i.e., through Versions and States). Its behavior can be extended and modified through inheritance and composition, as by the DECORATOR pattern. Transient memory-only, direct data-base access, lazy and journaling strategies (e.g., using CACHES) are just a few examples of existing (and sometimes simultaneous) configurations.

O. Representation

We already discussed that because this information will be readily available for runtime manipulation (e.g., in a database or other external storage mechanism), and not hardcoded, it allows the business model to be updated and immediately reflected in the applications.

Options for storage and manipulation of meta-data include relational and object-oriented databases, Domain Specific Languages, custom XML vocabularies, etc. Direct serialization of the model — e.g., using native language primitives — may simplify system initialization. Domain Specific Languages and XML vocabularies may need the usage of the INTERPRETER and BUILDER patterns.

However, one of the most powerful abilities of an AOM is to allow the end-user himself to introduce (confined) changes to the model at runtime. This raises a number of concerns which the AOM literature does not commonly address, and that may require more elaborate strategies to deal with the

evolution of data and meta-data, and which will be discussed in Section IV.

P. Persistency

The use of object-oriented databases (OODB) simplifies persistency in AOM-based systems, due to the object-oriented nature of the meta-model. Other techniques include model-to-model transformations for relational models, use of the filesystem for storing serialized objects, or BLOBs in databases.

Still, persistency based on static-scheming, such as automatic generation of DDL code for specifying relational databases schema and subsequent DML code for manipulating data, which attempts a semantic correspondence between both models, significantly increases the implementation complexity, particularly when dealing with model co-evolution [49]. This has been long known as object-relational impedance mismatch [50], and evidence of such issues may be observed in the way Object-Relational Mapping (ORM) frameworks attempt to deal with these issues, often requiring knowledge of both representations and manual specification of their correspondence (e.g., Migrations in RoR).

Therefore, though persistency may seem a solved issue, the highly-dynamic nature of AOMs and their inherent mismatch with relational-models suggest that new alternatives need to be investigated and tested, specially when dealing with multiple versions and migrations of data and metadata [49].

Q. Thread of Control

In Figure 6, the AOM BUILDER serves as a Controller for the system, and its key responsibility is to orchestrate the several other components in the framework by establishing a thread of control. It bootstraps the system by loading the meta-model, and the necessary versions of the domain-model from the Warehouse. It manages data requests by interacting with the Warehouse. It also provides several HOOKS to the framework through CHAINS OF RESPONSIBILITY and PLUGINS (e.g., interoperability with third-party systems by allowing subscribers to intercept requests). It is the AOM BUILDER that establishes the THREAD OF CONTROL for an AOM-based system.

R. User-Interface

Adaptive Graphical User-Interfaces (GUI) for AOMs work through inspection and interpretation of the model and by using a set of pre-defined heuristics and patterns [51]. The work in [48] describes an example of a minimalistic workflow for an automated GUI, based on: (i) a set of grouped entry-points declared in the model, and further presented to the user grouped by Packages, (ii) list of the instances by Entity Type or View, which show several details in distinct columns, inferred from special annotations made in the model, (iii) pre-defined automated views inferred by model inspection (edition and visualization) based on heuristics that consider the cardinality, navigability and role of properties, (iv) generic search mechanisms, (v) generic undo and redo mechanisms, (vi) support of custom panels for special types (e.g., dates) or model-chunks (e.g., user administration), through PLUGINS, etc.

This reactive user-interface also resembles a type of *offline mode*, similar to using version-control systems. User changes, instead of being immediately applied, are stored into the user Changeset, and sent to the main Controller (which would subsequently assert the resulting integrity of applying changes, and provide feedback on behavioral rules). The user can *commit* its work to the system when she wants to save it, review the list of Operations she has made, and additionally submit a descriptive text about her work.

Awareness of the system also makes use of several feedback techniques such as (i) graphics showing the history of changes, (ii) alerts for simultaneous pendent changes in the same subjects from other users, (iii) reconciliation wizards whenever conflicts are detected due to concurrent changes, etc.

S. Towards a Pattern Language

When building systems, there are recurrent problems which have proven, recurrent solutions, and as such are known as *patterns*. These patterns are presented as a three-part rule, which expresses a relation between a certain context, a problem, and a solution. A software design pattern addresses specific design problems specified in terms of interactions between elements of object-oriented design, such as classes, relations and objects [52]. They aren't meant to be applied as-is; rather, they provide a generic template to be instantiated in a concrete situation.

1) Categorization: The growing collection of AOM-related patterns which is forming a domain pattern language [49], [44], [53], is currently divided into six categories (i) Core, which defines the basis of a system, (ii) Creational, used for creating runtime instances, (iii) Behavioral, (iv) GUI, for providing human-machine interaction, (v) Process, to assist the creation of a AOM, and (vi) Instrumental, which helps the instrumentation:

- **Core.** This set of patterns constitutes the basis for an AOM-supported system. The patterns included in this category are Type Square, Type Object, Properties, Accountability, Value Object, Null Object and Smart Variables.
- **Creational.** These patterns are the ones used for creating runtime instances of AOMs: Builder, AOM Builder, Dynamic Factory, Bootstrapping, Dependency Injection and Editor / Visual Language.
- **Behavioral.** Behavioral patterns are used for adding and removing behavior of AOMs in a dynamic way.

They are Dynamic Hooks, Strategy, Rule Object, Rule Engine, Type Cube and Interpreter.
- **GUI.** User-interface rendering patterns have already been mentioned: Property Renderer, Entity View, and Dynamic View. Related to UI there's to add the GUI Workflow pattern.
- **Process.** Includes the patterns used in the process of creating AOMs. An AOM has usually much of a framework in it. The following patterns are good practices when building a framework as well as when building an AOM: Domain Specific Abstraction, Simple System, Three Examples, White Box Framework, Black Box Framework, Component Library, Hot Spots, Pluggable Objects, Fine-Grained Objects, Visual Builder and Language Tools.
- **Instrumental.** Patterns that help on the instrumentation of AOMs, namely, Context Object, Versioning, History and Caching.

2) Core Patterns:

- **Type Object.** As described in [54] and [55], a *TypeObject* decouples instances from their classes so that those classes can be implemented as instances of a class. Type Object allows new "classes" to be created dynamically at runtime, lets a system provide its own type-checking rules, and can lead to simpler, smaller systems.
- **Property.** The Property pattern gives a different solution to class attributes. Instead of being directly created as several class variables, attributes are kept in a collection, and stored as a single class variable. This makes it possible for different instances, of the same class, to have different attributes [45].
- **Type Square.** The combined application of the *TypeObject* and *Property* patterns result in the *TypeSquare* pattern [45]. Its name comes from the resulting layout when represented in class diagram, with the classes *Entity*, *EntityType*, *Attribute* and *AttributeType*.
- **Accountability.** Is used to represent different relations between parties, as described in [45] and [46], using an *AccountabilityType* to distinguish between different kinds of relation.
- **Composite.** This pattern consists of a way of representing part-hole hierarchies, by using the *Rule* and *CompositeRule* classes [54].
- **Strategy.** Strategies are a way to encapsulate behavior, so that it is independent of the client that uses it. Rules are Strategies, as they define behavior that can be attached to a given *EntityType* [54].
- **Rule Object.** This pattern results from the application of the Composite and Strategy patterns, for the representation of business rules by combining simpler elementary constraints [44].
- **Interpreter.** An AOM consists of a runtime interpretation of a model. The Interpreter pattern is used to extract meaning from a previously defined user representation of the model [54].
- **Builder.** A model used to feed an AOM-based system is interpreted from its user representation and a runtime representation of it is created. The Builder pattern is used in order to separate a model's interpretation from its runtime representation construction [54].

3) Graphical User Interface Patterns: The patterns in [53] focus specifically on User Interface (UI) generation issues when dealing with AOMs. In traditional systems, data presented in User Interfaces is usually obtained from business domain objects, which are thus mapped to UI elements in some way. In AOMs business objects exist under an additional level of indirection, which has to be considered. In fact, it can be taken into our advantage, as the existing meta-information, used to achieve adaptivity, can be used for the same purpose regarding user interfaces. User interfaces can this way be adaptive to the domain model in use.

- **Property Renderer.** Describes the handling of user-interface rendering for different types of properties.
- **Entity View.** Explains how to deal with the rendering of *EntityTypes*, and how *PropertyRenderers* can be coordinated for that purpose.
- **Dynamic View.** Approaches the rendering of a set of entities considering layout issues and the possibility of coordinating *EntityViews* and *PropertyRenderers* in that regard.

IV. Researching Adaptive Object-Models

The Adaptive Object-Model and its ecosystem is composed of architectural and design patterns that provide domain adaptability to Object-Oriented based systems. As patterns, they've been recurrently seen in the wild and systematically documented. However, we may argue there isn't enough scientific evidence of any specific benefits due to the lack of rigorous empirical validation. In this section, we raise several research questions about the benefits of AOMs, argue what metrics should be used to support common claims, point to what should be the baseline for such experiments, and underline the need to design them as repeatable packages for independent validation.

A. Epistemology

In order to understand the way software engineers build and maintain complex and evolving software systems, research needs to focus beyond the tools and methodologies. Researchers need to delve into the social and their surrounding cognitive processes vis-a-vis individuals, teams, and organizations. Therefore, research in software engineering may be regarded as inherently coupled with human activity, where the value of generated knowledge is directly dependent on the methods by which it was obtained.

Because the application of reductionism to assess the practice of software engineering, particularly in field research, is very complex (if not unsuitable), we claim that further research should be aligned with a pragmatistic view of truth, valuing acquired practical knowledge. That is, it should be used whatever methods will seem more appropriate to prove — or at least improve our knowledge about — the questions here raised, but mostly through the use of mixed methods, such as (i) (Quasi-)Experiments to primarily assess exploratory questions, which are suitable for an academic environment, and (ii) industrial Case-Studies, as both a conduit to harvest practical requirements, as to provide a tight feedback and application over the conducted investigation.

B. Fundamental Challenges

There are some fundamental questions directly inherited from the current research trends and challenges in the area of Model-Driven Engineering (MDE). A recent research roadmap by France et al. [2] states three driving issues: (i) what forms should runtime models take; (ii) how can the fidelity of the models be maintained; and (iii) what role should the models play in validation of the runtime behavior? Another survey by Teppola et al. [56] synthesize several obstacles related to wide adoption of MDE:

1) Understanding and managing the interrelations among artifacts: Multiple artifacts such as models, code and documentation, as well as multiple types of the same artifact (e.g., class, activity, state diagrams) are often used to represent different views or different levels of abstraction. Subsets of these overlap in the sense that they represent the same concepts. Often because they are manually created and maintained without any kind of connection, consistency poses a problem.

2) Evolving, comparing and merging different versions of models: The tools we currently have to visualize differences among code artifacts are suitable because they essentially deal with text. Models often don't have a textual representation, and when they do, it may not be the most appropriate to understand the differences and to make decisions, particularly if these are to be carried by the end-user.

3) Model transformations and causal connections: Models are often used to either (i) reflect a particular system, or (ii) dictate the system's behavior. The relationships between the system and its model, or between different models that represent different views of the same system, are called causal connections. Maintaining their consistency when artifacts evolve is a complex issue, often carried manually.

4) Model-level debugging: If the model is being used to dictate a system's behavior, enough causal connections must be kept in order to understand and debug a running application at the model-level.

5) Combination of graphical and forms-based syntaxes with text views: Developers and end-users have different preferences concerning textual syntaxes and graphical editors to view and edit models. To this extend, a complete correspondence between each strategy is currently not well supported.

6) Moving complexity rather than reducing it: Model-Driven Engineering is not a "silver-bullet" [57] and as such its benefits must be carefully weighted in context to assess whether the approach will actually reduce complexity, or simply move it elsewhere in the development process.

7) Level of expertise required: It is not clear if the interrelationships among multiple artifacts (which may have different formalisms), combined with the necessary (multiple) levels of abstraction to express a system's behavior actually eases the task of any given stakeholder to understand the impact and carry out a particular change, and to which extent current training in CS/SE courses is adequate.

C. Viewpoints

When researching Adaptive Object-Models, there are always two distinct viewpoints from where we can measure the benefits: (i) the developer viewpoint, which is actively trying to build a system for a specific use-case profile, and (ii) the end-user, which, when provided, will be evolving the system once delivered. The existence of an end-user as a change-agent, although always cited as a benefit of the use of AOMs, should not be taken lightly. What may seem as an excellent way to improve adaptivity to the well-trained developer, it may may reveal as an encumbrance to the end-user, or at worst, a designer's worst nightmare.

We thus suggest that some questions regarding end-user development should be: (i) either specifically researched in the area of AOMs, or (ii) borrowed from other fields of research:

1) End-user perception of the model: The way end-users see their systems is different from the abstraction the developer are used to. Understanding the differences between this two perspectives is essential to provide mechanisms in the user-interface that are suitable, and avoids an higher-level BIG BALL OF MUD.

2) Visual metaphors: We shouldn't expect the common end-user to actually type in a Domain Specific Language to express some new rules they want to insert in the system. Other kinds of visual metaphors should be considered as a proxy for the underlying rule engine. A more detailed discussion can be found in [58].

3) Evolving the model: A tentative, failed, evolution may be disastrous regarding the meaning of data. Mechanisms to recover from mistakes, though already useful to the developer, are paramount to the end-user.

D. Specific Challenges

Although the research in Adaptive Object-Models is a subset of the research in MDE, we think the following questions should be careful assessed and their answers would

contribute to the body of knowledge, particularly when choosing to use (or not) this pattern. Though the authors believe in the capability of Adaptive Object-Models to efficiently cope with several of the stated issues in software development, and this belief has been substantiated both by research on the wider area of MDE, as well as through the studies by the pattern community, we believe that we have much to gain if we could prove that an AOM, despite a pattern, is not an anti-pattern (i.e., an obvious, apparently good solution, but with unforeseen and eventually disastrous consequences):

1) Fitness for purpose: When is an AOM adequate to use? When should the use of an AOM be considered *over-engineering*. What metrics should we base our judgment for applicability?

2) Target audience: What type of developers are best suited for AOMs? Are current developers lacking in specific formation that hinders the usage and construction of AOMs? What about end-users? Are there specific profiles that could point to a more suitable audience?

3) Development speed indicator(s): What is the impact on the usage of AOMs during the several phases of the process? Do developers increase their ability to produce systems? How long is their *start-up* time?

4) Extensibility indicator(s): How easy is to extend an AOM-based system? How long does it take to HOOK a particular customization into the base architecture? Is this dependent on a specific implementation?

5) Quality indicator(s)?: What is the impact on software quality metrics when using AOMs? How does it affect Performance? How does it ensure Correctness? Is Consistency a major factor? What about the Usability of automatically-generated interfaces? How can we improve them?

E. Research Design

It is necessary to adequately define the experimental protocols which assess these claims in a rigorous and sound way. This includes a precise definition of the processes to be followed in industry case studies, as well as in the family of quasi-experiments to be performed in academic contexts. The design of experimental protocols for the industrial case studies, should attempt to cover the whole experimental process, i.e, from the requirements definition for each experiment, planning, data collection and analysis, to the results packaging. Discussion on guidelines for performing and reporting empirical studies have been recently going by the works of Shull et al. [59] and Kitchenham et al. [60]. The typical tasks and deliverables of a common experimental software engineering process can be found in [61].

F. (Quasi-)Experiments

(Quasi-)experiments conducted in an academic context should be randomized, multiple-group, comparison designs, which may be implemented as part of graduate student teams lab work. One scenario would involve splitting the students into three groups. One group would act as a baseline and use any traditional development methodology and tools to construct and evolve a particular system. The second group would be mandated to develop an AOM-based solution. The third group would have direct access to a framework which already provides a specific infrastructure to build AOM-based systems.

There should be an evaluation of the base skills for every member. For example, was their academic track the same, or did they take courses that could influence the experiments outcome? In this case, it should be taken into consideration subjects such as Software Specification, Agile Methodologies, Formal Methods, Design Patterns, Object-Oriented Programming, Model-Driven Engineering, to mention a few, which could pose a direct influence. In order to guarantee that there is no significant statistical deviation on their base skills, researchers should also use of a subset of computer science related GRE (or similar) questions prior to conducting the selection. There's also the need to ensure that all groups share common skills with respect to metamodeling, compilers, interpreters, and architectural and design patterns, the AOM and its ecosystem, thus specific training them should be taken into consideration.

A (quasi-)experiment may assess several distinct claims, which could match into different phases. For example, researchers could make (quasi-)experiments aligned with two different phases: development and maintenance. In the first phase, a Requirements Specification Report, which would include detailed user stories and UML diagrams (or similar artifacts) that could semi-formalize a particular small system (e.g., around 50 model elements), would be given to all groups. Their task would be to implement a full system using their given technique and restrictions. The time available for pursuing the implementation should be based in effort estimations made by software-engineer professionals. Here, several metrics should be collected and assessed.

The second phase would be pursued after all the systems are finished. A series of small changes (e.g., 10 model elements each) would be separately handled to each group, thus accounting for a change in 50% (this number here is being used rather arbitrarily; the rationale is that it should reflect real-world profiles of high-variable systems). The implementation of each of these series should occur in a more strict laboratory environment (compared to the first phase), with the supervision of researchers and lecturers. For each series relevant data should be collected and assessed.

In order to improve confidence in the results, there should be a repetition of this experiment where the groups would (randomly) switch positions — e.g., the group which was working with the framework-based solution would switch to the baseline approach — for a second round.

G. Data Collection

We propose the usage of metrics such as time, correctness and complexity of the produced artifacts, but it remains the specification of tools and methods to collect the data during experimentation that reveal non-intrusive for practitioners. A possible technical solution to those using the framework would be to instrument it in order collect as much significant data as possible, including all steps of model evolution. The instrumentation of Integrated Development Environments (IDE) is also a possibility.

H. Independent Validation

The independent experimental validation of claims is not as common in Software Engineering as in other, more mature sciences. Hence, we stress the need to build reusable experimental packages that support the experimental validation of each claim by independent groups. The family of (quasi-)experiments should be performed in different locations, and lead by different researchers, but based on the same experimental package, in order to enhance the ability to integrate the results obtained in each of the quasi-experiments, and allow further meta-analysis on them.

V. Conclusion and Future Work

Software development face an increasing difficulty in acquiring, inferring, capturing and formalizing requirements, particularly in domains that rely on constant adaptation of their processes to an evolving reality, and thus what support they expect from software. This type of software is said to be *incomplete by design* and thus require a *design for incompleteness* approach. Agile processes have intensified their focus on a highly iterative and incremental approach, accepting and embracing the relevance of efficiently coping with *change*.

While methodologies struggle to make the process and the development team more suitable, we are looking forward to what form should agile software take. One example of such software is the *WikiWikiWeb*, which embrace *incompleteness*, by relying on fundamental principles such as organic, overt, tolerant and observable. We conjecture about attempting the same principles in other systems, and those based on Adaptive Object-Models reveal a good candidate.

The Adaptive Object-Model and its ecosystem is composed of architectural and design patterns that provide adaptability to systems based on object-oriented domain models. AOMs, Software Product Lines, Model-Driven Engineering, and Frameworks are all solutions for a common set of problems, such as increase software reuse and reduce time-to-market. But AOMs, by leveraging the concept of adaptability outside the development team, empower end-users to introduce (confined) changes to the model during run-time, and thus control themselves the evolution of their own tool.

As patterns, they've been recurrently seen and documented. However, this fact doesn't seem enough to provide sufficient scientific evidence of their benefits, both to the developer and to the end-user. This may be due to the lack of sufficient empirical validation published upon the use of AOMs, such as detailed case-studies and (quasi-)experiments. In this work, we have raised several research questions that address the benefits of AOMs. We have argued what metrics should be used to support these claims, and we have pointed to what should be the baseline for such experiments, including designing them as repeatable packages for independent validation.

Acknowledgment

This work has been partially founded by the *Portuguese Foundation for Science and Technology* and *ParadigmaXis, S.A.*, through the doctorate scholarship grant SFRH / BDE / 33298 / 2008. We would also like to acknowledge the support of Joseph Yoder, which is currently cooperating in the supervision of the lead author's PhD in this area.

References

[1] H. S. Ferreira, A. Aguiar, and J. P. Faria, "Adaptive object-modelling: Patterns, tools and applications," *Software Engineering Advances, International Conference on*, vol. 0, pp. 530–535, 2009.

[2] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *FOSE '07: 2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–54.

[3] J. Arlow, W. Emmerich, and J. Quinn, "Literate modelling — capturing business knowledge with the uml," in *UML'98: Selected papers from the First International Workshop on The Unified Modeling Language*. London, UK: Springer-Verlag, 1999, pp. 189–199.

[4] J. Krogstie, A. L. Opdahl, and G. Sindre, Eds., *Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4495. Springer, 2007.

[5] M. Voelter, "A catalog of patterns for program generation," in *Proceedings of the Eighth European Conference on Pattern Languages of Programs*, Jun 2003.

[6] D. Riehle, S. Fraleigh, D. Bucka-Lassen, and N. Omorogbe, "The architecture of a uml virtual machine," in *OOPSLA '01: Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM, 2001, pp. 327–341.

[7] R. Garud, S. Jain, and P. Tuertscher, "Incomplete by design and designing for incompleteness," in *Organization studies as a science of design*, Marianne and G. Romme, Eds., 2007.

[8] N. Anquetil, K. M. de Oliveira, K. D. de Sousa, and M. G. Batista Dias, "Software maintenance seen as a knowledge management issue," *Inf. Softw. Technol.*, vol. 49, no. 5, pp. 515–529, 2007.

[9] L. Williams and A. Cockburn, "Guest editors' introduction: Agile software development: It's about feedback and change," *Computer*, vol. 36, pp. 39–43, 2003.

[10] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[11] B. Foote and J. Yoder, "Big ball of mud," in *Pattern Languages of Program Design*. Addison-Wesley, 1997, pp. 653–692.

[12] A. Kay, "The early history of smalltalk," *ACM SIGPLAN Notices*, Jan 1993.

[13] C. J. Neill and P. A. Laplante, "Paying down design debt with strategic refactoring," *Computer*, vol. 39, pp. 131–134, 2006.

[14] G. Jilles Van, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in *WICSA '01: Proceedings of the Working IEEE/IFIP Conference on Software Architecture*. Washington, DC, USA: IEEE Computer Society, 2001, p. 45.

[15] K. Andresen and N. Gronau, "An approach to increase adaptability in erp systems," in *Proceedings of the 2005 Information Recources Management Association International Conference*. Idea Group Publishing, May 2005, pp. 883–885.

[16] P. Meso and R. Jain, "Agile software development: Adaptive systems principles and best practices," *IS Management*, vol. 23, no. 3, pp. 19–30, 2006.

[17] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer-Verlag, 2009.

[18] W. Cunningham, "WikiWiki," 1995. [Online]. Available: http://c2.com/cgi/wiki

[19] A. Aguiar, "A minimalist approach to framework documentation," Ph.D. dissertation, Faculdade de Engenharia da Universidade do Porto, Sep. 2003.

[20] W. Cunningham, "Wiki design principles." [Online]. Available: http://c2.com/cgi/wiki$?WikiDesignPrinciples

[21] Wikipedia, "Abstraction," July 2010. [Online]. Available: http://en.wikipedia.org/wiki/Abstraction

[22] J. Spolsky, "The law of leaky abstractions," Nov 2002. [Online]. Available: http://www.joelonsoftware.com/articles/LeakyAbstractions.html

[23] R. CAMERON and M. ITO, "Grammar-based definition of metaprogramming systems," *ACM Transactions on Programming Languages and Systems*, Jan 1984.

[24] W. Cazzola, "Evaluation of object-oriented reflective models," *Proceedings of ECOOP Workshop on Reflective Object-Oriented Programming and Systems (EWROOPS'98), in 12th European Conference on Object-Oriented Programming (ECOOP'98)*, Jan 1998.

[25] L. Levy, "A metaprogramming method and its economic justification," *IEEE Transactions on Software Engineering*, Jan 1986.

[26] T. Stahl and M. Völter, "Model-driven software development: Technology, engineering, management," 2006.

[27] T. Parr and R. Quong, "ANTLR: A Predicated-LL(k) parser generator," *Journal of Software Practice and Experience,*, vol. 25, no. 7, pp. 789–810, July 1995.

[28] S. C. Johnson, "Yacc: Yet another compiler-compiler," Bell Laboratories, Tech. Rep., 1978.

[29] W. Schuster, "What's a ruby dsl and what isn't?" Jun 2007. [Online]. Available: http://www.infoq.com/news/2007/06/dsl-or-not

[30] M. P. Ward, "Language-oriented programming," *Software — Concepts and Tools*, vol. 15, no. 4, pp. 147–161, 1994.

[31] *Software product lines: practices and patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[32] R. Pawson, "Naked objects," Ph.D. dissertation, University of Dublin, Trinity College, Jun 2004.

[33] E. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, Aug 2003.

[34] OMG – MDA, "Model driven architecture." [Online]. Available: http://www.omg.org/mda/

[35] OMG – MOF, "MetaObject Facility." [Online]. Available: http://www.omg.org/mof/

[36] D. Roberts and R. Johnson, "Evolving frameworks: A pattern language for developing object-oriented frameworks," in *Proceedings of the Third Conference on Pattern Languages and Programming*, vol. 3, 1996.

[37] G. Roy, J. Kelso, and C. Standing, "Towards a visual programming environment for software development," *Proceedings on Software Engineering: Education & Practice*, Jan 1998.

[38] K. Czarnecki, "Overview of generative software development," *Unconventional Programming Paradigms (UPP)*, Jan 2004.

[39] G. Kiczales and E. Hilsdale, "Aspect-oriented programming," in *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*. New York, NY, USA: ACM, 2001, p. 313.

[40] A. Dantas, J. Yoder, P. Borba, and R. Johnson, "Using aspects to make adaptive object-models adaptable," *Research Reports on Mathematical and Computing Sciences*.

[41] K. Czarnecki and U. Eisenecker, "Generative programming: Methods, tools, and applications," p. 832, Jan 2000.

[42] J. Yoder, F. Balaguer, and R. Johnson, "Adaptive object-models for implementing business rules," *Urbana*, 2001.

[43] J. W. Yoder, F. Balaguer, and R. Johnson, "Architecture and design of adaptive object-models," *ACM SIG-PLAN Notices*, vol. 36, pp. 50–60, Dec. 2001.

[44] L. Welicki, J. W. Yoder, R. Wirfs-Brock, and R. E. Johnson, "Towards a pattern language for adaptive object models," in *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. New York, NY, USA: ACM, 2007, pp. 787–788.

[45] M. Fowler, "Analysis patterns: Reusable object models," 1997.

[46] A. Arsanjani, "Rule object: A pattern language for adaptive and scalable business rule construction," *Proceeding of PLoP*, 2000.

[47] J. Yoder, B. Foote, D. Riehle, and M. Tilman, "Metadata and active object models," *Conference on Object-Oriented Programming*, 1998.

[48] H. S. Ferreira, F. F. Correia, and A. Aguiar, "Design for an adaptive object-model framework: An overview," in *Proceedings of the 4th Workshop on Modelsrun.time, held at the ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS'09)*, October 2009.

[49] H. S. Ferreira, F. F. Correia, and L. Welicki, "Patterns for data and metadata evolution in adaptive object-models," *Proceedings of the Pattern Languages of Programs*, 2008.

[50] S. Ambler, *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. New York, NY, USA: John Wiley & Sons, Inc., 2003.

[51] L. Welicki, J. W. Yoder, and R. Wirfs-Brock, "A pattern language for adaptive object models: Part i - rendering patterns," in *PLoP 2007*, Monticello, Illinois, 2007.

[52] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Elements of reusable object-oriented software," p. 395, Jan 1995.

[53] L. Welicki, J. Yoder, and R. Wirfs-Brock, "A pattern language for adaptive object models: Part i-rendering patterns," *hillside.net*, 2007.

[54] R. Johnson and B. Woolf, "Type object," *Addison-Wesley Software Pattern Series*, Jan 1997.

[55] J. Yoder, F. Balaguer, and R. Johnson, "Architecture and design of adaptive object-models," *ACM SIGPLAN Notices*, Jan 2001.

[56] S. Teppola, P. Parviainen, and J. Takalo, "Challenges in deployment of model driven development," *Software Engineering Advances, International Conference on*, vol. 0, pp. 15–20, 2009.

[57] J. B. F.P., "No silver bullet essence and accidents of software engineering," *Computer*, vol. 20, pp. 10–19, 1987.

[58] B. A. Nardi, *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA, USA: MIT Press, 1993.

[59] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to Advanced Empirical Software Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[60] B. Kitchenham, H. Al-Khilidar, M. A. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu, "Evaluating guidelines for reporting empirical software engineering studies," *Empirical Softw. Eng.*, vol. 13, no. 1, pp. 97–121, 2008.

[61] M. Goulao and F. B. Abreu, "Modeling the experimental software engineering process," in *QUATIC '07: Proceedings of the 6th International Conference on Quality of Information and Communications Technology*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 77–90.

Goal Sketching from a Concise Business Case

Kenneth Boness
School of Systems Engineering
University of Reading,
Berkshire, RG6 6AY, UK
k.d.boness@reading.ac.uk

Rachel Harrison
Dept of Computing & Electronics
Oxford Brookes University
Wheatley, Oxford OX33 1HX, UK
rachel.harrison@brookes.ac.uk

***Abstract* - This paper describes how the business case can be characterized and used to quickly make an initial and structurally complete goal-responsibility model. This eases the task of bringing disciplined support to key decision makers in a development project in such a way that it can be instantiated quickly and thereafter support all key decisions. This process also greatly improves the understanding shared by the key decision makers and helps to identify and manage load-bearing assumptions. Recent research has revealed two interesting issues, which are highlighted in this paper.**

Keywords-goal-oriented requirements engineering; project management; agile development.

I. INTRODUCTION

This paper amplifies work originally presented in [1]. Our aim with goal sketching is to help stakeholders who have to make critical decisions in projects that develop evolving systems. We are developing goal sketching through action research to provide an agile way of maintaining a coherent representation of what is known about what the project is to do and how it is to do it. In [2] we state four key objectives to help the decision makers set and manage stakeholders' expectations and nurture shared understanding. Two of these objectives are bringing this help to bear as close as possible to the beginning of the project and ensuring that the methods can be adopted easily by project managers as well as analysts.

In [3; 4] we reported on case studies that suggest that building the stepwise goal refinement arguments that are fundamental to goal sketching can be more difficult than the simplicity of the concept would suggest. In [4] we showed that stepwise refinement of functional goals can be accelerated using simplified activity diagrams provided that due attention is paid to the environment (for example the contexts of construction, commissioning and operation) [2].

Reflecting on our own use of goal sketching in real projects and observing undergraduate students' difficulties it became clear that creating an initial goal sketch, which is useful can also be difficult. In this paper we report on new work to overcome this difficulty. This paper consolidates the material reported in [1] and extends it in the light of further experience.

The basic idea is to determine the roots of the goal sketch from the project's business case. Our new technique takes that part of the business case, which is crucial to the existence of the project and casts it into a structured format of goal oriented propositions (GOPs) that we call a *concise business case*. As our goal sketching is entirely based on GOPs and their refinement [2] the concise business case thus provides a disciplined start to the process. The goal sketch initiated in this way is immediately turned into a structurally complete goal-responsibility (G-R) model by adding suitable additional GOPs; usually with a generous quantity of assumptions. This process quickly brings load-bearing assumptions [5] and constraints to the fore thereby quickly framing a picture of what is known about the requirements and a clear understanding of the current threats jeopardizing the satisfaction of the business case. The aim is to help managers and developers recognize where the project might safely proceed, where it would be prudent to invest more resources into analysis, which assumptions should be "hedged" (mitigated against) and/or "sign-posted" (flagged as early warnings) [5].

It should be noted that there is no presumption in our technique that a single immutable business case is created at the outset. It is simply asserted that the purpose of the project is to deliver products that satisfy the business case at the time.

In standard product based planning (PBP) as espoused in [6; 7] the scope of a project is defined by the sum of its specialist products. Thus with the inclusion of 'management' products (project plans, contracts etc) all the expected contributory effort to a project can be estimated; at least in principle. However this is only true in practice when what is to be done and how it is to be done are both clear (such as the "painting by numbers projects" described in [8]). But when the situation harbors considerable uncertainties about what and how it is then said to be "in the fog" [8]. Setting realistic stakeholder expectations (including the eventual satisfaction of the business case) is then problematical and would need the investigative methods of requirements analysis to discover the *what* while technical invention may be needed to accomplish the *how*.

The methods in this paper concentrate on projects that have invention and/or discovery as prerequisites to their conclusion. In terms of the classifications in [8] they are the projects with a preponderance of "quest" (clear *what* and unclear *how*), "movie" (clear knowledge of *how* but unclear *what*) or "in the fog" (unclear about *what* and *how*). These situations are typical of, but not limited to, projects where Agile methodologies apply. In the wider project

management community they can be recognized as *soft* projects [9].

This paper proceeds with Section II putting the work into the context of related work. Section III outlines goal-responsibility (G-R) models as used in goal sketching. Section IV introduces the concise business case template in a simple form that is then expressed as a goal-sketch in Section V. Section VI introduces two refinements of the simple idea. Experiences from a pilot study with students and real-world case studies/y are reported in Section VII. Conclusions are presented in Section VIII.

II. RELATED WORK

The G-R modeling used in goal sketching has an antecedent in KAOS [10], which itself has applications in business process modeling as well as requirements analysis such as illustrated in [11], An alternative goal oriented approach I* [12]has been applied to a wide range of requirements and business process re-engineering. In [13]the authors combine I* with problem frames [14]and business processing to model business strategy with goal oriented analysis. Use-case techniques of goal oriented requirements engineering can offer considerable agility especially when applied with the breadth before depth pattern [15]. However use-cases primarily concern the functional behavior and outcome guarantees [16] and even when the 'wheel and hub' [16] is accounted for many more project concerns remain to be managed. None of these approaches has been specialized for projects and their business cases.

The methods of project management emphasize product delivery, risk and the *raison d'etre* provided by the business case. This is perhaps best exemplified in PRINCE2 and its product based planning [6]. However there appear to be no techniques to assure that the products will actually satisfy the business case; especially when the business case includes a requirement to satisfice the concerns of a complex customer community.

We accept that the above approaches offer potential rigor and precision in their specialized ways but none provide a combined model of business and technical requirements analysis that meets the agile aims of goal sketching [2]; speed and a concise capture of rationale in just enough precision for managing expectations and enriching stakeholder negotiation. Our new technique of goal sketching from the business case complements, and can be used with, the appropriate best practice requirements engineering and project management techniques.

Pursuit of alignment and shared understanding leads to complexities known as “problematical situations” in soft systems methodology (SSM) [17] Therefore soft systems thinking could be used for investigating soft projects. However this would tend to be cumbersome in the face of demand for agility. Nevertheless the attention to *weltanschauungen* (world views) and human activity systems (*holons*) in SSM [17] has informed our approach to goal sketching. Our goal sketching may be related to the nesting of *holons* in SSM but with the simplification of a focus on project appraisal as a problematical situation. We are currently exploring this relationship more deeply.

III. GOAL-RESPONSIBILITY MODELS

Goal sketching is a lightweight technique for producing goal-responsibility models. An example goal-responsibility model is shown in Figure 1. The figure and the explanation provided here is abstracted from [2].Each box in Figure 1 is a 'goal oriented proposition' (GOP). There are goals, assumptions and constraints. In this example P is satisfied by the combined soundness of A,Q and R. R is satisfied by actors 1 and 2 taking necessary joint and collaborative responsibilities. Similarly Goal Q is satisfied by C,S and T where S and T are satisfied by actors 1 and 3 respectively. C is a constraint that will be satisfied by the definition of a 'rule' for cross-cutting the responsibilities of actors. [2].

In this example P is a single root and A,C,S,T and R are the leaves of the G-R model. Note that the necessary behavior (and other qualities) that must be instantiated is described only at the leaves of the model; it is not distributed across the model.

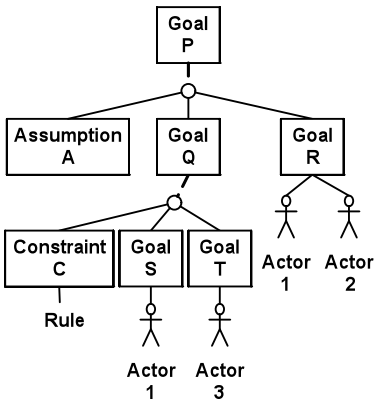


Figure 1. Goal-Responsibility model.

A *structurally complete* G-R model is one (such as Figure 1) where: all goal leaves are guaranteed by responsible actors and constraints are guaranteed through cross-cutting rules. The only leaves not guaranteed are assumptions, that must be trusted.

The skill of the analyst is to organize the GOPs into a structurally complete and persuasive stepwise argument. This discipline is a powerful aid to understanding what is known about the requirements and their preferred satisfaction. It allows the analyst to guide the setting of expectations among the stakeholders. For example in order to achieve structural completeness the analyst may need to add GOPs as “TBD” (to be determined) or to add one or more assumptions. These moves may reveal a lack of information as well as vulnerable assumptions and thus point to the risks surrounding expectations on the current understanding.

IV. THE CONCISE BUSINESS CASE (CBC)

Major project management methodologies emphasize the temporary nature of projects and how a project’s

continued existence can be justified by a viable business case; see [6; 18; 19]. Taking PRINCE2 as an exemplar, a project is defined as:-

"a temporary organization that is created for the purpose of delivering one or more business products according to a specified business case." [6].

Business products (aka specialist products [6]) define the intended outcome of the project. A product may be all or part of what Alexander calls *kit* [20] or an accomplishment such as completing the training for a group of staff who will be served by the kit. At the heart of the above definition is the imperative that these products satisfy a business case. It follows that the requirements, or acceptance criteria, for the products should be traceable to the business case. One way of assuring this is to create a G-R model such as Figure 1 to represent the business case roots, constraints and assumptions and after suitable stepwise GOP refinement placing the products that are to participate in the live system among the actors.

A business case typically includes a promissory part and a rationale justifying the investment needed to accomplish it. The promissory part will include benefits that are both direct (i.e. immediate) and indirect (realized later). The project is obliged to deliver only the direct benefits. Under the definition (above) it is the promissory part of the business case that concerns the project. Bearing this in mind and considering projects that we have observed (system & product development projects and investigation projects) we have postulated certain characteristics and summarize them in what we call the *concise business case template,* as described below.

CBC Template: Subject to the validity of certain assumptions it is agreed by the project owners that it is a sound investment proposition to realize certain direct benefits and enable other indirect benefits to the project owners through the development of products that will satisfice the concerns of a given community of 'customers'. This is to be accomplished within defined constraints on time, cost and prescribed approach.

The terms owner and customer are adopted from [17]. The term owner thus stands for those people, or their representatives, sponsoring the project so long as they can expect a satisfactory return on their investment. The term customer stands for someone (or agency) that will be a beneficiary or potential victim of the results of the project fulfilling its obligations. The community of such customers includes all the 'on-stage' and 'off-stage' actors [16] such as users and regulators.

The underlined text in the CBC template affords a basis for structuring the promissory part of the business case as a set of goal oriented propositions; the motivations (/m/), behaviors (/b/), constraints (/c/) and assumptions (/a/) described in [2]. This may be more easily visualized through a Goal Frame [2] as shown in Figure 2.

In Figure 2 the large box represents the target domain of the project, which here (and according to the CBC template) contains two sub-domains: the products to be produced and the customer community. Usually in practice both of these domains are decomposed into their own sub domains.

The underlined terms in the CBC template are represented in Figure 2 as follows: The benefits appear mostly as motivation goals at the top of the frame but there may also be motivations involved in satisficing [21] the concerns of customer community. The assumptions appear mostly as load-bearing assumptions (holding up the frame at the bottom) but there may be further assumptions involved.

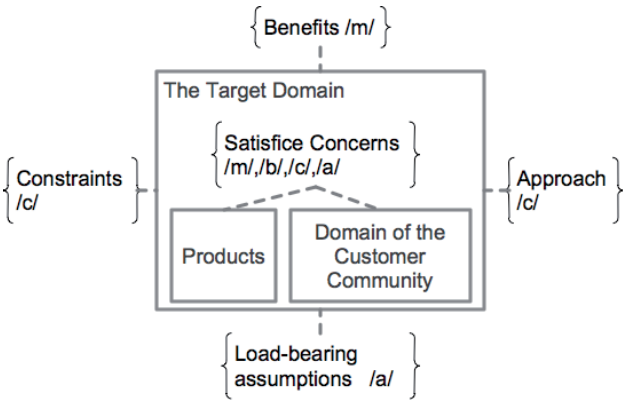


Figure 2. Concise Business Case as a Goal Frame.

The constraints and approach appear mostly as the constraints on each side of the frame (containing it) but again there may be further constraints emerging through the concerns of the customer community.

A simple illustration adapted from the zoo turnstile example in [22] serves to demonstrate the above ideas: The sponsor is the management of a zoo who believe that it is worth the investment to develop a computer-controlled turnstile guarding the entrance to their zoo. Their concerns therefore relate to an application domain involving the public and their interaction with the zoo. The GOPs in the business case could be those shown in Figure 3.

Assumptions:
- Admission to the zoo is through one gate alone. /a1/
- *Revenue is being lost by visitors evading payment. /a2/*

Benefits:
- *Increased profit for the Zoo /m1/*
- *Control of admissions /m2/*

Satisfice Customer Community:
- *Safety of the visitors (Emergency services) /m3/*
- *No additional workload (Staff)./m4/*
- *Easier reporting of visitor statistics (Staff) /m5/*

Defined Constraints:
- *The new system shall be operational by 1st April 2009. /c1/*

- *The development resources are X. /c2/*

Approach:

- *Develop a computer-controlled turnstile guarding the entrance to the zoo. /c3/*

Figure 3. The GOPs for the Zoo project.

The approach presented in Figure 3 is what we refer to as the *simple* form of the CBC. Experience has shown that certain additional concerns may need attention. These are introduced in section V following an illustration.

V. GOAL-RESPONSIBILITY MODEL FOR THE BUSINESS CASE

A structurally complete goal refinement model for the concise business case template is shown in Figure 4. Because of the lack of detail provided the completeness of the G-R model depends, as anticipated in Section III, on added assumptions and TBDs.

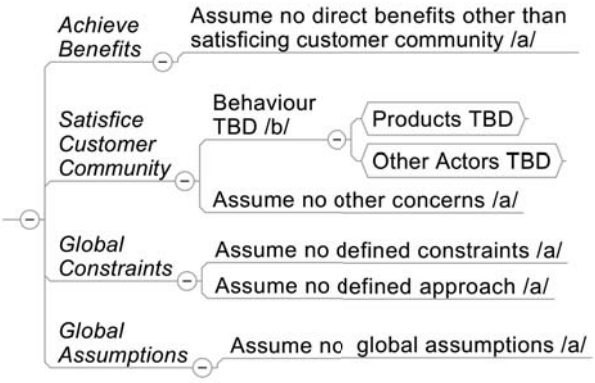


Figure 4. Structurally complete Goal-Responsibility Model for the Concise Business Case.

The goal responsibility model in Figure 4 reads from left to right. The nodes without type indicators (such as 'global constraints') are inserted as grouping nodes to make the reading easier.

In order to make Figure 4 structurally complete a set of assumptions were added to the effect that there are no other known concerns at each level of refinement over and above the concerns explicitly addressed. Such assumptions can provide a powerful challenge to the stakeholders and this helps the elicitation of technical and project requirements. A single behavior goal (/b/) has been added as a place marker and is yet to be determined (TBD) in detail. This behavior must be terminated with defined responsibilities to guarantee it.

If the project board trust the assumptions in Figure 4 and believe that its TBD can be safely resolved some time later they may judge that enough analysis has been completed; there is enough precision where it is needed and the assumptions are sound. However it would be difficult on the basis of Figure 4 as it stands to have any confidence in setting the stakeholders' realistic expectations. It is more likely that further analysis would be undertaken to validate or replace the assumptions and clarify the TBD. Completing a structurally complete G-R model with just enough detail and precision to satisfy the project board is an iterative process.

Returning to the turnstile example, the GOPs of Figure 3 are laid out in the G-R model Figure 5, which owing to space limitations is not shown in structurally complete form.

In Figure 5 the assumptions are satisfactory from a structural completeness point of view; though they are probably not persuasive.

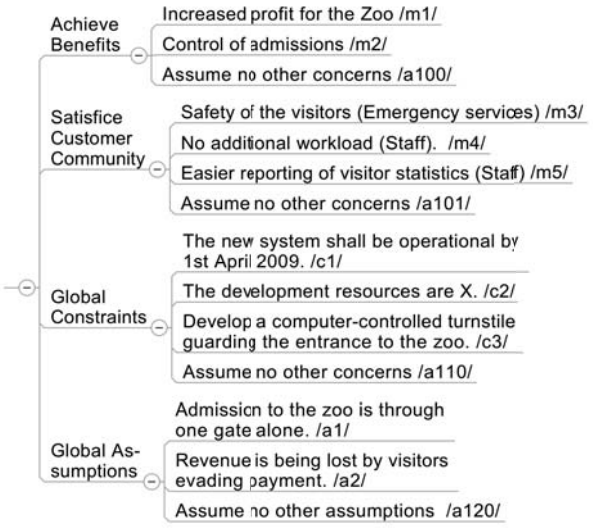


Figure 5. Structurally incomplete G-R model for the Zoo Turnstile project.

A rationale for enforcing the constraints needs to be added. They will be handled differently: /c1/ and /c2/ cross cut the project plan and impact on the feasibility of the production of project products (see [2; 23] and Cockburn's 'wheel and hub' [16]); /c3/ is a design constraint that would be testable in any products developed by the project. The motivation goals /m1/ through /m5/ will require refinement into behaviors guaranteed by appropriate actors; as in Figure 4 some of the actors will be those of the application domain and some will be the products.

As a temporary measure Figure 5 could be made structurally complete by adding TBD behaviors and suitable assumptions. As an example, a speculative first analysis is provided in Figure 6 for /m2/.

The behavior /b1/ in Figure 6 is described by a use case, which is indicated as TBD. If the project board are content that this can safely be left to the future or to chance in the hands of the developers then no more precision is needed even though the actual project products that will provide machines or props are also TBD.

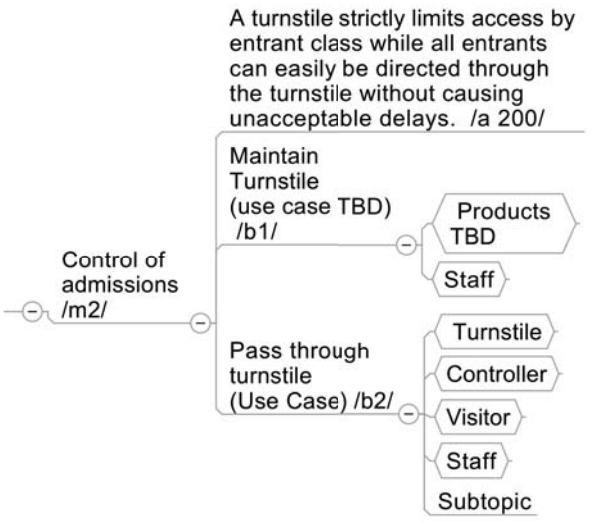


Figure 6. Structurally complete refinement to /m2/.

The goal /b2/ has definite assigned responsible actors in the form of two products (Turnstile and Controller) and people. Again if the project board is satisfied with its use case then no more analysis is needed on this matter and attention can be directed to the outstanding /m2/ through /m5/ and the composition rules for /c3/.

VI. Two Practical Issues

Practical experience has drawn our attention to two issues. The first of these relates to the context of project instantiation and execution.

A. Direct and Indirect Benefits

Turner [24] points out that (as mentioned earlier) projects are temporary organizations within a regime of project based management. Projects deliver products but the benefits sought might depend upon their subsequent exploitation (see [24] figure 1). This implies that the 'benefits' in the CBC may be indirect pending their exploitation. Thus entries listed in the CBC (such as benefits and assumptions) may need to be adjusted in order that the CBC expresses the promissory obligations on the project alone. If this observation is overlooked it can be the cause of confusion by practitioners of our method; this was found in the experience reported below.

To help circumvent the confusion we strongly advocate that the CBC is reviewed and adjusted to ensure that it expresses the project viewpoint and differentiates between direct and indirect expectations. The adjustments tend to be that those benefits that are indirect are commuted to assumptions.

B. Efficiency

The second concern pertains to the efficiency of goal sketching from the CBC.

Refining the CBC towards concrete responsibilities reveals a set of project acceptance criteria based on constraints, behaviors and assumptions. In simple examples such a refinement offers obvious 'stems' from which systems requirements can be expressed as refinement trees. For example Figure 6 takes /m2/ as a stem from which the turnstile system requirements can be obtained by refinement. This was assumed in [1] and has been assumed in all sections above.

It is more often the case that the branch from which a system refinement would logically follow will not be obvious. We have recently adjusted our technique to overcome this difficulty. Two goal graph trees are now used. The first representing the project CBC and the second representing a presumed goal of 'sound solution architecture'.

The 'sound solution architecture' goal provides a systematic opportunity to represent the full scope of the required products and the full criteria by which they would be acceptable. This point is easily overlooked. Direct refinement of the CBC can be expected to provide the ultimate acceptance criteria. However in a systems or software development project these are satisfied through the construction of a new structure. The project owners might not be able to verify the structure and it might not be their primary concern so long as the direct CBC derived acceptance criteria are satisfied. However to the project the structure is the means to satisfying the CBC goals but will itself need to be a complete and coherent structure that must be acceptance tested. It is as if the CBC presents 'black box' acceptance criteria and the second tree offers 'white box' acceptance criteria. For a project to be successful both must be satisfied. Separating problem (CBC) and solution (the solution architecture) in this way is a manifestation of the well-known fact that requirements and design so often intertwine [25; 26] as do problem and solution [27].

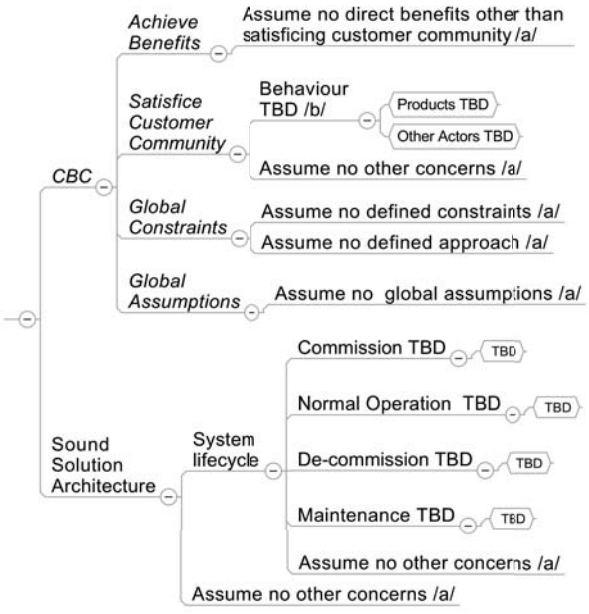


Figure 7. Showing the introduction of the sound solution architecture.

The principles are illustrated in Figure 7 where the original CBC of Figure 4 is supplemented with a 'sound solution architecture' goal.

The sound solution architecture is refined into system lifecycle concerns and an assumption of no others. In practice this assumption is unlikely to hold as there often are other parts of the solution architecture such as enterprise architecture phasing and personnel training. All of these entail products to be supplied by the project. However for the present audience the main interest will be the system lifecycle. Here we have assumed a lifecycle pattern that we often find. There is a main system with its normal operation and system actors to support that operation (all TDB at this stage as indicated by the TBD). There are also systems for commissioning and maintenance. In general there will be cross-cutting acceptance responsibilities between the actors of the different systems and between the systems and the CBC criteria.

VII. CASE STUDY

The example application is taken from a project undertaken by a small software product development company supplying tools for use in the UK medical primary care sector. The example is generalized to illustrate the fact that it has already become a reusable analysis pattern [28] to the company. This company is referred to as the supplier in the following example where company and organization names have been changed to safeguard confidentiality.

Example: A Pharmaceuticals company (PCo) wants to provide a software tool that can be installed and used in general medical practices in the UK as a supplement to their usual medical systems. The tool is intended to access and analyze the electronic records for patients registered with the medical practices who have a particular condition (the cohort of interest in each medical practice). The analysis will show compliance and deviations with nominated best practice care guidelines published by a College of Physicians (CP) and will provide data to be analyzed in a research department at the University of X (UoX) supporting the guidelines. It is a part of the business justification that this will obtain the endorsement of the National Society for the Condition (NSC). A hidden justification is that such acts of educational contribution improve the standing of the PCo among the healthcare professionals. A pharmaceutical industry regulatory body (RB) gives strict rules that the PCo must obey when interacting with the practices and the National Health Service (NHS) regulates codes of confidentiality in regard to the access and use of patients' data.

The sponsor is thus the PCo and the Customer Community includes the doctors, the CP, the Regulators (RB and NHS). The application domain comprises the medical surgeries with their staff and standard medical computer systems. The complete Customer Community Domain (see Figure 2) is shown in Figure 8. Each of the sub domains harbors people with concerns that will be satisfied by the project's products alone and/or in collaborations with actors from the sub domains.

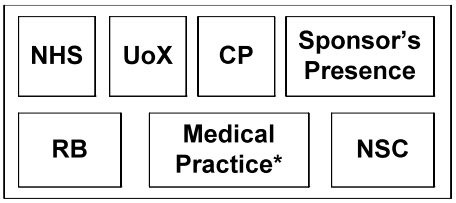


Figure 8. The Customer Community Domain for the PCo project.
*(Note: the * indicates multiplicity)*

The sponsor requires the supplier to provide the tool but, crucially, is not acquainted with the normal working activities associated with a medical practice nor with the different medical computer systems in use. The sponsor relies on the supplier for this knowledge. Thus sponsor's business case requires the satisficing of a customer community's concerns that are not appreciated by the sponsor. Figure 9 shows the CBC.

Benefits:
- *Enhance the PCo's standing appropriately with healthcare professionals /m1/*
- *Contribute to the evidence base for the Guidelines /m2/*

Satisfice Customer Community:
- *Satisfy the regulators concerns. /m3/*
- *Provide a practical service to help the doctors manage the care for their patients in the cohort of interest. /m4/*
- *Collect suitable data for onward supply to the Guidelines research centre./m5/*
- *Satisfy all brand and commercial presentation concerns /m6/*

Global Constraints:
- The tool shall be operational by 1st April 2009. /c1/
- A fixed price development fee of £X. /c2/

Approach:
- Develop an independent software tool that can be worked cooperatively with standard medical computer systems /c3/

Global Assumptions:
- *The best practice guidelines would be adopted more rigorously in the medical centres if they could be made more accessible. /a1/*
- *Support for the guidelines is note provided as a part of the normal behavior of the standard medical systems. /a2/*
- The supplier knows how to satisfice the normal working needs of the intended users /a3/

Figure 9. The GOPs for the PCo project.

An initial structurally complete G-R model was constructed from Figure 8. Six low precision TBD and assumption GOPs were needed to establish the initial structural completeness. In general the assumptions and

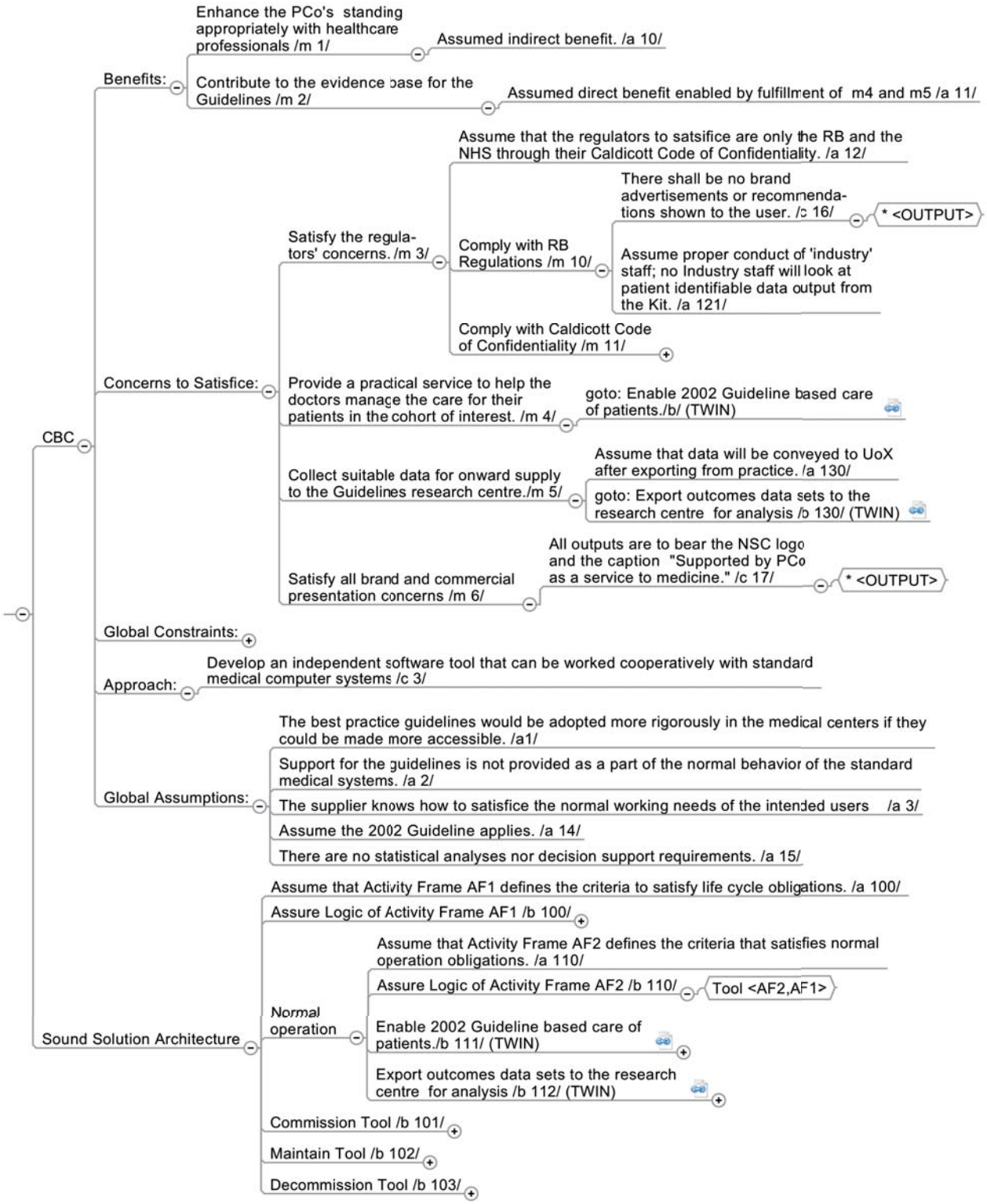


Figure 10. Partially expanded G-R model for the CBC in Figure 9.

TBDs could not be accepted by the stakeholders; however after a couple of cycles of iteration involving discussion and the goal sketching techniques outlined in [2] just enough precision was established to make proceeding on some parts of the development acceptable to the stakeholders (e.g. the refinements of /m3/,/m5/ and /m6/) whilst other parts (e.g. the refinement of /m4/) needed to be analyzed further before proceeding. The developed G-R model is shown in Figure 10.

The figure is only partially expanded because of space limitations. The (+) marked at leaves indicates hidden ('rolled-up') detail. If all of the (+) are expanded the reader would see that the graph is indeed structurally complete. All of the elements of Figure 9 are transcribed onto the graph. Benefit /m1/ is an example indirect benefit as it depends on exploitation and effect outside the competence of the project itself. On the other hand /m2/ is accepted as a direct goal, which is assumed to be satisfied as a consequence of satisfying the two customer community GOPs /m4/ and /m5/. These two GOPs imply complex requirements. These requirements require the vehicle of a coherent and structurally complete system to support their satisfaction. Such a system must be self-consistent and complete in its own terms hence it is more appropriate to apply the 'sound solution architecture' method described in section VI.II above. The /m4/ and /m5/ branches therefore point to /b111/ and /b112/ in the normal operation branch of the sound solution architecture GOP. (The link between pairs such as /m4/ and /b111/ is indicated by a hyperlink icon and the supplementary test '(TWIN)'.)

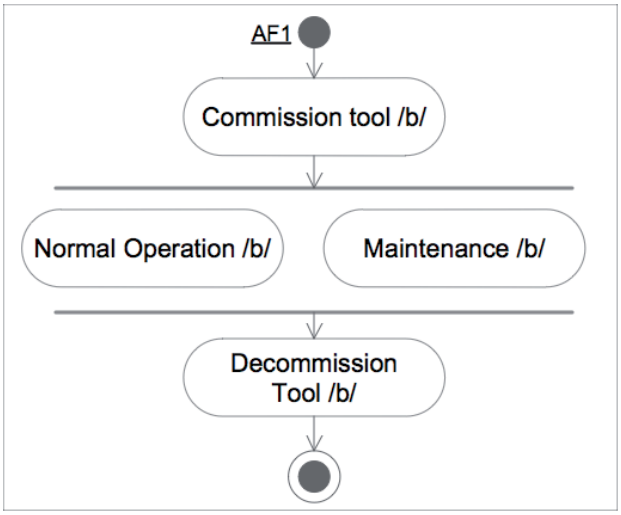


Figure 11. Activity Frame AF1 referenced in Figure 10
It depicts a life-cycle pattern.

The sound solution architecture GOP has been refined using a life-cycle refinement pattern that we frequently find appropriate. The pattern is shown in Figure 11. It has the form of an activity frame and has been applied as a refinement device in the manner we describe in [2; 4]. By applying this pattern (or an alternative) we can trust that the additional 'systems' are not overlooked.

The activity frame of Figure 11 appears in Figure 10 as AF1 in /a 100/ and /b 100/. Another, more complex activity frame has been used called AF2, which appears in /a110/ and /b111/. We refer the reader to [2; 4] for details of the technique of working with activity frames and goal refinements.

Some of the leaves of Figure 10 show operationalization by responsibility assignment. /c 16/ represents a constraint on all (signified by *) system elements that involve outputs to the screen, file or paper. The semantic tag <OUTPUT> indicates cross-cutting to all other responsibilities marked also with the tag (see [2] for details of the method). Similarly the system element 'Tool' in /b110/ will cross-cut all others with tags <AF1> and <AF2>.

This example and others have led us to the following interesting observations:-

1 Although the technique was established to accelerate goal sketching on new problems this approach has (as mentioned above) become the standard pattern used by the company on its development projects.
2 The resulting G-R models appear to focus from the start on the assumptions that are load-bearing and vulnerable [5] and this can readily lead to assumption based planning [5] with its recommended 'hedging' and 'sign-posting' tactics.
3 As anticipated when discussing Figure 4, the assumption 'no known further assumptions' and the need to impose provisional TBDs provoked keen attention to the assumptions and consequently increases the understanding shared by the stakeholders.

The effectiveness of this approach is difficult to quantify as it is uneconomic to execute a project twice concurrently; one for control and the other for comparison. Nevertheless looking at three projects (of similar size and complexity) where we used goal sketching prior to the incorporation of the CBC it can be said with some confidence that there was an appreciable acceleration brought about by the use of the CBC. Qualitatively this appears to be due to increased confidence brought about by the focus of the CBC. There were also benefits due to achieving higher that usual early shared understanding.

Pilot study: At the University of Reading each final year undergraduate of the School of Systems Engineering has an individually assigned project. 44 such students who attended an optional short module on requirements analysis were set an exercise where they would have to use a CBC. The students represented a spread of discipline from IT with management through to computer science. They were all novices at goal argumentation (such as goal graph refinement) but had a basic grasp of the principles. Most students had a year in industry and a large proportion of the students were on degrees that require a grasp of business imperatives. It was felt that this group would make an interesting test of the ease of learning and applying the

CBC technique. Despite their inexperience they are typical of the kind people who will eventually be involved in managing software and systems development projects.

In the exercise the students were instructed to prepare by writing a brief summary (100 words) of their project and then were instructed to develop a mind map of the stakeholders to the project and their respective concerns (limited to what they perceived to be the 20 most important stakeholder-concern items). They were instructed to create a CBC in their own time up to a deadline of three weeks. Again a budget as given to focus attention on expressing the full scope through the discipline of condensing what they perceived as the most important of 20 entries on the CBC. This occurred during a period that coincided with a high load of other academic work on each student.

We found that 30% (13) of the students created CBCs that were good enough to take directly to their project stakeholders for clarification and improvement by discussion and to be used as the basis for full G-R model refinement. 20% were adjudged as not understanding the technique as they needed more mentoring on the use of the technique before they could redraft their CBC and reach the standard of the higher cohort. The remainder 50% were judged as not suitable to be shown to stakeholders without prior intensive mentoring and rework.

Our technique has given us an insight that helps to distinguish those students who have the skills and ability to formulate abstract concepts about a future world. This discrimination correlated well with the more general observations of the students' tutors; people who have observed the students over several years.

The skill to formulate abstract concepts about a future world is crucial to prospective requirements analysts and it is one that industrial practitioners and students find difficult to acquire.

Looking at the CBCs produced by the upper 50% was instructive. It illustrated some weaknesses in the original formulation of the CBC, which led to the changes discussed in the above section. When using the original formulation it is easy to confuse whether a requirements statement refers to a benefit, assumption, constraint or satisfied concern. Sometimes the classification does not matter as the key outcome is the elicitation of important concerns.

Further Examples: The concise business case has been applied to other soft projects. For example a recent project between a major enterprise architecture service company and the University of Reading showed that the methods described here can be used to bring focus to a project as a whole and to stages (e.g. sprints) of the project. We can also report that in a dozen real projects considered the concise business case template proposed here in every case provides a robust and suitable template to commence a goal sketch. We have also observed its successful use in drawing up proposals and contracts.

Early indications suggest that the application of the CBC could be formulated as number of requirements analysis patterns. This matter is being investigated as further work.

VIII. Conclusion and Further Work

We have shown how goal sketching can be accelerated by introducing a template *concise business case* and have corroborated our expectation using industrial case studies. The template will be one in a family of templates. We are also confident that there is an underlying analysis pattern: choose a template, map the business case to it and transform that into a structurally complete G-R model by adding such assumptions as necessary. The pattern also appears to have a fractal nature as it can be applied to the whole project or to its stages (or agile sprints). More work is needed to clarify and document the pattern.

G-R modeling with the concise business case is most appropriate to soft projects with uncertainty about *what* and *how*. Otherwise best practice project management methods (e.g. PBP) would be advised as more cost effective.

The use of the concise business case begins a goal refinement process in which techniques such as use-case goal refinement and KAOS can be used for additional rigor with regard to operationalizing functional requirements.

It was expected that the process of building a structurally complete model from the concise business case would nurture improved shared understanding among the stakeholders. Early signs are that this is indeed the case. This is apparent in the value of the assumptions identified when attempting to build a structurally complete model from the concise business case. Additionally we identify a potential synergy with assumption based planning [5] and its 'hedging' and 'sign-posting' tactics.

Further, the elicitation of assumptions can be helped by the identification of *weltanschauungen* using soft systems methodology [17].

Creating a structurally complete model based on the concise business case might best be considered as a digest of what is known and provides a project board's viewpoint. It compliments (and does not compete with) best practice requirements engineering and project management.

Acknowledgment

The authors would like to acknowledge their industrial collaborators. In particular colleagues in OSKIS Informatics Ltd and Secerno Ltd.

REFERENCES

[1] Boness, K. and Harrison, R. 2009. Goal Sketching and the Business Case. Software Engineering Advances, 2009. ICSEA'09. The Fourth International Conference on.
[2] Boness, K., Harrison, R., and Liu, K. 2008. Goal sketching: An Agile Approach to Clarifying Requirements. International Journal on Advances in Software, IARIA. 1, 1.
[3] Boness, K. and Harrison, R. 2007. Goal sketching: Towards agile requirements engineering. Software Engineering Advances, 2007. ICSEA 2007. International Conference on. 71-71.
[4] Boness, K. and Harrison, R. 2008. Goal Sketching with Activity Diagrams. Software Engineering Advances, 2008. ICSEA'08. The Third International Conference on. 277-283.
[5] Dewar, J. A. 2002 Assumption-based planning: a tool for reducing avoidable surprises. Cambridge Univ Pr.
[6] OGC 2002. PRINCE2: Managing Successful Projects. London: HMSO.
[7] Taylor, M. D. 2003. How to Develop Work Breakdown Structures.
[8] Checkland, P. and Winter, M. 2006. Process and content: two ways of using SSM. Journal of the Operational Research Society. 57, 12, 1435-1441.
[9] Crawford, L. and Pollack, J. 2004. Hard and soft projects: a framework for analysis. International Journal of Project Management. 22, 8, 645-653.
[10] Dardenne, A., Lamsweerde, A. V., and Fickas, S. 1993. Goal-directed requirements acquisition. Science of computer Programming.
[11] Koladis, G. and Ghose, A. 2006. Relating Business Process Models to Goal-Oriented Requirements models in KAOS. Advances in Knowledge Aquisition and Management.
[12] Yu, E. 1997. Towards modelling and reasoning support for early-phase requirements engineering. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97). 226.
[13] Bleistein, S. J., Cox, K., Verner, J., and Phalp, K. T. 2006. B-SCP: A requirements analysis framework for validating strategic alignment of organizational IT based on strategy, context, and process. Information and Software Technology. 48, 9, 846-868.
[14] Jackson, M. 2000 Problem Frames: Analyzing and structuring software development problems. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
[15] Adolph, S., Cockburn, A., and Bramble, P. 2002 Patterns for effective use cases. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
[16] Cockburn, A. 2002 Writing effective use cases. Addison-Wesley Reading, MA.
[17] Checkland, P. 1998 Systems thinking, systems practice. Wiley.
[18] APM. 2006 Project management body of knowledge. Association for Project Management (APM).
[19] PMI 2000. Guide to the Project Management Body of Knowledge, PMBOK Guide 2000 edition.
[20] Alexander, I. F. 2005. A Taxonomy of Stakeholders: Human Roles in System Development. International Journal of Technology and Human Interaction. 1, 1, 23-59.
[21] Simon, H. A. 1996 The sciences of the artificial. The MIT Press.
[22] Zave, P. and Jackson, M. 1997. Four dark corners of requirements engineering. ACM Transactions on Software Engineering and Methodology (TOSEM). 6, 1, 1-30.
[23] Boness, K., Finkelstein, A., and Harrison, R. 2008. A lightweight technique for assessing risks in requirements analysis. IET Software. 2, 1, 46-57.
[24] Turner, J. R. 2008 The handbook of project-based management. McGraw-Hill Professional.
[25] Krabbel, A., Wetzel, I., and Züllighoven, H. 1997. On the inevitable intertwining of analysis and design: developing systems for complex cooperations. Proceedings of the 2nd conference on Designing interactive systems: processes, practices, methods, and techniques. 205-213.
[26] Nuseibeh, B. 2001. Weaving together requirements and architectures. IEEE Computer. 34, 2, 115-117.
[27] Hall, J. G. and Rapanotti, L. 2009. The discipline of natural design. Undisciplined! Proceedings of the Design Research Society Conference 2008. 2.
[28] Alexander, C., Ishikawa, S., and Silverstein, M. 1977 A pattern language: towns, buildings, construction. Oxford University Press, USA.

A Meta-model for Problem Frames: Conceptual Issues and Tool Building Support

P. Colombo, L. Lavazza, A. Coen-Porisini
Dipartimento di Informatica e Comunicazione
Università degli Studi dell'Insubria
Varese, Italy
{pietro.colombo, luigi.lavazza,
alberto.coenporisini}@uninsubria.it

V. del Bianco
Systems Research Group, CASL
University College Dublin
Dublin, Ireland
vieri.delbianco@ucd.ie

***Abstract*—Problem frames are an approach to requirements modeling that is gaining increasing attention and popularity. The approach provides useful concepts and methodological guidelines. However, problem frames are not equipped with an expressive and complete notation and they lack tools support. These limitations can be addressed by introducing a suitable meta-model to formally define the notation. In this way it is also possible to identify the aspects that are not covered by the problem frames notation and to provide hooks for user-defined extensions. The meta-model is expected to support the underlying analysis methodology, and the following design and verification phases. Furthermore, it can provide the basis for building a tool supporting both the editing of problem frames and the other activities associated with the approach (frame concern, composition, correctness argument, etc.). This paper presents a meta-model that addresses the former issues and was used for building a tool with the EMF/GMF technology.**

Keywords- meta-modeling; modeling tools; problem frames.

I. INTRODUCTION

Problem Frames (PFs) [1] are an approach to requirements analysis and modeling that drives the analyst to model the problem in terms of (physical) problem domains, their properties, the information they exchange and the user requirements. The solution of the problem is specified in terms of a machine, whose behavior is defined so that the interaction of the machine with the given environment satisfies the requirements.

PFs allow analysts to analyze complex problems by decomposing them into simpler ones; these basic problems are modeled according to basic patterns (i.e., the frames, which represent common, well understood problems). Then the analyst can show that the user requirements are satisfied by the outcome of the previously defined modeling activity; finally, the various problem frames are composed into a complete description.

While a great effort has been dedicated to define the PF methodology, little attention was given to the definition of an expressive and complete notation and to tools supporting PFs. For instance, PFs do not provide any language for describing the properties of problem domains, or for specifying the desired behavior of the system: the analyst has to select and use a language among the available ones (in [1] Jackson uses state-charts, pseudo-code, and natural language). Problem Frames also lack automated support: no tool is available for defining, analyzing, or composing PFs.

The aforementioned problems can be solved with the help of a meta-model that defines precisely the Problem Frames concepts, supports the methodology, and provides the basis for the construction of tools.

An initial proposal of a meta-model for Problem Frames was presented by the authors of this paper in [21]. The usage of the proposed meta-model in the construction of a prototype tool using the meta-model in combination with EMF [13] and GMF [14] was also discussed.

The meta-model presented did not cover a very important part of the Problem Frame methodology, namely the correctness argument [1]. This paper is an extended version of [21]. Besides refining the material already presented in [21], here we illustrate and discuss the usage of the meta-model in describing the requirements, domain characteristics (with special reference to behavioral properties), and machine specifications. These are the ingredients for building correctness arguments, that is, for showing that the proposed machine specification satisfies the requirements in the problem domain.

A PF-based development process is introduced as well, in which PF models are exploited also in the design and verification phases.

The paper is organized as follows: Section II provides a brief introduction to Problem Frames; Section III illustrates the proposed meta-model, while Section IV describes the UML definition of the meta-model and exemplifies the usage of the meta-model in describing a problem. Section V illustrates the usage of the meta-model for expressing requirements and describing machine and problem domain behavior, and the support to correctness arguments. Section VI describes the construction of a tool based on the meta-model, exploiting the EMF/GMF methodology. In Section VII a PF-based development process is introduced; Section VIII accounts for related work; finally Section IX draws some conclusions.

II. PROBLEM FRAMES

Problem Frames are based on the concept that user requirements are about relationships in the real world and not about functions that the software system must perform. The

desired relationships in the real world are achieved with the help of a machine; however, in the requirements analysis phase, the Machine is only specified as far as its role in the real world is concerned: only the interface between the machine and the problem domain needs to be specified, while the machine internals are left unspecified, since they will be addressed in the design phase.

Thus, the first task is to understand and represent the context in which the problem is set: the context diagram shows the various problem domains in the application environment along with their connections, and the Machine and its connections to (some of) the problem domains. A domain is simply a part of the world that we are interested in. It consists of phenomena such as individuals, events, states, relationships, and behaviors. An interface is a place where domains overlap, so that the phenomena in the interface are shared, thus allowing connection and communication between domains. A set of shared phenomena is controlled by a domain and is observed by other domains.

Problem diagrams add requirements to context diagrams. Requirements are attached to domains and specify conditions involving the phenomena of those domains (possibly including the private, non-shared ones).

An interface that connects a problem domain to the Machine is called a specification interface. The goal of the analyst is to develop a specification of the behavior that the Machine must exhibit at its interface in order to satisfy the user requirements. A PF is a description of a recognizable class of problems, and thus in some sense problem frames are problem patterns.

Figure 1 shows an example of a *commanded behavior* frame: "there is some part of the physical world whose behavior is to be controlled in accordance with commands issued by an operator. The problem is to build a machine that will accept the operator's commands and impose the control accordingly [1]".

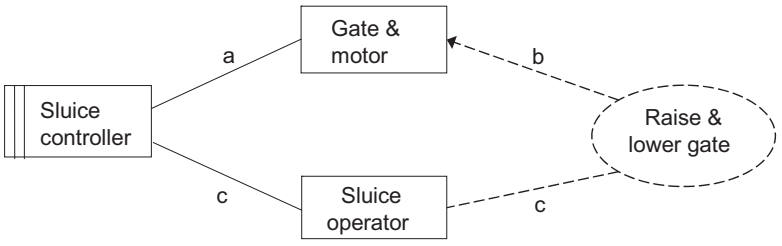


Figure 1. The sluice gate commanded behavior frame.

Such a frame (illustrated in Figure 1) is described using a simple example concerning the specification of a controller that operates a sluice gate. A small sluice, with a rising and a falling gate, is used in a simple irrigation system. A computer system is needed to raise and lower the sluice gate in response to the commands issued by an operator. The gate is opened and closed by rotating vertical screws. The screws are driven by a small motor, which can be controlled by clockwise, anticlockwise, on and off pulses. There are sensors both at the top and the bottom of the gate travel: when the top sensor is active the gate is fully open, when the bottom sensor is active, it is fully shut. The connection to the computer consists of four pulse lines for motor control, two status lines for the gate sensors, and a status line for each class of operator commands. The position of the gate is defined as the fraction of space occupied by the gate: when it is open Position=0, when it is closed Position=1. Finally, the top and the bottom sensors are active when Position becomes less than 0.05 and greater than 0.95, respectively.

The PF diagrams involves three domains: the Sluice Controller, which is the machine that will be developed to satisfy the requirements; the Gate & motor, which is the domain to be controlled (it is a causal domain since its properties include predictable causal relationship among its causal phenomena); the Sluice Operator, which is a biddable domain indicating a user without a positive predictable behavior (that is, the user can issue commands but cannot be constrained to act in any way).

It has to be assured that requirements, domain and specification descriptions fit together properly. Addressing this issue (the "frame concern") must result in a 'correctness argument' showing that the proposed machine will make the requirements satisfied in the problem domain [1].

In the case of the commanded behavior frame, we have to assure that only sensible and viable commands are executed. Requirements can be expressed as effects on the problem domain caused directly by the user's commands or by other events, such as reaching the completely open or closed position. According to Jackson, these effects can be expressed in a rather straightforward way by means of state machines. Also the behavior of the problem domain can be represented by means of a state machine, showing the states of Gate & motor, and specifying the reactions to external commands, as well as the evolution in time of the domain. For instance, the behavior of the Gate & motor domain is specified by the state machine reported in Figure 2 (taken from [1]); state 5 is an 'unknown' state, which should never be reached in normal operations; in fact, the gate would probably break if entering this state were attempted.

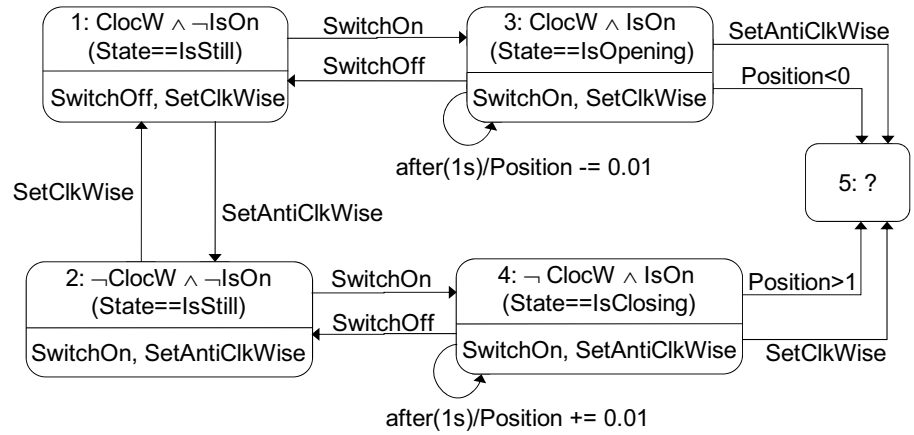


Figure 2. The specification of the Gate & motor domain.

III. THE META-MODEL

In order to build a tool that supports the PFs approach, several aspects need to be defined. Our approach consists in introducing a meta-model that supports the definition of all the aspects necessary to specify both notational and methodological concepts.

Notational concepts are used to represent the structural elements of a problem, the behavioral properties associated with such elements, the user's goals, and the machine specification. In our case the notational concepts have to support the representation of the Problem Frames diagrams as specified in [1].

Methodological elements are a collection of concepts, rules and suggestions that drive requirement analysis. For instance, phenomena that are internal to a domain are modeled, although they do not appear in problem diagrams, because they can be useful to define shared phenomena and the domains behavior.

Moreover, the definition of the meta-model allows us to identify possible inconsistencies, weaknesses or incomplete definitions in the notation, and therefore to propose solutions to address such issues. The meta-model introduces the elements needed to describe the following concepts:

- The basic structural elements and connections associated with a problem.
- The dynamic and behavioral properties associated with structural elements.
- The goals of the user, i.e., the user requirements.
- The specification of the solution, i.e., of the machine.
- The decomposition criteria.

The Problem Frames specific elements to be addressed are the following:

- A *Problem Domain* represents a physical domain of the environment where the problem is located, whose properties can be either given or designed by the user. A *Machine Domain* is a computer that interacts with the Problem domains in a way that satisfies the requirements.
- *Phenomena* are properties of a domain and can be classified as Entities, Events or States.
- *Interfaces* are connections between Domains characterized by shared phenomena.
- A *Shared phenomenon* is controlled by a domain and observed by one or more other connected domains.
- The *behavior* of a domain is specified in terms of the involved domain's phenomena. Even though the PFs methodology does not prescribe a notation for describing the behavior, the meta-model should be able to explicitly indicate the existence of a behavioral specification element and which phenomena are involved.
- *Requirements* are associated with domains; requirements are described in terms of domains' phenomena; in particular, they should be modeled as capable of referring to and constraining phenomena.
- *Machine specification*s specify the properties of the machine's interface with the problem domains.

In the next section the meta-model is described using UML [16]. This choice is motivated by the expressiveness of the language and by its diffusion in the communities of analysts and designers. Moreover, the serialization of a UML model via XMI [17] is recognized as a valid description of the meta-model by frameworks –like EMF [13]– that support the generation of tools.

IV. UML DESCRIPTION OF THE META-MODEL

A UML Class Diagram that introduces the essential features of the meta-model is shown in Figure 3.

The root element of the model is named *PFsModel*. It is composed of entities representing essential structural concepts such as *Domain* and *Interface*.

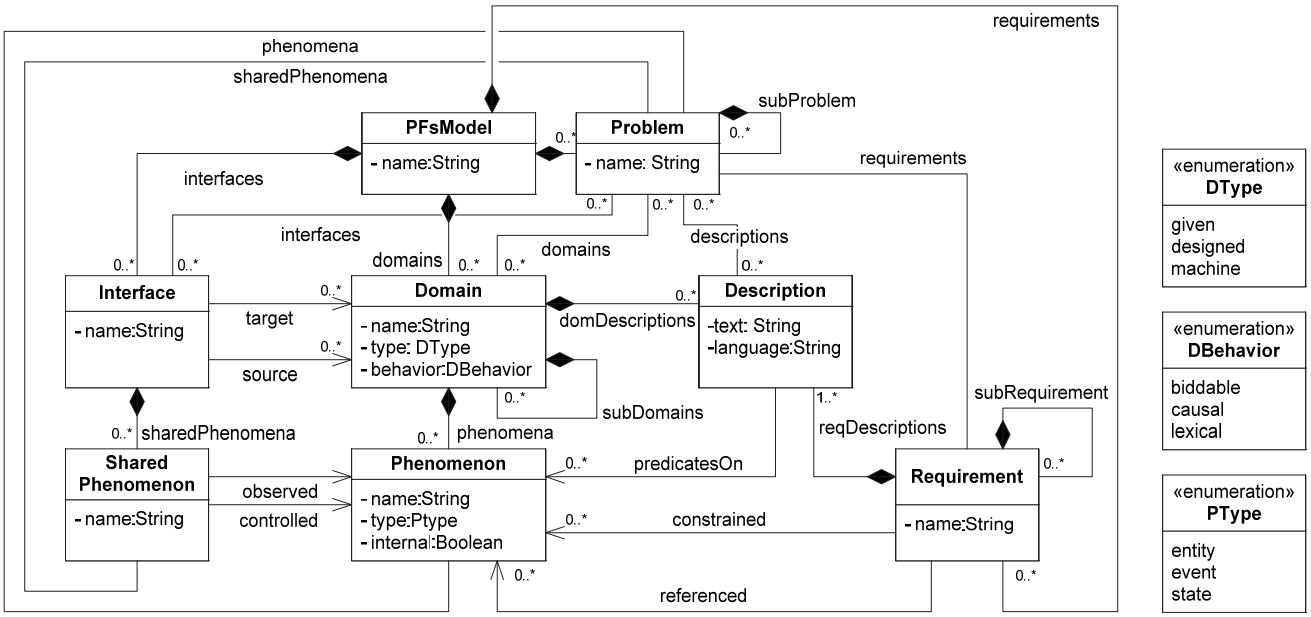


Figure 3. The UML Class Diagram defining the meta-model.

Domain is characterized by an attribute *name* (for identification purposes), an attribute *type* (to express whether the domain represents the machine, or it is given or designed), an attribute *behaviour* (to specify whether the domain is lexical, biddable or causal). The attributes *type* and *behavior* are typed by means of two enumerative data types named *DType* and *DBehavior*, respectively. Some constraints are defined on the values associated with these attributes. More specifically, a machine domain is always a causal domain, and a designed domain is never biddable. These properties are specified by means of OCL [15] constraints:

```
context Domain inv:
 ((self.type=Dtype::machine) implies
       (self.behavior=Dbehavior::causal)) and
 ((self.type=Dtype::designed) implies
       (self.behavior<>Dbehavior::biddable))
```

Another constraint imposes that domains have unique names:

```
context Domain inv:
Domain.allInstances()->forAll(p1, p2 | p1 <> p2
implies p1.name <> p2.name)
```

Similar rules are defined to assure that distinct elements of a model are given different names.

Domain is composed of sub-domains and internal phenomena. *Phenomenon* is characterized by the attributes *name* and *type*. According to Jackson, the latter is used to express whether a phenomenon is a state, an event, a value, etc. The attribute *type* is typed by means of the enumerative data type *PType*; the Boolean attribute *internal* specifies whether the phenomenon is owned and controlled or just visible by the connected domain. Also in this case some constraints are introduced. More specifically, a lexical domain cannot be characterized by causal phenomena such as events or states.

```
context Domain inv:
 self.phenomenon->forAll(p |
  self.behavior=Dbehavior::lexical implies
  (p.type<>Ptype::event and p.type<>Ptype::state))
```

Domains can be connected by means of the element *Interface*. Two directional association relationships named *source* and *target* connect the class *Interface* to the class *Domain*. Notice that the terms ‘target’ and ‘source’ do not imply that the interface has an orientation. A constraint is defined in order to assure that the involved domains are distinct:

```
context Interface inv: self.target <> self.source
```

An *Interface* exists when one or more phenomena are shared between two domains. The shared phenomenon concept is represented in the meta-model by the homonymous class. In the proposed meta-model, whenever a phenomenon (for instance, SC!Off in Figure 1) is shared, a corresponding phenomenon is created and added to the phenomena of the connected domain (in our example, a non internal phenomenon is added to the *gate&Motor* domain). An instance of *SharedPhenomenon* is also created, and connected to the instances of the corresponding phenomena (in our example, *controlled* will identify the *Off* phenomenon of *sluiceController*, while *observed* will identify the *Off* non internal phenomenon of *gate&Motor*). This rather baroque representation is motivated by the goal of using the meta-model for the development of a tool based on GMF/EMF technology. In fact, in order to guarantee that an element of a model is accessible for editing, such technology imposes that the element belongs to a containment hierarchy having the diagram being edited as root. In order to satisfy such

constraint, we identified the following solution: domains contain phenomena, and interfaces contain shared phenomena that in turn refer to the phenomena (both controlled and visible) of domains.

An additional constraint assures that the usage of the relationships *controlled* and *observed* is consistent with the value of the attribute *internal*.

```
context SharedPhenomenon inv:
 (self.controlled.internal=true and
 self.observed.internal=false and
 self.controlled.name=self.observed.name) and
 self.name=self.observed.name
```

The following constraint states that every instance of SharedPhenomenon is properly connected.

```
context SharedPhenomenon inv:
 (self.controlled.domain=self.interface.target and
 self.observed.domain=self.interface.source) or
 (self.controlled.domain=self.interface.source and
 self.observed.domain=self.interface.target)
```

The proposed solution supports the association of a different, specific editor with each element of the meta-model: the editable elements are those recursively contained in the element; the elements that are reachable from the considered element via non containment relationships can be accessed by the editor in a read-only manner. As an example, the proposed solution supports the definition of a diagram editor for domains and another editor for interfaces. With the former, internal phenomena may be added to the domain instance, which is the root element of the diagram. With the interface editor,, shared phenomena that refer to the internal phenomena of the involved domains are added to an instance of *Interface*. No internal phenomena can be added to a domain with the interface editor, being only possible to refer to existing instances.

The specification of the behavior of the domains is supported by means of the element *Description*. *Description* allows the model to be extended, i.e. several kinds of elements can be attached to this element for specifying the behavior by means of *ad hoc* notations. In fact, descriptions can be expressed with different notations such as state machines, natural language, formal languages, modeling languages like UML or SysML, etc. This can be done by importing elements from the meta-models of external notations, and by connecting them to *Description*; both the choice of which elements to import and the definition of the associations depend on the involved notation. Extending the PFs meta-model is out of the scope of this work and therefore the meta-model simply provides two attributes named *text* and *language*. The former describes the behavior by means of a textual description, while the latter indicates in which language the description is written.

Descriptions predicate on the phenomena of a domain (both controlled and visible); this concept is expressed by means of a directional association named *predicatesOn* between the classes *Description* and *Phenomenon*.

PFsModel also includes class *Requirement*, whose instances are crosscutting elements that specify static or dynamic properties with reference to the structural elements of a model. Class *Requirement* represents the user requirements, expressed by predicating on the domains' phenomena. The class is characterized by the attribute *name*, and by the relationships *constrained* and *referenced*, which express whether the requirement constrains the phenomena or just observes them. The specification of requirements is supported by the class *Description*. In fact, in the meta-model, the requirements, the machine specification and the behavior of domains are all represented by the same element *Description*.

PFsModel also introduces concepts that aim at supporting problem decomposition. More specifically, the problem concept is defined by the class *Problem*, while the decomposition is represented by the *subProblems* relationship. Other relationships are defined in order to express that a problem is characterized by requirements and domains, which are interconnected by means of interfaces. Notice that such relationships are simple associations, i.e. an instance of *Problem* is associated with instances of other elements that are contained in an instance of *PFsModel*. Such relationships support both the decomposition of a problem, and the definition of multiple views. A problem may involve only a subset of the domains (and of the corresponding phenomena) of a model: instances of *Problem* may be considered partial views on the model, consisting of subsets of the elements contained in an instance of the *PFsModel* class.

Figure 4 reports a fragment of the instance of the proposed meta-model that describes the sluice gate control problem. In particular, the model contains the Gate&Motor and SluiceController domains and their internal phenomena. Moreover, interface 'a', which connects the two domains (see Figure 1), is also shown: the interface involves a set of shared phenomena, each one corresponding to an internal phenomenon of the controlling domain and an external phenomenon, which is observed by the other domain participating in the interface. For instance, the phenomenon 'on', controlled by the SluiceController is made visible to the Gate&Motor domain through interface 'a' and the shared phenomenon 'onSP'.

The description is actually a bit redundant, with each phenomenon represented several times; however, this kind of organization was practically imposed by the constraints due to the usage of GMF.

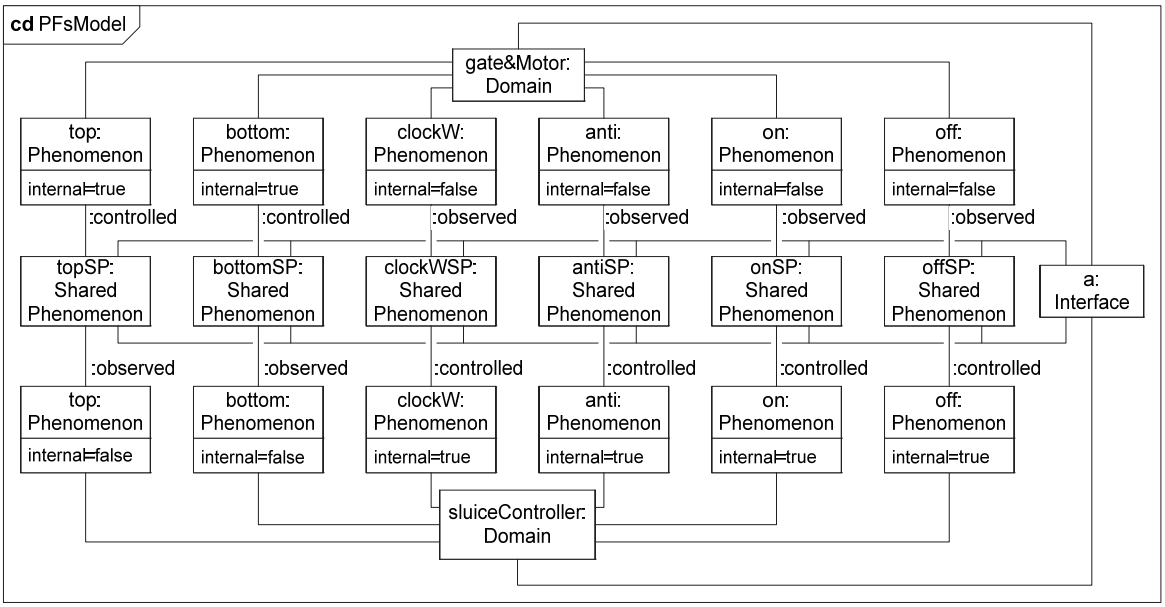


Figure 4. An instance of the meta-model (fragment).

V. REPRESENTING REQUIREMENTS AND REASONING ABOUT CORRECTNESS

A fundamental part of the problem frame methodology deals with correctness arguments.

With problem frames, the idea is that requirements are described as relations that the user wants to be established among domains in the problem environment. Requirements are therefore given by means of optative descriptions involving problem domain elements. For instance, a requirement of the sluice gate control system is that when a Raise command is issued and the state of the system makes the required operation sensible and viable, the command is executed, i.e., the gate starts rising.

Some characteristics of the relevant domains belonging to the problem environment have also to be described, because they contribute to the actual behavior of the proposed solution. For instance, the fact that the gate starts moving when the motor is set on, or that the Bottom signal is issued when the gate reaches the closed position clearly contribute to the behavior of the system. The behavior of given domains is specified by means of indicative descriptions.

The machine is the hardware/software part of the proposed solution. Its behavior is defined via suitable specifications that involve only the machine interface. The machine specification must guarantee that the interaction of the machine with the problem domain causes the required relations in the problem environment to hold.

The correctness argument must convince that the proposed machine satisfies the requirements in the problem domain.

Figure 5 illustrates a piece of the correctness (or adequacy) argument for the sluice gate control problem. Figure 5 is an adaptation from [1]. According to [1], in this kind of problem, requirements (1) state what commands are sensible in which situations and (5) what effects they should cause in the problem domain if they are viable. The specifications of the machine (2 and 3) define what is the reaction of the machine to commands (including those that are not sensible or viable). The description of the behavior of the problem domain describes how the domain state and behavior are affected by what the machine does at their shared interface.

Figure 5 provides an excerpt of the just mentioned description concerning a specific case (i.e., what happens when the Raise command is issued and the Gate is closed). The correctness argument shows the domain behavior resulting from the commands and that the final state of the system complies with the requirements, which prescribe the consequences of commands. In order to support this type of argument, the meta-model must be able to support the proper description of requirements (in terms of phenomena of the problem domains), of domain behaviors (in terms of their own phenomena and phenomena that are visible because shared by other domains, including the machine) and of machine specifications (in terms of phenomena shared through its interfaces; talking about machine's internal phenomena is strictly forbidden in this phase).

Figure 3 shows that the meta-model includes "descriptions" that belong either to domains (including the machine) or to requirements. These descriptions consist of text, written in some language, and of references to the phenomena that are mentioned in the description itself.

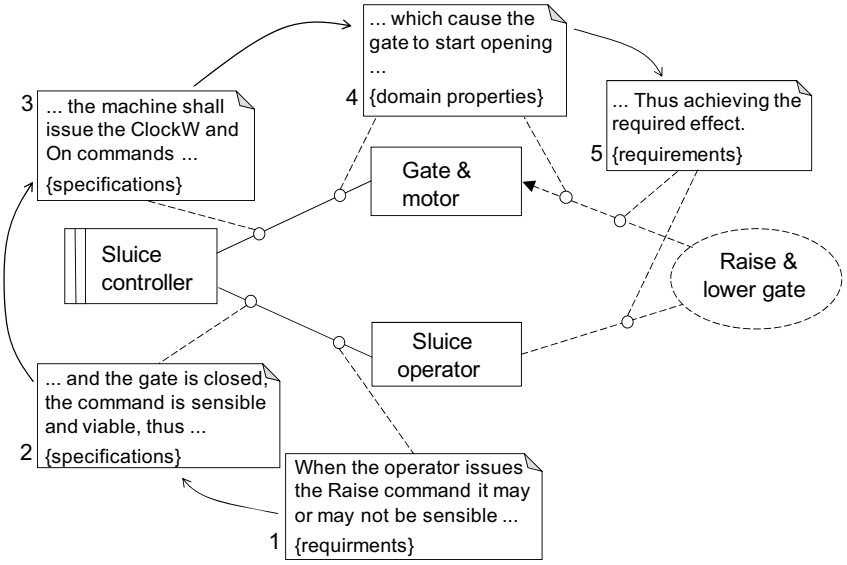


Figure 5. An adequacy argument.

Figure 6 shows a fragment of an instance of a meta-model, reporting the requirement that prescribes the effect of a Raise command when the gate is closed. In Figure 6 the textual description of the requirement is written in plain English. Therefore, it brings no meaning to a possible tool using the meta-model (unless, perhaps, sophisticated artificial intelligence techniques are used; we do not consider this possibility). However, the underlined words in the descriptions have a specific meaning, comprehensible by a tool using the meta-model: they correspond to the references to phenomena having the same names. Therefore, when the analyst that is defining the model of a system selects the phenomena that are relevant for a requirement, he/she is also determining the vocabulary that can be used in the textual description of the requirement.

Note that the analyst, when describing requirements, has to classify every phenomenon connected to a requirement as 'referenced' or 'constrained', according to the notation defined in [1].

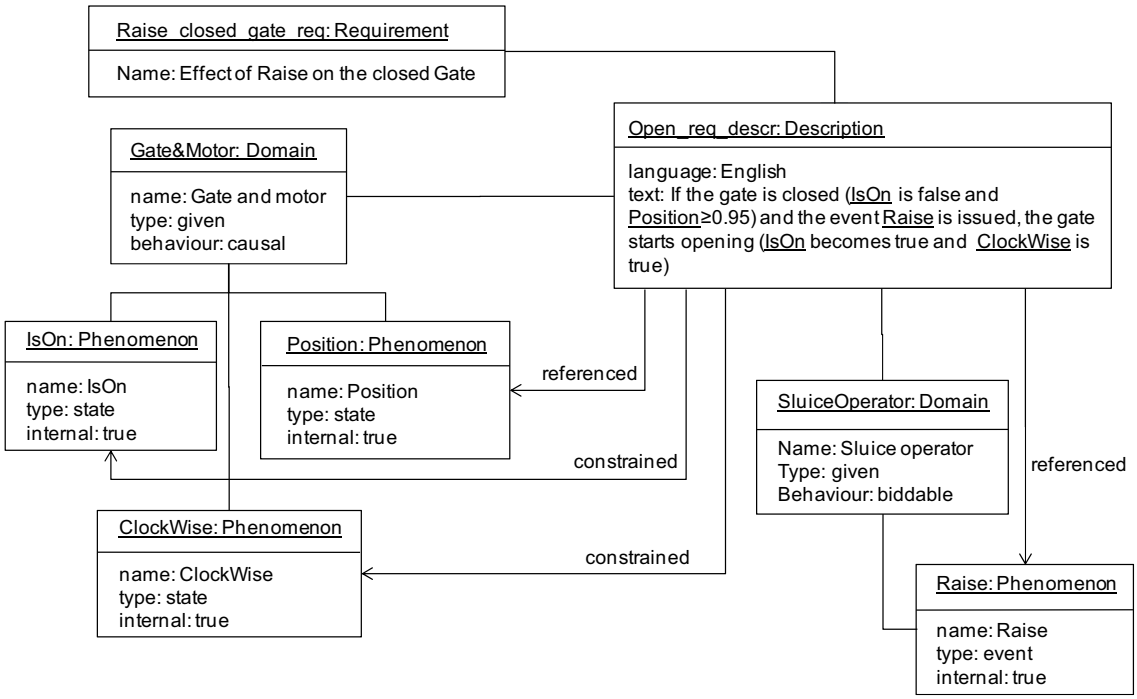


Figure 6. A fragment of meta-model instance that specifies a requirement.

Figure 7 shows a piece of the description of a domain, namely the Gate and motor. It is possible to see that this type of description works exactly like a requirement description. The only difference is that –following Jackson– we do not distinguish referenced phenomena from constrained ones. However, it must be noted that since external phenomena are always referenced, it is possible to omit their classification.

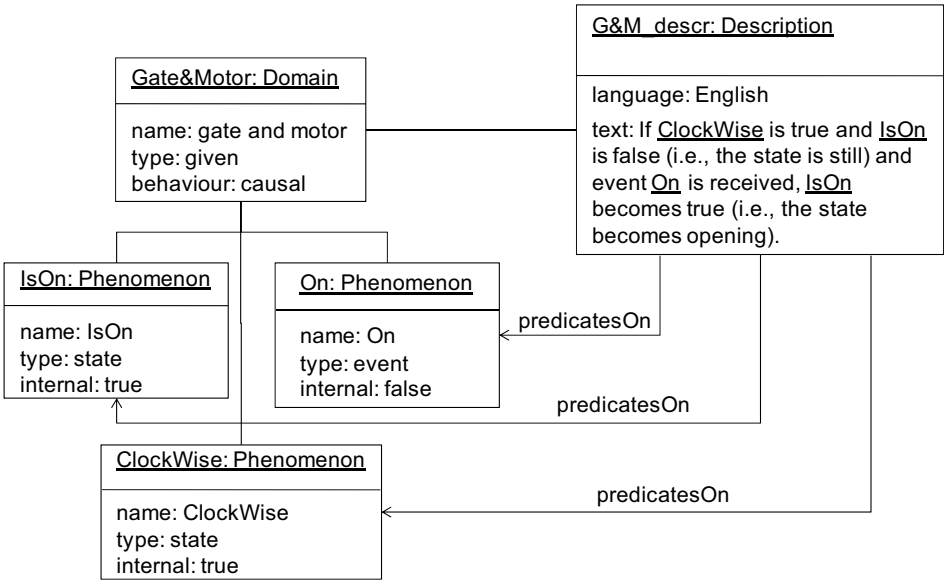


Figure 7. A fragment of meta-model instance that specifies the behaviour of the Gate&Motor given domain.

Finally, Figure 8 reports the specifications of the machine. The diagram is similar to those describing requirements and domain behavior. However, more instances of domains and phenomena are involved, since the specifications deal with phenomena from three different domains (the machine, the operator and the Gate and motor).

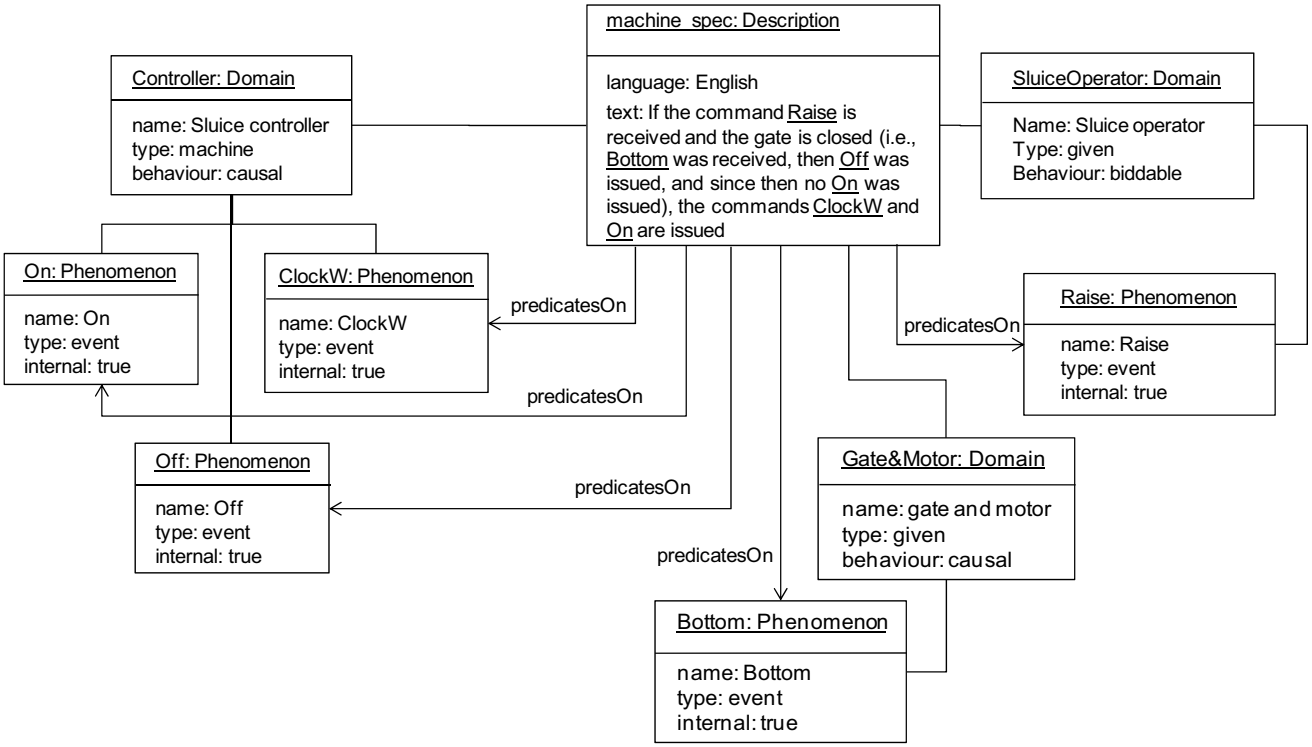


Figure 8. A fragment of meta-model instance that specifies a piece of the machine specifications.

The above reported descriptions provide the information needed to build correctness arguments. In principle, it could be possible to develop a tool that assists the user in building such arguments. In fact, the tool could help the analyst in selecting domains and phenomena that are relevant to the argument. For instance, given the Raise command, the tool could automatically select the involved requirements, the triggered machine reactions, etc., thus providing the user with the 'bricks' that can be used to build the argument.

In [1] Jackson uses a few different notations, and leaves the analysts free to choose the notation they like the most. Accordingly, the proposed meta-model allows the analyst to use any notation: it is only necessary to specify the 'language' attribute of the descriptions.

The fact that the description of domains, requirements and the machine can be used to build correctness requirements, suggests that better results could be achieved if a formal notation is used. In order to illustrate this possibility, in what follows we rewrite the descriptions already reported in Figure 6, Figure 7 and Figure 8 using event calculus (EC).

The EC is a system of logical formalism, which draws from first-order predicate calculus [24]. EC has already been used for describing and reasoning about event-based temporal systems, and has been used in conjunction with problem frames [23][22]. Of the several variations of EC that have been proposed, the version discussed in [25] was used in [22]; we also use that version.

Domain behaviour

If ClockWise is true and IsOn is false (i.e., the state is still) and event SwitchOn is received, IsOn becomes true (i.e., the state becomes opening).

$$HoldsAt(ClockWise \wedge \neg IsOn, t) \wedge Happens(On, t) \rightarrow HoldsAt(IsOn, t+1)$$

The state IsOn persists until SwitchOff is issued

$$HoldsAt(IsOn, t) \wedge \neg Happens(Off, t) \rightarrow HoldsAt(IsOn, t+1)$$

Requirements

If the gate is closed (IsOn is false and Position≥0.95) and the event Raise is issued, the gate starts opening (IsOn becomes true and ClockWise is true).

$$Holds(\neg IsOn \wedge Position \geq 0.95, t) \wedge Happens(Raise, t) \rightarrow Holds(IsOn \wedge ClockWise, t+1)$$

Machine specifications

If the command Raise is received and the gate is closed (Bottom was received, then Off was issued, and since then no On was issued), the commands ClockW and On are issued

$$Holds(Closed, t) \wedge Happens(Raise, t) \rightarrow Happens(ClockW, t+1) \wedge Happens(On, t+2)$$

$$Holds(Closed, t) \leftarrow Happens(Bottom, t1) \wedge Happens(Off, t2) \wedge t1<t2<t \wedge \neg\exists t3 (Happens(On, t3) \wedge t2<t3<t)$$

Starting from descriptions written in EC, correctness arguments can be built, also with the help of reasoning tools.

In [22] if an event or a fluent is a part of an interface, its name is parameterized –under some circumstances– with the name of the interface. For example, `Happens(e1(p),t1)` indicates that the event e1 is generated by a controlling domain at the interface p at the time t1. Similarly when describing the effect of an event on a fluent that is controlled by a domain, the fluent name is parameterized with the name of the domain. For example, `Initiates(e1(p),f2(D),t)` indicates that when the event e1 occurs at the interface p, the fluent f2 controlled by Domain D becomes true. Our meta-model is defined so that all the mentioned fluents or events correspond to specific phenomena, therefore they are unambiguously and precisely characterized in terms of the domain they belong to and the interfaces they participate into.

VI. FROM THE META-MODEL TO THE TOOL

The meta-model presented above was used as a basis for the development of a tool supporting the editing of Problem Frames as well as other aspects of the approach.

The proposed solution exploits the Eclipse Graphical Modeling Framework (GMF) [13], a "state of the art" technology for the definition of model editors in the Eclipse development framework [18]. GMF provides advanced services that guide the developer in the definition of visual editors starting from a meta-model. The generated editors also provide different kinds of advanced services such as diagram editing, validation, transformation, and support for a standardized XMI model serialization format.

GMF provides both a generative component and a runtime infrastructure for developing graphical editors based on the Eclipse Modeling Framework (EMF) [12] and the Eclipse Graphical Editing Framework (GEF) [11].

EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model. EMF consists of three fundamental parts [10]:

- The Core framework: it includes a meta-model (Ecore) for describing models and runtime support for change notification and XMI serialization.
- The Edit framework: it includes generic reusable classes for building editors for EMF models.
- The Codegen framework: it provides code generation facilities to build a complete editor for an EMF model.

EMF supports the definition of OCL constraints by providing a framework usable for property validation. EMF also provides tools for the automatic definition of basic editors that aim at visualizing and manipulating models (instances of the meta-model). GEF is a framework to be used in conjunction with GMF to create graphical editors characterized by a model-view-controller architecture. The development of diagram editors that handle EMF models based on the direct usage of GEF is an onerous activity, since it requires an in-depth knowledge of the architecture and the API of both GEF and EMF. In order to ease the development of graphical editors, the capabilities of EMF and GEF were

composed and made available through the GMF infrastructure. In fact, GMF combines the advantages of EMF and GEF, and provides tools that aim at simplifying and automating the generation of diagram editors. The usage of such technologies provides several advantages to the designer:

- A collection of reusable components for graphical editors, such as geometrical shapes, icons, etc.
- A standardized model to describe diagram elements. Diagram elements are described by means of graphical models that define both the characteristics of the visual elements shown in a diagram, and the mechanisms through which it is possible to access them.
- The separation of semantic aspects from diagrams. Semantic elements are defined in an Ecore model, and they are accessed by means of EMF, while diagram models are directly managed by GMF.

The generated editors are open, thus the interested user can access the generated source code in order to modify or extend the functionalities of the editor. Moreover, the generated editors are Eclipse plug-ins; hence extensions can exploit the standard Eclipse mechanisms.

A PF editor has to support problem analysis according to the various concepts of the PFs approach. We decided to partition the required functionalities into several editors, since the involved activities are fairly independent and use different notations. For instance, the specification of the requirements (or of the machine) uses a notation that is different from the one used for defining the problem structure.

Although functionalities are allocated to distinct editors, all the editors operate on the same model. In other words, multiple views insist on the same elements. For instance, a dedicated editor for domain behavior specification is opened whenever the user double clicks a domain instance in the problem editor. Several problems may arise when supporting diagram partitioning: editor instances have to cooperate and to stay constantly synchronized with the state of the global model.

We identified the following distinct editors: A context editor, for editing context diagrams; A problem editor, for editing problem diagram; A domain editor, for specifying the internal structure of a domain (internal phenomena as well as internal sub-domains may be defined); A domain specification editor, for describing the behavior of a domain; An interface editor, for specifying shared phenomena between the domains; A requirement editor, which supports the specification of the user requirement.

The editors were defined according to the typical GMF building process [13]. First of all, an EMF model was defined; in particular, the meta-model proposed in the Section IV –including the properties expressed via OCL– was defined via EMF Ecore technology. The EMF model describes the global model shared among the different editors. Then, a framework supporting the manipulation of the previously defined model was automatically generated using EMF.

All the previously introduced diagram editors were defined starting from the model and the generated editing code. The same GMF process was applied to the definition of each diagram editor.

A graphical model for the representation of diagram elements was defined, using the GMF graphical model creation wizard. A visual layout was defined –according to Jackson's notation [1]– for the elements of the EMF model of Problem Frames, and the access points and services to modify the attributes of each element were also defined.

The definition of the tool models for the manipulation of diagram elements exploited the GMF tool model wizard. The Problem Editor was defined so that all the elements that can be visualized (e.g., domains, requirements and interfaces) can also be edited, while the Interface Editor supports the definition of shared phenomena among domains that are given.

A mapping model specifies which graphical elements can be used in each diagram, and which tool is used for the manipulation of such elements. The definition of the mapping model was performed in part by using the GMF mapping wizard, and in part by configuring the generated model. The resulting model relates the elements of the models defined in the previous steps. Moreover, it supports editor partitioning: this model was used to specify that the interface element of the Problem editor has to be shown in the canvas of an Interface editor.

A generator model was defined for each mapping model specified for the editors by using the GMFGen Model tool. Such model supported the definition of code generation criteria, such as the specification of the serialization formats for diagrams.

Once all the generator models were defined, dedicated tools were used for automatic code generation. Then the generated source code was extended by implementing advanced functionalities, mainly concerning the partitioning of diagrams.

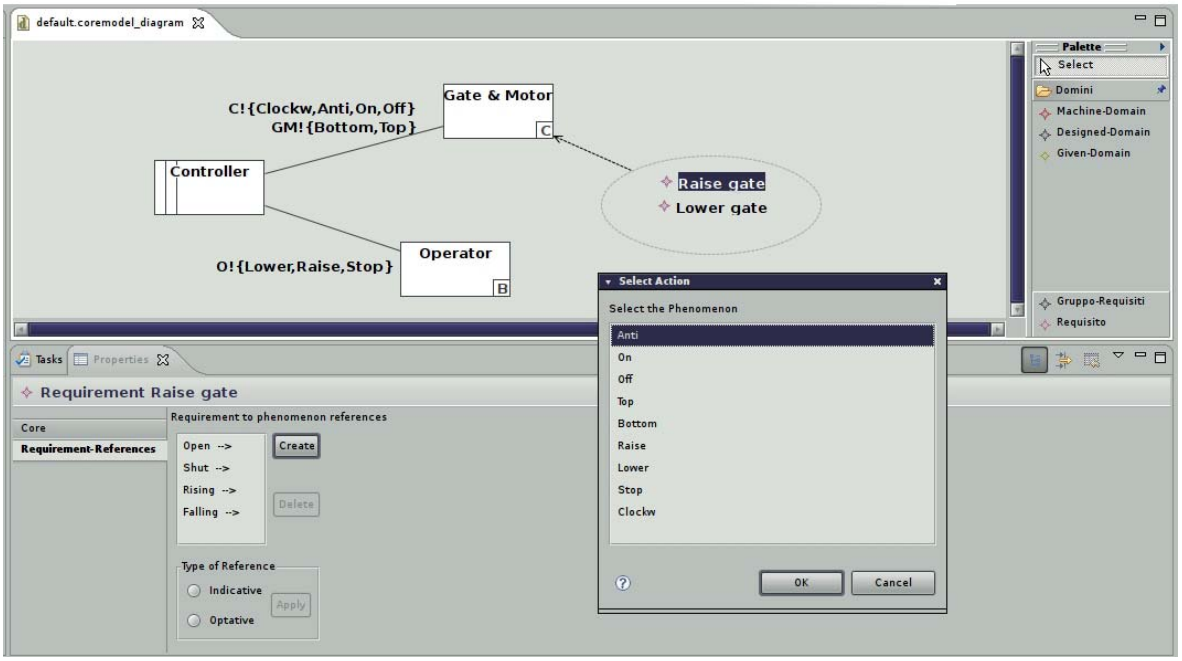


Figure 9. A snapshot of the generated tool.

Figure 9 shows a snapshot of the generated PFs editor, which is composed of the previously described editors and coordinates the activities performed by them. The current prototype is an editor that supports the PF notation. The aspects that are strictly related to the notations/languages adopted for the specifications, like developing correctness arguments, are only partly supported. Future work includes completing the support for problem composition and developing a full support for notation dependent activities.

Our experience with GMF technology was not fully satisfactory. EMF and GEF frameworks are becoming *de facto* standards for the definition of Eclipse based editors, but their combination in GMF appears not mature yet. In particular, problems arise as soon as one tries to define non-trivial editors characterized by features such as diagram partitioning, multiple views, and synchronization of different diagrams. More specifically, compilation errors, and the lack of support for a few needed functionalities, which are not properly implemented, oblige the user to manually patch the generated code and to implement the missing functionalities. Such activities are furthermore complicated by the high complexity of the structure of the automatically generated code, and by the poorly documented API of GMF.

GMF also constrains the structure of the meta-model. The worst limitation we found concerns the elements that can be edited in a diagram: they have to belong to a composition hierarchy rooted in the element associated with the editor. We addressed such issue by means of an extensive (and unusual) usage of composition relations, and by adding additional elements, which, as in the previously discussed case of the *SharedPhenomenon* class, increase the complexity of the model.

VII. A PROBLEM FRAME-BASED DEVELOPMENT PROCESS

Currently, the tool described above supports problem frame modeling only in writing syntactically correct diagrams. However, the formal descriptions of requirements and specifications could be easily exploited to verify properties of the model, in particular to prove that the specification of the machine actually satisfies (some of the) requirements when used in the modeled environment. To this end, the PF editor could be used in combination with an event calculus off-the-shelf tool, as in [23]. The resulting process (see Figure 10) would lead to reliable requirements specifications, whose most important properties would have been formally proved.

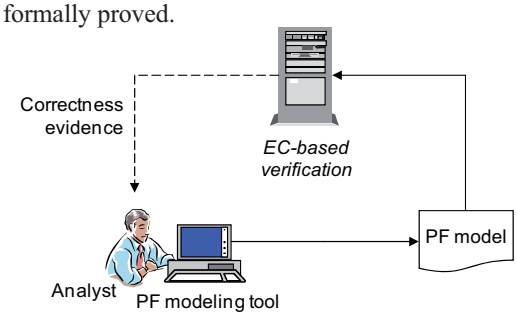


Figure 10. Problem frame editing and verification.

The process described in Figure 10 had already been envisaged by the authors for UML specifications [27], in the context of a whole UML-based development process.

The PF models that result from the modeling and verification activities are the base for the following development phases. Therefore, we need to understand to what extent the valuable information embedded in the PF

models can be exploited in the rest of the development process.

The notation used to model problem frames is not suited to support the design and implementation phases, thus we have to translate the PF diagrams into a more implementation-oriented notation. Since the authors have already shown that problem frames can be successfully used in conjunction with UML [5][26], it is quite natural to choose UML as the design notation to be used in combination with PF-based requirements specifications. The usage of UML is also eased by several EMF/UML2 based Eclipse projects (such as ATL [29] and the other transformation engines developed in the context of the Eclipse Model-to-model transformation project [28]) which support the generation of UML models from meta-model instances.

A possible problem frame-based development process is schematically described in Figure 11. The idea is to exploit to the maximum possible extent the knowledge about the environment and the machine embedded in the PF diagrams.

A first step concerns the design phase: problem frames can suggest which architectural structures are best suited for implementing the machine. In [4], Hall et al. show how each problem frame can be implemented with an appropriate design structure, while in our preceding work [6] we show how to use UML and SysML to represent PF models, thus building a starting point for the following design phase.

A PF model is often also useful for understanding the scenarios of the system (especially if complemented with UML sequence diagrams, as in [26]). Scenarios are on their turn strictly connected with testing activities: in fact, in functional testing at least one test case must be written for each scenario. A scenario involves actions, activities and events originated by both problem domains and the machine, which are described in PF diagrams: the latter can thus be used to devise test cases. Moreover, since executing test cases involves exercising some domain behavior, if the test has to be carried out in a laboratory, the problem domain behavior must be simulated: in this case, the PF diagrams provide an accurate specification of the domain behavior to be simulated. Finally, the requirements specify the expected outcome for each scenario, i.e., the oracle of the test case. In conclusion, the PF diagrams contain the whole knowledge needed to define a complete testing environment.

To summarize: PF models can be used to schematically define the software architecture, to provide domain simulators properties, and to derive functional tests cases. A tool able to understand the element constituting a PF model (based on the previously presented meta-model), could generate automatically –through model transformations– three different models: the formal model of the system, used to understand whether the systems fulfills the required proofs of correctness; the design model, used as a starting point to develop the system; the test model, used to verify the implementation.

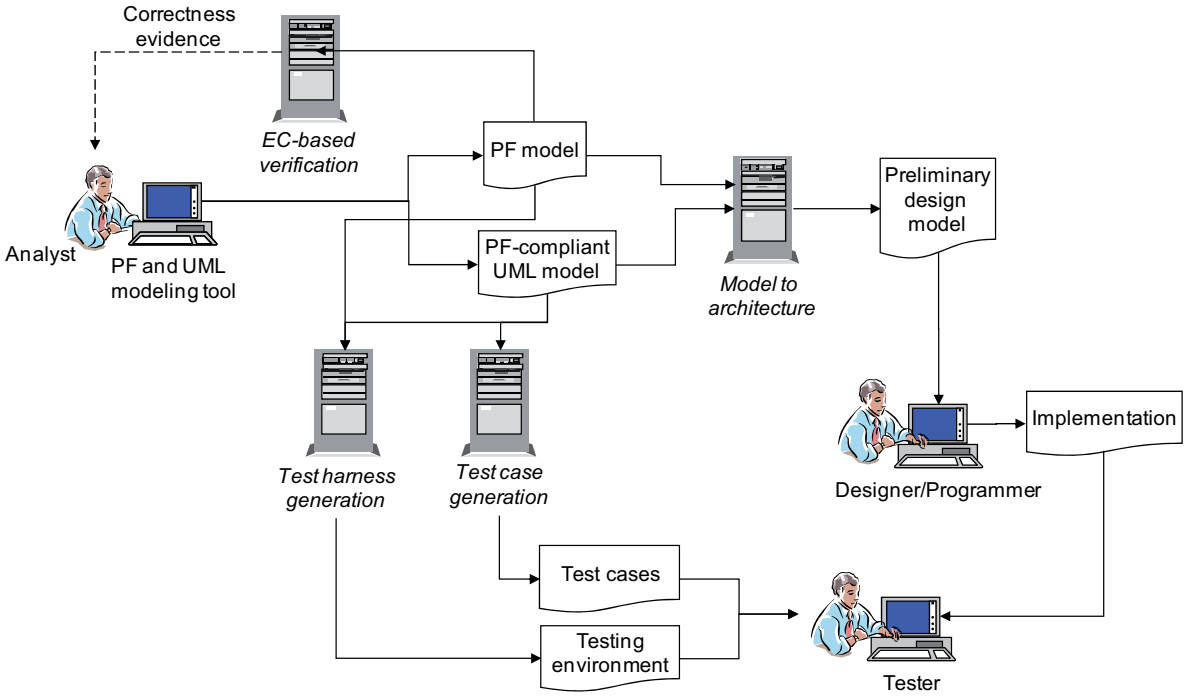


Figure 11. The PF customized development process.

VIII. Related Work

The existing PFs meta-models describe the PF domain with different objectives, which are reflected on the meta-model structure. The meta-model described in [2], [7], [8] is highly detailed, as the one presented in [9]; a less detailed meta-model can be found in [20], while a very concise meta-model is described in [10].

The meta-model presented in [2], [7], [8] describes the main relationships among most of the concepts introduced in Problem Frames methodology. This meta-model suffers from some inadequacies: some of its concepts are exclusively dedicated to represent methodological concepts, such as frame flavors and frame concerns, which we just keep out of the meta-model and out of the tool's responsibilities, leaving them to the user. Another problem (from our specific point of view) with the meta-model is the very fine granularity of the concepts presented, sometimes introducing inheritance hierarchies. Unfortunately, the management of generalization hierarchies is quite cumbersome in GMF. In practice, when working with GMF it is necessary to deal with meta-models that represent the relevant information without employing generalization/ specialization.

The ontology of the Problem Frames proposed in [3], [9] captures even more concepts than the meta-model defined in [2], [7], [8], and it is more abstract, since it does not provide any meta-model (a meta-model is always an ontology, but the vice versa is not guaranteed). From our point of view this ontology presents the same problems as the meta-model introduced in [2], [7], [8]. Moreover, it does not address the specification of behaviors and requirements, which are clearly relevant to the user.

The essential meta-model proposed in [10] is oversimplified: it does not include all the concepts that are expressed in PF diagrams, and some important pieces of information are missing. For instance, the meta-model includes a relation between domains for specifying that the involved domains overlap, but this relation does not indicate which phenomena are shared by the overlapping domains; this information is elsewhere in the model and can be retrieved in a rather complicated way. For these and other similar reasons, the meta-model presented in [10] –which, in fact, was defined to support requirements progression– is not adequate for the construction of tools.

With respect to the approaches to meta-modeling mentioned above, our approach is more pragmatic: on one hand, we strived to provide a synthetic though fairly complete description of the problem frames notation, including a few elements that –although not strictly belonging to the notation– are necessary to support the methodology of PF; on the other hand, we kept the meta-model compliant with the requirements of the EMF/GMF development method. The result was that we were able in a relatively short time to create a prototype of a tool fully supporting the problem frame notation, and well on the way of supporting the PF methodology.

In [20] a work very similar to the one reported here is described: the meta-model is expressed in UML, with added constraints in OCL, and the resulting meta-model is –quite comprehensibly– similar. However, there are also relevant differences. Our meta-model was specifically structured to be used in an EMF/GMF framework: this is reflected in the meta-model itself, e.g., composition relations and decorators are often used instead of inheritance relations. There are also some semantic differences between the two meta-models: the meta-model in [20] does not account for sub-problems and descriptions involving multiple frames. Finally, we have implemented the PF editor: a full working prototype of a PF modeling tool, able to generate UML compatible models.

Another project that exploits the GMF framework is UML2Tools: the Eclipse Ecore UML2 meta-model is used as a basis for building a tool for editing UML2 class, state, component and activity diagrams [19]. Although the goal and the approach of UML2Tools are similar to ours, it does not support a common model shared by the diagram editors (e.g., the editor of class diagrams, the editor of state diagrams, etc.): instead, each editor deals with a distinct instance of the model. In practice the user is not allowed to define a single coherent model: multiple independent diagram-specific models have to be created.

An alternative approach to directly supporting PF notation is to integrate PFs concepts and methodology in the usage of well known modeling languages, like UML and SysML. Such approach has been proposed in [5] and [6].

IX. Conclusions and Future Work

There are several reasons for defining the meta-model of Problem Frames. The first one is that the meta-model helps defining the notation in a precise way; this activity is much needed, since the Problem Frames approach provides essentially methodological guidelines and concepts, but does not precisely define the notation. A second motivation is that the meta-model supports the (semi-automatic) construction of a tool, and tool availability is an essential condition to promote the usage of Problem Frames in industrial software processes. A third motivation is that a precise model (based on a defined meta-model) can be used to automate model transformations, thus feeding other development phases, such as formal verification of the specifications (to prove that the specifications satisfy the requirements), development and test. Finally, a tool based on the meta-model provides a sort of training environment that is compliant by construction with the problem frames approach. Such environment is expected to favor the learning of the PF based requirements analysis techniques, to allow users of the PF approach to evaluate both the tool and the approach, and to stimulate the suggestion of improvements. This paper reports the definition of a meta-model for problem frames that can effectively be used as a basis for the construction of a tool. The proposed meta-model represents all the elements of the PF notation, but leaves the support of a few methodological issues to the initiative of the user. The effectiveness of the meta-model was demonstrated by building a prototype tool with GMF. This activity was also an occasion to evaluate the GMF technology, which appears still rather immature, since a few essential features (such as editing the same subset of elements in two different editors) are neither well supported nor documented.

The main goal of the work reported here was to define a meta-model that could be used as a basis for developing a tool supporting the problem frames technique. While achieving such goal, we put aside a couple of issues that will be object of future work. A first issue concerns the definition of a way to integrate *Descriptions* with the rest of the model: in essence, the issue is that the *text* attribute of *Descriptions* should be connected to the *predicateOn* links to *Phenomena*.

In other words, the occurrence of a phenomenon's name in the text of a description should be recognized as a reference to an instance of *Phenomenon*.

A second important issue involves implementing full-fledged problem composition and decomposition mechanisms, thus testing the ability of the meta-model to support this very relevant part of the problem frames method.

ACKNOWLEDGMENTS

The research presented in this paper has been partially funded by the project "Elementi metodologici per la specifica, la misura e lo sviluppo di sistemi software basati su modelli", funded by the Università degli Studi dell'Insubria.

Massimiliano Bosetti contributed to the development of the prototype tool using the proposed meta-model and approach.

REFERENCES

[1] Jackson, M.: Problem Frames - Analysing and Structuring Software Development Problems. Addison-Wesley ACM Press (2001)

[2] Lencastre, M., Boetlho, J., Clericuzzi, P., and Araújo, J.: A Meta-model for the Problem Frames Approach. In 4th Workshop in Software Modeling Engineering (WiSME'05), Montego Bay, 3 October 2005.

[3] Chen, X., Jin, Z., and Yi L.: An ontology of problem frames for guiding problem frame specification. In: 2nd International Conference-Knowledge Science, Engineering and Management, 2007.

[4] Hall, J.G., Rapanotti, L., and Jackson, M.: Problem frame semantics for software development, Software System Modeling. 4, 189-198 Springer-Verlag (2005)

[5] Lavazza, L. and del Bianco, V.: Combining problem frames and UML in the description of software requirements. In Fundamental Approaches to Software Engineering (FASE 2006), March-April 2006, Vienna.

[6] Colombo, P., Del Bianco, V., Lavazza, L., and Coen-Porisini, A.: A methodological framework for SysML: a Problem Frames-based approach. In 14th Asia-Pacific Software Engineering Conference (APSEC 2007), 5-7 Dec. 2007, Nagoya, Japan.

[7] Lencastre, M., Araujo, J., Moreira, A., and Castro, J.: Analyzing crosscutting in the problem frames approach. Proceedings of the 2006 international workshop on Advances and applications of problem frames, Shanghai, China, ACM, pp. 59--64, 2006.

[8] Lencastre, M., Araujo, J., Moreira, A., and Castro, J.: Towards aspectual problem frames: an example. Expert Systems, vol. 25, pp. 74-86, 2008.

[9] Jin, Z. and Liu, L.: Towards automatic problem decomposition: an ontology-based approach. Proceedings of the 2006 international workshop on Advances and applications of problem frames, Shanghai, China, ACM, pp. 41-48, 2006.

[10] Seater, R., Jackson, D., and Gheyi, R.: Requirement progression in problem frames: deriving specifications from requirements. Requirements Engineering, vol. 12, pp. 77-102, 2007.

[11] Moore, B., Dean, D., Gerber, A., Wagenknecht, G., and Vanderheyden, P.: Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Redbooks, 2004.

[12] Graphical Editing Framework (GEF), http://www.eclipse.org/gef [June 16, 2009]

[13] Eclipse Modeling Framework (EMF), http://www.eclipse.org/modeling/emf [June 16, 2009]

[14] Graphical Modeling Framework (GMF), http://www.eclipse.org/modeling/gmf [June 16, 2009]

[15] Object Constraint Language Specification, version 2.0, OMG formal/06-05-01, 2006

[16] OMG, UML Superstructure Specification, v. 2.1.2. formal/2007-11-02, 2007

[17] OMG, MOF 2.0/XMI Mapping, v2.1.1, formal/2007-12-01, 2007

[18] des Rivières, J. and Wiegand, J.: Eclipse: A platform for integrating development tools, IBM Systems Journal, Vol 43, No 2, 2004

[19] Model Development Tools (MDT), http://www.eclipse.org/modeling/mdt/ [June 16, 2009]

[20] D. Hatebur, M. Heisel, and H. Schmidt: A Formal Metamodel for Problem Frames. Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems, Toulouse, France: Springer-Verlag, pp. 68-82, 2008.

[21] P. Colombo, V. del Bianco, L. Lavazza, A. Coen-Porisini, "Towards a Meta-model for Problem Frames: Conceptual Issues and Tool Building Support", *The 4th Int. Conf. on Software Engineering Advances – ICSEA 2009,* September 20-25, 2009 - Porto, Portugal.

[22] Thein Than Tun, Tim Trew, Michael Jackson, Robin Laney and Bashar Nuseibeh: Specifying features of an evolving software system, Software Practice and Experience 2009; 39.

[23] Classen A, Laney R, Tun TT, Heymans P, Hubaux A.: Using the event calculus to reason about problem diagrams, Proceedings of the 3rd International Workshop on Applications and Advances of Problem Frames, Leipzig, 10 May 2008, ACM, 2008.

[24] Kowalski R, Sergot M.: A logic-based calculus of events. New Generation Computing 1986; 4(1).

[25] Miller R, Shanahan M.: The event calculus in classical logic—alternative axiomatisations. Journal of Electronic Transactions on Artificial Intelligence 1999; 3.

[26] Del Bianco, V., Lavazza, L., Enhancing Problem Frames with Scenarios and Histories in UML-based software development, Expert Systems – The Journal of Knowledge Engineering, Special issue on applications and advances in problem frames, February 2008 - Vol. 25 n. 1 Pag. 28-53 – Blackell publishing.

[27] Del Bianco, V., Lavazza, L., Mauri, M.: Model Checking UML Specifications of Real-Time Software, The Eighth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2002), Greenbelt, Maryland, 2–4 December, 2002.

[28] M2M, http://www.eclipse.org/m2m/

[29] Frédéric Jouault and Ivan Kurtev, "Transforming Models with ATL", in Satellite Events at the MoDELS 2005 Conference, Springer LNCS, Vol. 3844/2006.

Understanding Frameworks Collaboratively : Tool Requirements

Nuno Flores
Departamento de Engenharia Informática
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
e-mail: nuno.flores@fe.up.pt

Ademar Aguiar
INESC Porto, DEI
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
e-mail: ademar.aguiar@fe.up.pt

Abstract — **Software development is a social activity. Teams of developers join together to coordinate their efforts to produce software systems. This effort encompasses the development of a shared understanding surrounding multiple artifacts throughout the process. Frameworks are a powerful technique for large-scale reuse, but their complexity often makes them hard to understand and learn how to use. Developers resort to their colleagues for help and insight, at the expense of time and intrusion, as documentation is often outdated and incomplete. This paper presents a study on the state-of-the art on program comprehension, framework understanding and collaborative software environments, proposing a set of requirements for developing tools to improve the understanding of frameworks in a collaborative way.**

Keywords- Frameworks; Understanding; Collaborative; Tools; Requirements

I. INTRODUCTION

As software systems evolve in size and complexity, frameworks are becoming increasingly more important in many kinds of applications, in different technologies (object-orientation and recently aspect-orientation too), in new domains, and in different contexts: industry, academia, and single organizations.

Frameworks are a powerful technique for large-scale reuse that helps developers to improve quality and to reduce costs and time-to-market. However, before being able to reuse a framework effectively, developers have to invest considerable effort on understanding it. Especially for first time users, frameworks can become difficult to learn, mainly because its design is often very complex and hard to communicate, due to its abstractness, incompleteness, superfluous flexibility, and obscurity.

Understanding a piece of software is an important activity of software development, with a big social emphasis. Advances in global software development are leading to teams continuously becoming more and more distributed. A software development project requires people to collaborate. Trends toward distributed development, extensible IDEs, and social software influence makers of development tools to consider how to better assist the social aspects of development.

Learning how to use a framework deals with understanding its components, from its purpose and high-level architecture to its source code. Understanding framework means understanding software. The program comprehension community addresses this, in a broad sense. The social aspects of software development encompass the concerns of the collaborative software development research areas.

This paper outlines a set of requirements that should be tackled in order to develop tools to improve framework understanding using a collaborative approach.

Sections II to IV present a state-of-the art survey on the main domain areas dealt by this paper, namely Program Comprehension, Framework Understanding and Collaborative Software Development Environments. Section V points out open issues in those domains, converging to the key research questions in section VI. Section VII presents the solution approach and lists the set of requirements proposed by the authors. The paper concludes in section VIII.

These findings are part of an on-going research work [1].

II. PROGRAM COMPREHENSION

Program comprehension research can be characterized by both the theories that provide rich explanations about how programmers comprehend software as well as the tools that are used to assist in comprehension tasks.

Since the time of the first software engineering workshop [99], challenges in understanding programs became too familiar. As such, the field of program comprehension as a research discipline has evolved considerably. The main goal of the community is to build an understanding of these challenges, with the ultimate objective of developing more effective tools and methods that supports them [129].

This research has been rich and diversified, with various shifts in paradigms and research cultures during the last decades.

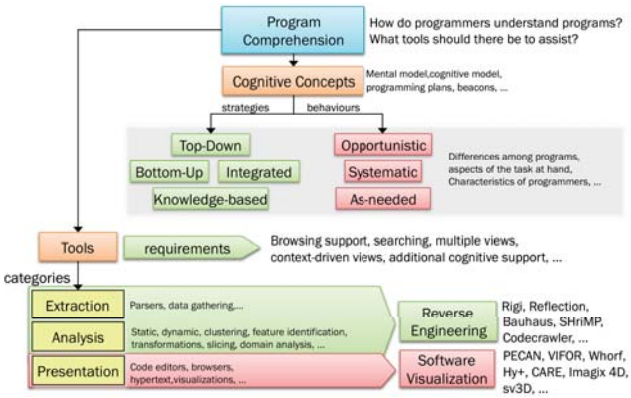


Figure 1 - Program Comprehension topics

A multitude of differences in program characteristics, programmer ability and software tasks have led to many diverse theories, research methods and tools.

Consequently, there is a wide variety of theories that provide rich explanations of how programmers understand programs and can provide advice on how program comprehension tools and methods may be improved.

In this section, an overview of existing comprehension theories, models and methods is presented, as an attempt to create a landscape of program comprehension research and possibly trends for future work directions. An overall depiction of the main topics can be seen in Figure 1.

A. Cognitive theories and models

At first, experiments were done without theoretical frameworks to guide the evaluations, and thus it was neither possible to understand nor to explain to others why one approach might be superior to other approaches [32].

As a lack of theories was being recognized as problematic, methods and theories were borrowed from other areas of research, such as text comprehension, problem solving and education. These theoretical underpinnings led to the development of cognitive theories about how programmers understand programs and ways of building supporting tools. These theories brought rich explanations of behaviors that would lead to more efficient processes and methods as well as improved education procedures [64].

1) Concepts

A **mental model** describes a developer's mental representation of the program to be understood whereas a **cognitive model** describes the cognitive processes and temporary information structures in the programmer's head that are used to form the mental model. **Cognitive support** assists cognitive tasks such as thinking or reasoning [145].

Programming plans are generic fragments of code that represent typical scenarios in programming. For example, a sorting program will contain a loop, which compares two numbers in each iteration, or visiting a structure will have a loop going through all its elements [128].

Beacons are recognizable, familiar features in the code that act as cues to the presence of certain structures [21]. **Rules of programming** discourse capture the conventions of programming, such as coding standards and algorithm implementations [128].

Then there are strategies and behaviors. Behaviors are ways of changing from one strategy to another.

2) Top-down comprehension strategy

Two main theories emerged that support a top-down comprehension strategy. Brooks [21] suggested that programmers understand a completed program in a top-down way where the comprehension process relies on reconstructing knowledge about the application domain and mapping that to the source code. The process starts with a hypothesis about the general nature of the program. This initial hypothesis is then refined in a hierarchical fashion by forming secondary hypothesis. These are then refined and evaluated in a depth-first manner, whose verification (or rejection) depends heavily on the absence or presence of beacons.

Soloway and Ehrlich [128] observed that top-down understanding is used when the code or type of code is familiar. They observed that expert programmers use beacons, programming plans and rules of programming discourse to decompose goals and plans into lower-level plans. They noted that delocalized plans complicate program comprehension.

3) Bottom-up comprehension strategy

The bottom-up theory of program comprehension proposed by Shneiderman and Mayer [119] assumes that programmers first read code statements and then mentally chunk or group these statements into higher levels abstractions. These abstractions (chunks) are aggregated further until a high-level understanding of the program is attained. The authors differentiate between syntactic and semantic knowledge of programs: syntactic knowledge is language dependent and concerns the statements and basic units in a program; semantic knowledge is language independent and is built in progressive layers until a mental model is formed, which describes the application domain.

Similarly, Pennington [102] also observed programmers using a bottom-up strategy initially gathering statement and control-flow information. These micro-structures (statements, control constructs and relationships) were chunked and cross-referenced by macro-structures (text structure abstractions) to form a program model. A subsequent situation model was formed, also bottom-up, using application-domain knowledge to produce a hierarchy of data-flow and functional abstractions (the program goal hierarchy).

4) Knowledge-based strategies

Littman et al [84] observed that programmers use either a systematic approach, reading the code in detail and tracing through control and data-flow, or they use an "as-needed approach, focusing only on the code related to the task at hand. Subjects using a systematic strategy acquired both static knowledge (information about the structure of the program) and causal knowledge (interactions between components in the program when it is executed). This enabled them to form a mental model of the program, however, those using the as-needed approach only acquired

static knowledge resulting in a weaker mental model of how the program worked. More errors occurred since the programmers failed to recognize casual interactions between components in the program.

Soloway et al. [127] combined these two theories as macro-strategies aimed at understanding the software at a more global level. In the systematic macro-strategy, the programmer traces the flow of the whole program and performs simulations as all of the code and documentation is read. However, this strategy is less feasible for large programs. In the more commonly used as-needed macro-strategy, the programmer looks at only what they think is relevant. However, more mistakes could occur since important interactions might be overlooked.

5) Integrated strategies

Von Mayrhauser and Vans [88] combined the top-down, bottom-up, and knowledge-based approaches into a single metamodel. In their experiments, they observed that some programmers frequently switched between all three strategies. They formulated an integrated metamodel where understanding is built concurrently at several levels of abstractions by freely switching between the three types of comprehension strategies.

The model consists of four major components. The first three components describe the comprehension processes used to create mental representations at various levels of abstraction and the fourth component describes the knowledge base needed to perform a comprehension process:

- The top-down (domain) model is usually invoked and developed using an as-needed strategy, when the programming language or code is familiar. It incorporates domain knowledge as a starting point for formulating hypotheses.
- The program model may be invoked when the code and application is completely unfamiliar. The program model is a control-flow abstraction.
- The situation model describes data-flow and functional abstraction in the program. It may be developed after partial program model is formed using systematic or opportunistic strategies.
- The knowledge base consists of information needed to build these three cognitive models. It represents the programmer's current knowledge and is used to store new and inferred knowledge.

6) Factors affecting comprehension strategies

The general opinion most researchers realize is that certain factors will influence the comprehension strategy adopted by a programmer [130] [140]. These factors also explain the apparently wide variation in the comprehension strategies discussed above. The variations are primarily due to:

- Differences among programs,
- Aspects of the task at hand, and
- Varied characteristics of programmers.

To evaluate how programmers understand programs, these factors must be considered [130]. These are further explored in section 2.1.1.

With experience, programmers "know" which strategy is the most effective for the given program and task. A change of strategy may be needed because of some anomaly of the program or the requested task. Program understanding tools should enhance or ease the programmer's preferred strategies, rather impose a fixed strategy may not always be suitable.

B. Program and programmers trends

Both program and programmer influence a comprehension strategy choice by their inherent and varied characteristics. Additionally, this choice also depends of the task at hand. This section debates these issues giving an insight on the subject, available studies and trends for future research.

1) Program characteristics

Programs that are carefully designed and well documented will be easier to understand change or reuse in the future. Pennington's experiments showed that the choice of language as an effect on comprehension processes [102[104]. For instance, COBOL programmers consistently fared better at answering question related to data-flow than FORTRAN programmers, while these would fare better at control-flow questions than their counterparts.

Object-oriented (OO) programs are often seen as a more natural fit to problems in real world because of "is-a" and "is-part-of" relationships in a class hierarchy and structure, but others argue that objects do not always map easily to real world problems [32]. In OO programs, abstractions are achieved through encapsulation and polymorphism. Message passing is used for communication between class methods and hence programming plans are dispersed (i.e., scattered) throughout classes.

2) Program trends

As new techniques and programming paradigms emerge and evolve, the comprehension process must shift to embrace these changes. New characteristics on both program and programming approaches seem to produce new trends for comprehension research. A few follow [129]:

a) Distributed applications.

Along with web-based applications, both are becoming more prevalent with technologies such as .NET, J2EE and web services. One programming challenge that is occurring now and is likely to increase is the combination of different paradigms in distributed applications, e.g., a client side script sends XML to a server application (which currently evolved to the AJAX [51] technology).

b) Higher levels of abstraction.

Visual composition languages for business applications are also on the increase. As the level of abstraction increases, comprehension challenges are shifting from code understanding to more abstract concepts.

c) Aspect-oriented programming.

The introduction of aspects [76] as a construct to manage scattered concerns (delocalized plans) in a program has created new excitement in the software engineering community. Aspects have been shown to be effective for managing many programming concerns, such as logging and security. However, it is not clear how aspects written by

others will improve program understanding, especially in the long term. More empirical work is needed to validate the assumed benefits of aspects.

d) Improved software engineering practices.

The more informed processes that are used for developing software today will hopefully lead to software that is easier to comprehend in the future. Component-based software systems are currently being designed using familiar design patterns [49][25] and other conventions. Future software may have traceability links to requirements and improved documentation such as formal program specifications. Also, future software may have autonomic properties, where the software self-heals and adapts as its environment changes – thus in some cases reducing time spent on maintenance.

e) Diverse sources of information.

The program comprehension community, until quite recently, mostly focused on how static and dynamic analyses of source code, in conjunction with documentation, could facilitate program comprehension. Modern software integrated development environments, such as the Eclipse Java development environment [36], NetBeans or Visual Studio [93], also manage other kinds of information such as bug tracking, test cases and version control. This information, combined with human activity information such as emails and instant messages, will be more readily available to support analysis in program comprehension. Domain information should also be more accessible due to model driven development and the semantic web.

3) Programmer individual characteristics

There are many individual characteristics that will impact how a programmer tackles a comprehension task. These differences also impact the requirements for a supporting tool. There is a huge disparity in programmer ability and creativity, which cannot be measured simply by their experience.

In her work [143], Vessey presents an exploratory study to investigate expert and novice's debugging processes. She classified programmers as expert or novice based on their ability to chunk effectively. She notes that experts used breadth-first approaches and at the same time were able to adopt a system view of the problem area, whereas novices used breadth-first and depth-first approaches but were unable to think in system terms.

Détienne [32] also notes that experts make more use of external devices as memory aids. Experts tend to reason about programs according to both functional and object-oriented relationships and consider the algorithm, whereas novices tend to focus on objects.

4) Programmer trends

As with everything else, programmers also adapt and evolve, trying to accompany the paradigm shifts and new trends in their development environment. Relevant issues are [129]:

a) Program comprehension everywhere.

The need to use computers and software intersects every walk of life. Programming, and hence program comprehension, is no longer a niche activity. Scientists and knowledge workers in many walks of life have to use and customize software to help them do science or other work. Scientists from diverse fields, such as forestry, astronomy or medical science, are using and developing sophisticated software without a formal education in computer science. Consequently, there is a need for techniques to assist in non-expert and end-user program comprehension. Fortunately, there is much work on this area (especially at conferences such as Visual Languages and the PPIG group, where they investigate how comprehension can be improved through tool support for spreadsheet and other end user applications.

b) Sophisticated users.

Currently, advanced visual interfaces are not often used in development environments. A large concern by many tool designers is that these advanced visual interfaces require complex user interactions. However, tomorrow's programmers will be more familiar with game software and other media that displays information rapidly and requires sophisticated user controls. Consequently, the next generation of users will have more skill at interpreting information presented visually and at manipulating and learning how to use complex controls.

c) Globally distributed teams.

Advances in communication technologies have enabled globally distributed collaborations in software development. Distributed open source development is having an impact on industry. Some notable examples are Linux and Eclipse. Some research has been conducted studying collaborative processes in open-source projects [94] [58] [52], but more research is needed to study how distributed collaborations impact comprehension.

C. Tools for Program Comprehension

Understanding a software program is often a difficult process because of missing, inconsistent, or even too much information. The source code often becomes the sole arbiter of how the system works. The field of program comprehension research has resulted in many diverse tools to assist in program comprehension. When developing such tools, experts bring knowledge from other fields of research as Software Visualization and Reverse Engineering as means to answer the researched requirements. This section provides insight over the studies made to improve tool development to assist on program comprehension.

1) Tool requirements studies

Which features should an ideal tool have to efficiently support program comprehension? Needless to say that these tools will only play a supporting role to other software engineer tasks, such as design, development, maintenance, and (re) documentation.

There are mainly two ways of conducting studies to discover effective features to support program comprehension: an empirical approach by observing programmers trying to understand programs and an approach based on personal experience and intuition. Given the variability in comprehension settings, both approaches contribute to answering this complex question.

As such, several studies already conducted by several authors revealed a number of tool requirements, as follows.

Biggerstaff [15] notes that one of the main difficulties in understanding comes from mapping what is in the code to the software requirements – he terms this the concept assignment problem. Although automated techniques can help locate programming concepts and features, it is challenging to automatically detect human oriented concepts. The user may need to indicate a starting point and then use slicing techniques to find a related code. It may also be possible for an intelligent agent (that has domain knowledge) to scan the code and search for candidate start points. From his research prototypes he found that queries, graphical views and hypertext were important tool features.

Von Mayrhauser and Vans [89], from their research on the Integrated Metamodel, make an explicit recommendation for tool support for reverse engineering. They determined basic information needs according to cognitive tasks and suggested the following tool capabilities to meet those needs:

- Top-down model: online documents with keyword search across documents; pruning of call tree based on specific categories; smart differencing features; history of browsed locations; and entity fan-in.
- Situation model: provide a complete list of domain sources including non-code related sources; and visual representation of major domain function.
- Program model: Pop-up declarations; online cross-reference reports and function count.

Singer and Lethbridge [123] also observed the work practices of software engineers. They explored the activities of a single engineer, a group of engineers, and considered company-wide tool usage statistics. Their study led to the requirements for a tool that was implemented and successfully adopted by the company. Specifically they suggested tool features to support "just-in-time comprehension of source-code". They noted that engineers after working on a specific part of the program quickly forget details when they move to a new location. This forces them to rediscover information at a later time. They suggest that tools need the following features to support rediscovery:

- Search capabilities so that the user can search for code artifacts by name or by pattern matching.
- Capabilities to display all relevant attributes of the items retrieved as well as relationships among items.
- Features to keep track of searches and problem-solving sessions, to support the navigation of a persistent history.

Erdös and Sneed [41] designed a tool to support maintenance following many years of experience in the maintenance and reengineering industry. They proposed that the following seven questions needed to be answered for a programmer to maintain a program that is only partially understood:

- Where is a particular subroutine/procedure invoked?
- What are the arguments and results of a function?
- How does control flow reach a particular location?
- Where is a particular variable set, used or queried?
- Where is a particular variable declared?
- Where is a particular data object accessed?
- What are the inputs and outputs of a module?

Other attempts to capture tool requirements were made that involved observation of programmers performing different tasks.

Murray and Lethbridge [98] observed software professionals using a mixed approach combining elements from specific methods used in software engineering empirical research and a sociological qualitative research called "ground theory". From this approach, they were able to develop the basis for a theory of the ways people think when explaining and comprehending software, which they called "cognitive patterns". These patterns can then be applied to further empirical observatory studies as a roadmap to capture programmer behaviour.

Zayour [150] proposes a methodology for assessing cognitive requirements and adoption success for reverse engineering tools, from which he concludes five main rules of thumb: (1) A clear and realistic definition of the problem space to be targeted is a must; (2) direct observation of the targeted user is required to form a realistic perception of users problems and tasks; (3) Tool designers should document their perception of the user's problems and tasks; (4) When determining the success of a tool, cognitive load is a more important indicator to measure than elapsed time (because it affects adoptability more) and (5) design should be aimed at satisfying cognitive requirements and thus should be guided by cognitive principles.

Work by other authors included recall tests to evaluate the ability to answer questions regarding a piece of code programmers studied for a limited period of time [102]. Subjective ratings [120] have been used recently to measure different levels of comprehension. Additionally, other studies may ask subjects to label or group different code members based on the similarity of their functionalities [113]. Soloway and Erlich [128] asked programmers to fill blank lines and complete unfinished programs on paper in an unfamiliar source code without providing specifications about the program's use or functionality. Similarly, Bertholf et al. [14] asked novice developers to complete incomplete literal programs on paper. Additional techniques to measure program comprehension involved completing incomplete call graphs, modifying existing code, report a bug, or separate source code from two different algorithms [121].

From this research and derived from cognitive theories, Storey [129] extracts and synthesizes several tool requirements:

a) Browsing support.

The top-down process requires browsing from high-level abstractions or concepts to lower-level details, taking advantage of beacons in the code; bottom-up comprehension requires following control-flow and data-flow links, both novices and experts can benefit from tools that support breadth-first and depth-first browsing; and the Integrated Metamodel suggests that switching between top-down and bottom-up browsing should be supported. Flexible browsing support also will help to offset the challenges from delocalized plans.

b) Searching.

Tool support is needed when looking for code snippets by analogy and for iterative searching. Also inquiry episodes should be supported by allowing the programmer to query on the role of a variable, function, etc.

c) Multiple views.

Programming environments should provide different ways of visualizing programs. One view could show the message call graph providing insight into the programming plans, while another view could show a representation of the classes and relationship between the to show an object-centric or data-centric view of the program. These orthogonal views, if easily accessible, can facilitate comprehension, especially when combined.

d) Context-drive views.

The size of the program and another program metrics will influence which view is the preferred one to show a programmer browsing the code for the first time. For example, in an object-oriented program, it is usually preferable to show the inheritance hierarchy as the initial view. However, if the inheritance hierarchy is flat, it may be more appropriate to show a call graph as the default view.

e) Additional cognitive support.

Experts need external devices and scratchpads to support their cognitive tasks, whereas novices need pedagogical support to help them access information about the programming language and the corresponding domain.

2) Tool development

Programming comprehension tools can be roughly grouped into three categories [139]:

- Extraction tools include parsers and data gathering tools.
- Analysis tools do static and dynamic analysis to support activities such as clustering, concept assignment, feature identification, transformations, domain analysis, slicing and metrics calculation.
- Presentation tools include code editors, browsers, hypertext and visualizations. They are strongly linked to research in software visualization.

Integrated software development and reverse engineering environments will usually have some features form each category. The set of features they support is usually determined by the purpose for the resulting tool or by the focus of the research. As such, two majors areas that relate to this issue are Software Visualization and Reverse Engineering.

a) Software Visualization

Software visualization tools and browsing tools provide information that is useful for program understanding.

These tools use graphical and textual representations for the navigation, analysis and presentation of software information to increase understanding. Mixed results have been reported through the literature on the role of text and graphics for program comprehension. While Green and Petre [53] observed that text was faster than graphics for experimental program comprehension tasks, Scanlan [117] reported an improvement using graphical visualizations when comparing textual algorithms and structured flowcharts. Petre [104] attributes the difficulty in understanding program visualizations to the fact that graphical representations have fewer navigational cues, namely secondary notations, when compared to program text: source code implies a serial inspection strategy. Moreover, she observed that experienced readers tend to use parallel textual and graphical information whenever available to assist their comprehension process: they use text as a main source to guide their understanding of graphical representation.

Several software visualization tools show animations to teach widely used algorithms and data structures [22] [125] [131]. Another class of tools shows dynamic execution of programs for debugging, profiling and for understanding run-time behavior [68] [115]. Other software visualization tools mainly focus on showing textual representations, some of which may be pretty printed to increase understanding [9] [63] or use hypertext in an effort to improve the navigability of the software [104]. Typography plays a significant role in the usefulness of these textual visualizations.

Many tools present relevant information in the form of a graph where nodes represent software objects and arcs show the relations between the objects. This method is used by PECAN [102], Rigi [96], VIFOR [107], Whorf [19], CARE [83], Hy+ [91] and Imagix 4D [67]. Other tools use additional pretty printing techniques or other diagrams to show structures or information about the software. For example, the GRASP tool uses a control structure diagram to display control constructs, control paths and the overall structure of programming units [130].

b) Reverse Engineering

Reverse Engineering concerns how to extract relevant knowledge from source code and present it in a way that facilitates comprehension. Several studies conducted in the past have proposed solutions on how to overcome caveats in the program comprehension process. Maryhauser and Vans [89], Singer and Lethbridge [123] and Zayour [150] have given their insight on how to address tool development for reverse engineering of useful information to assist on program understanding (seen on section 2.1). K.Wong also discusses reverse engineering tool features [149]. He specifically mentions the benefits of using a “notebook” to support ongoing comprehension.

Usually, the reverse engineering tools and techniques associated to program comprehension are bundled into broader development environments where other types of tools also co-exist.

It is possible to examine each of these environments and to recover the motivation for the features they provide by tracing back to the cognitive theories. For example, the Rigi system [96] has support for multiple views, cross-referencing and queries to support bottom-up comprehension. The Reflection tool [97] has support for the top-down approach through hypothesis generation and verification. The Bauhaus tool [38] has features to support clustering (identification of components) and concept analysis. The SHriMP tool [132] provides navigation support for the Integrated Metamodel, i.e, frequent switching between strategies. And the

Codecrawler tool [79] uses visualization of metrics to support understanding of an unfamiliar system and to identify bottlenecks and other architectural features.

All these tools combine reverse engineering tasks with software visualization techniques to improve program comprehension on different levels of abstraction, gathering information recovered or simply mined together into user-friendly viewed chunks of valuable data for the programmer.

3) Tool trends

The forthcoming breakthroughs in tool technology seem promising as research and evaluation methods and theories become more relevant to end-users doing programming-like tasks. Therefore, directions in tool evolution appear to follow several guidelines presented next [129].

a) Faster tool innovations.

The use of frameworks as an underlying technology for software tools is leading to faster tool innovations, as less time needs to be spent reinventing the wheel. A prime example of how frameworks can improve tool development is the Eclipse platform [36]. Eclipse was specifically designed with the goal of creating reusable components, which would be shared across different tools. The research community benefits from this approach in several ways. Firstly, they are able to spend more time writing new and innovative features as they can reuse the core underlying features offered by Eclipse and its plug-ins; and secondly, researchers can evaluate their prototypes in more ecologically valid ways as they can compare their new features against existing industrial tools.

b) Mix 'n match tools.

Given a suite of tools that all plug in to the same framework, together with a standard exchange format (such as GXL), researchers will be able to more easily try different combinations of tools to meet their research needs. This should result in increased collaborations and more relevant research results. Such integrations will also lead to improved accessibility to repositories of information related to the software including code, documentation, analysis results, domain information and human activity information. Integrated tools will also lead to fewer disruptions for programmers.

c) Recommenders and search.

Software engineering tools, especially those developed in research, are increasingly leveraging advances in intelligent user interfaces (e.g., tools with some domain or user knowledge). Recommender systems are being proposed to guide navigation in software spaces. Examples of such systems include Mylar [74] and NavTracks [124]. Mylar, (now called MyLyn) uses a degree of interest model to filter non-relevant files from the file explorer and other views in Eclipse. NavTracks provides recommendations of which files are related to the currently selected files. Deline et al. also discuss a system to improve navigation [31]. The FEAT tool suggests using concern graphs (explicitly created by the programmer) to improve navigation efficiency and enhance comprehension [114]. Search technologies, such as Google, show much promise at improving search for relevant components, code snippets and related code. The Hipikat tool [30] recommends relevant software artifacts based on the developer's current project context and development history. The Prospector system recommends relevant code snippets [86]. It combines a search engine with the content assist in Eclipse to help programmers use complex APIs. Although new, this work shows much promise and it is expected to improve navigation in large systems while reducing the barriers to reuse components from large libraries.

d) Adaptive interfaces.

Software tools typically have many features, which may be overwhelming not only for novice users, but also for expert users. This information overload could be reduced through the use of adaptive interfaces. The idea is that the user interface can be tailored automatically, i.e., will self-adapt, to suit different kind of users and tasks. Adaptive interfaces are now common in Windows applications such as Word. Eclipse has several novice views (such as Gild [132] and Penumbra) and Visual Studio has the Express configuration for new users. However, neither of these mainstream tools currently have the ability to adapt nor even to be easily manually adapted to the continuum of novice to expert users.

e) Visualizations.

These have been subject of much research over the past ten to twenty years. Many visualizations, and in particular graph-based visualizations, have been proposed to support comprehensions tasks, some of then already referred in section 2.2.2. Other examples include Seesoft [11], Bloom [111], Landscape views [103], and sv3D [87]. Graph visualization is used in many advanced commercial tools such as Klocwork, Imagix4D and Together. UML Diagrams are also commonplace in mainstream development tools. One challenge with visualizing software is scale and knowing at what level of abstraction details should be shown, as well as selecting which view to show. More details about the user's task combined with metrics describing the program's characteristics (such as inheritance depth) will improve how visualizations are currently presented to the user. A recommender system could suggest relevant views as a starting point. Bull proposes the notion of model-driven visualization [24]. He suggests creating a tool for tool designers and expert users that recommends useful views based on characteristics of the model and the data.

f) Collaborative support.

As software teams increase in size and become more distributed, collaborative tools to support distributed software development activities are more crucial. In research, there are several collaborative software engineering tools being developed such as Jazz and Augur [66] [47]. There are also some collaborative software engineering tools deployed in industry, such as CollabNet, but they tend to have simple tool features to support communication and collaboration, such as version control, email and instant messaging. Current industrial tools lack more advanced collaborative features such as shared editors, and research falls short on providing empirical work to improve these

tools. Another area for research that may prove useful is the use of large screen displays to support co-located comprehension. O'Reilly et al. [101] propose a war room command console to share visualizations for team coordination. There are other research ideas in the CSCW (computer supported collaborative work) field that could be applied to program comprehension.

g) Domain and pedagogical support.

The need to support domain experts that lack formal computer science training will necessarily result in more domain-specific languages and tools. Non-experts will also need more cognitive scaffolding to help them learn new tools, languages and domains more rapidly. Pedagogical support, such as providing examples by analogy, will likely be an integral part of the future software tools. The work discussed above on recommending code examples is also suggested at helping novices and software immigrants (i.e., programmers to a new project). Results from the empirical work also suggest that there is a need for tools to help programmers learn a new language. Technologies such as TXL [29] can play a role in helping a user see examples of how code constructs in one language would appear in a new language.

III. FRAMEWORK UNDERSTANDING

Program comprehension covers a wide range of sub-areas when it comes to comprehend programs. When we say programs, we mean software artefacts: constructs built upon source-code. A framework can be considered one of such artefacts and, due to its importance and growing adherence by the software community, spawned and research area of its own. Framework understanding deals with understanding and learning about a framework for usage, implementation, and evolution.

Object-oriented frameworks are a powerful form of reuse but they can be difficult to understand and reuse correctly. They are promoted as having the potential to provide the benefits of large-scale reuse [49] [25] [43]. While practical evidence does suggest that framework usage can increase reusability and decrease development effort [95], experience has identified a number of issues that complicate framework application and limit potential benefits [18]. One of the major challenges is effective framework understanding – a specialized kind of program comprehension.

Over the past decade a large range of candidate documentation techniques has been proposed to support framework understanding, including design patterns [29], pattern languages [70], example-based learning [118], cookbooks [78], hooks [48] and exemplars [50].

However, the lack of investigation of these techniques and their impact in framework understanding, together with the lack of insight into problems that limit the comprehension and reuse of software frameworks, spurred a few studies, which identified some concerns and bases for future research in the field. The next section will briefly address some of these studies, and, afterwards, a brief review of some existing tools and approaches to aid in framework understanding and reuse. An overall depiction of the main ideas behind framework understanding is shown in Figure 2.

A. Reuse and comprehension issues

There is a considerable quantity of literature into framework domain, but little of it deals with the identification of reuse problems or evaluation of strategies to support the framework developer as a whole. There are tools that address topics under the realm of framework building, design recovery and documentation, but none clearly emphasizes or studies the overall symptoms behind ineffective framework reuse, and thus hindering a framework's main goal.

Fayad and Schmidt [43] claimed that different alternatives could improve framework understandability:

- Refining the framework's internal design.
- Using methods that can ensure a successful development and usage of frameworks.
- Adhering to standards for framework development, adaptation, and integration.
- Producing comprehensible framework documentation.

These guidelines are mainly preventive and don't focus on the issue of reusability, being general advices. Nevertheless, they can be relevant as rules of thumb for framework development and maintenance.

Butler, Keller and Milli [26] describe a taxonomy of framework documentation primitives that appear to address reusability issues. They describe six primitives, which emphasize the need for information about class interfaces and communication protocols between classes.

Johnson [70] identifies three important areas for framework documentation to address – purpose, how to use and design. He argues that the purpose of the framework and its constituent parts should be communicated so that developers may select the correct parts for a task. While knowledge of how those parts are expected to operate allows them to be employed correctly and a description of the underlying design provides developers with an understanding of how to adapt and extend the framework in a manner consistent with existing structure.

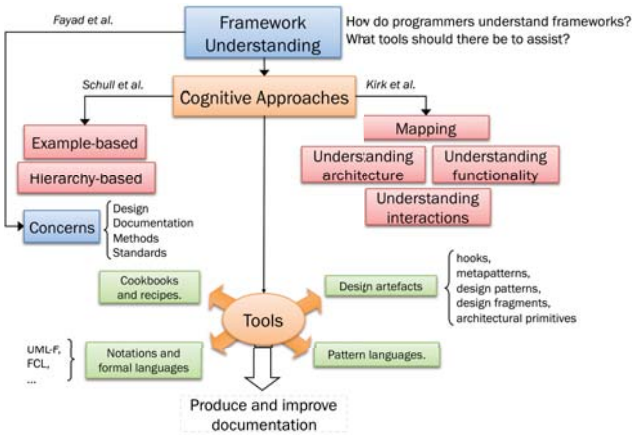


Figure 2 - Framework Understanding topics

Shull et al. [118] presents an evaluation of the role that examples play in framework reuse. Their study compared two approaches to framework reading and eventual documentation, and example-based approach and a hierarchical-based approach. Their results suggested that examples are an effective learning strategy, especially for those beginning to learn a framework. They also identify potential problems with an example-based approach: finding the small pieces of required functionality in larger examples, inconsistent organization and structure of examples and lack of design choice rationale in example documentation. They also discuss the possibility that developers become too reliant on examples and do not understand the system at a sufficient level of detail to implement effectively from scratch when necessary.

Kirk et al. [75] conducted a research, through observation of both novice and experienced re-users, where they identified four fundamental problems of framework reuse:

- Mapping identifies the problem on translating an abstract, conceptual solution into a concrete implementation, which reuses the existing structures within the framework. Such problems were often expressed as "what should I use to represent…?" or "How do I express…?"
- Understanding functionality describes problems understanding what specific parts of the framework actually do. Manifestations of this problem included "How does … work?", "Where … does happen?" or "Where is … defined/created/called?"
- Understanding interactions focuses on problems concerning the communication between classes in the framework ("What happen if …?" or "Where should I put …?"). Such problems are significant because of hidden or subtle dependencies within the framework that may cause failures to occur elsewhere as the result of a wrongly positioned modification.
- Understanding the framework architecture is the problem of making modifications without giving appropriate consideration to the high-level architectural qualities of the framework. Such alterations might have no short-term effects but ultimately lead to the framework losing its flexibility.

From these problems, the authors experimented applying two known solutions they deemed the most suited to address these issues: pattern languages and micro-architectures. Their results showed that the pattern language provided some support for mapping problems, particularly for those with no experience of the framework, by introducing key framework concepts and providing examples of framework use. However, it was clear that previous experience dominated the explicit use of the pattern language, as well as being an inhibitor to other forms of documentation as its immediacy often precludes consideration of alternative solutions.

Although the micro-architectures, used to help develop and understanding of the key interactions within the framework, seemed relatively ineffective, it is the authors' belief that documentation of this kind is necessary to address these problems in particular.

B. Tools to assist framework understanding

As for program comprehension tools, the same line of thinking applies for framework understanding tools. Both subjects share the same problems and trends, yet some framework specific issues may be addressed when devising aids to framework learning and understanding.

The past and present research in the field focus on topics that range from uncovering design artifacts to representing processes and behaviors that might help using the framework. Mostly, the proposals converge to producing and enhancing existing documentation with adequate information that can be mined and represented using different formats (recipes, cookbooks), languages (patterns, beacons, idioms) and notations (textual, graphical, UML, formal languages, etc.). Next, a brief summary of these proposals is presented. The categorization used emerged from its most relevant technique, yet several use mixed approaches combining several techniques to optimize their results.

1) Cookbooks

Confronting the challenge of communicating how to use the Model-View-Controller framework in Smalltalk-80, Krasner and Pope [78] built an 18-page cookbook that explained the purpose, structure, and implementation of the MVC framework. This cookbook was designed to be read from beginning to end by programmers and could also be used as a reference but every recipe did not follow a consistent structure nor was it suitable for parsing by automatic tools.

The Framework EDitor / JavaFrames project [59] [60] [61] has developed a language for modelling design patterns and tools that act as smarter cookbooks, guiding programmers step-by-step to use a framework. With the 2.0 release of JavaFrames, many of these tools work within the Eclipse IDE. Their language allows expression of structural constraints and the tool can check conformance with the structural constraints. Code can be generated that conforms to the patterns definition, optionally including default implementations of method bodies. Specific patterns can be

related to general patterns; for example a specific use of the Observer pattern in a particular framework can be connected to a general definition of the Observer pattern.

2) Design Artifacts

Ralph Johnson seems to be the first to suggest documenting frameworks using patterns [70]. He notes that the typical user of framework documentation wants to use the framework to solve typical problems, but also that cookbooks do not help the most advanced users [71]. Patterns can be used both to describe a framework's design as well as how it is commonly used. He argues that the framework documentation should describe the purpose of the framework, how to use the framework, and the detailed design of the framework. After presenting some graduate student with his initial set of patterns for HotDraw [20], he realized that a pattern isolated from examples is hard to comprehend.

Froelich et al.'s hooks [48] focus on documenting the way a framework is used, not the design of the framework. They are similar in intent to cookbook recipes but are more structured in their natural language. The elements listed are: name, requirement, type, area, uses, participants, changes, constraints, and comments. The instructions for framework users (the changes section) read a bit like pseudo code but are natural languages and do not appear to be parsable by tools.

Design patterns themselves can be decomposed into more primitive elements [106]. Pree calls these primitive elements metapatterns and catalogues several of them with example usage. He proposes a simple process for developing frameworks where identified points of variabilility are implemented with an appropriate metapattern, enabling the framework user to provide an appropriate implementation.

From the declarative metaprogramming group from Vrije University, Tourwé and Mens [141] [142] use Pree's metapatterns to document framework hotspots and define transformations for each framework and design patterns. Framework instances (plug-ins) can be evolved (or created) by application of the transformations. The tool uses SOUL, a prolog-like logic language. The validation was done on the HotDraw framework by specifying the metapatterns, patterns and transformations needed. The validation uncovered design flaws in HotDraw, despite its widespread use, along with some false positives. The declarative metaprogramming approach to modeling framework hotspots appears to have significant up-front investment before payoff in order to provide its guarantees about correct use of the framework. It may additionally assume a higher level of accuracy or correctness in frameworks than will commonly be found in practice. The authors comment that their approach specifically avoids design patterns in favor of metapatterns because there could be many design patterns. While this makes their technique generally applicable and composable, it will be difficult to add pattern-specific semantics and behavior checking to their approach.

JFREEDOM [44] is a design recovery tool that discovers metapatterns in a framework or software system. It relies on Tourwé's formal definition of metapatterns and uses JQuery, a logic inference-engine, to search the code for instances of these metapatterns. It then recommends possible GoF [49] design pattern instances based on its found metapatterns. Other design pattern recovery tools exist and a brief review of each one can be found in [44]. Design pattern recovery is, by itself, a research field where a community recently formed to combine efforts.

Bruch et al. [23] propose the use of data mining techniques to extract reuse patterns from existing framework instantiations. Based on these patterns, suggestions about other relevant parts of the framework are presented to novice users in a context-dependent manner. They built FrUiT, an Eclipse plug-in that implements the approach and, yet at an early stage, already presents several benefits: relying on expert-written framework instantiations, there is no need to create special artifacts such as documentation or code snippets; using data mining, significant reuse rules are extracted, only concerning how to use the framework; and the tool makes automatic context search relieving developers from searching for rules explicitly.

Fairbanks et al. [42] present a pattern language based on the notion of design fragment. A design fragment is a pattern that encodes a conventional solution to how a programmer interacts with a framework to accomplish a certain goal. It provides the programmer with a "smart flashlight" to help him/her understand the framework, illuminating only those parts of the framework he/she needs to understand for the task at hand. They use XML to express these patterns, so that automation tools are a step away. They have analyzed the 20 Java applets provided by Sun and came up with a catalogue of design fragments, which evaluated against other 36 applets from the internet proved that those design fragments were common and recurrent. Design fragments gives programmers immediate benefit through tool-based conformance and long-term benefit through expression of design intent.

Zdun and Avgeriou [151] propose to remedy the problem of modeling architectural patterns through identifying and representing a number of architectural primitives that can act as the participants in the solution that patterns convey. According to the authors, these "primitives" are the fundamental modeling elements in representing a pattern and also they are the smallest units that make sense at the architectural level of abstraction (e.g., specialized components, connectors, ports, interfaces). Their approach relies on the assumption that architectural patterns contain a number of architectural primitives that are recurring participants in several other patterns. They chose UML as the preferred notation to represent the primitives and pretend to formalize the definitions using OCL.

3) Notations and formal languages

A UML profile is a restricted set of UML markup along with new notations and semantics [46]. Fontoura et al. present the UML-F profile that provides UML stereotypes and tags for annotating UML diagrams to encode framework constraints. Methods and attributes in both framework and user code can be marked up with boxes (grey, white, half-and-half, and a diagonal slash) that indicate the method/attribute's participation in superclass-defined template patterns. A grey-box indicates newly defined or

completely overridden superclass method. A white box indicate inherited and not redefined, a half-and-half indicates refined but call to super(), and a slashed box indicates an abstract superclass method.

The Fixed, Adapt-static, and Adapt-dyn tags annotate the framework and constrain how users can subclass. Template and Hook tags annotate framework and user code to document template methods. Stereotypes for Pree's metapatterns (like unification and separation variants) are present, as are predefined tags for the GoF patterns. Recipes for framework use are present in a format very similar to that of design patterns but there is no explicit representation of the solution versus the framework. The recipe encodes a list of steps for programmer to perform.

The Framework Constraint Language (FCL) [65] applies the ideas from Richard Helms object oriented contracts [62] to frameworks. Much like Riehle's role models [112], FCLs specify the interface between the framework and the user code such that the specification describes all legal uses of the framework. The researchers raise the metaphor of FCL as framework-specific typing rules and validate their approach by applying it to Microsoft Foundation Classes, historically one of the most widely used frameworks. The language has a number of built-in predicates and logical operators and is designed to operate on the parse tree of the user's code.

C. Trends

Not as developed as program comprehension, framework understanding research still has room for expansion, and future work is needed to address existing open issues. It shares the same trends as program comprehension, yet it has its own issues. Reuse problems must be better addressed by documentation or tool support if frameworks are to be widely adopted. There are still significant and stimulant challenges:

1) Pattern languages.

While developing pattern languages for framework documentation, some issues have to be addressed such as identifying the expertise necessary to create effective pattern languages, how to identify the framework domain problems that should be the basis of patterns in the pattern language, how to best describe patterns, and what inter-pattern relationships should be included.

2) Widen context domain research.

There is a clear need to investigate the prevalence of framework understanding problems in industrial context frameworks. Industry and academia have to join efforts to ascertain the impact frameworks learning problems have in large-scale software development environments, so that adequate solution may be searched for.

3) Integrated environments.

With the advent of pluggable and extensible software development environments (like, Eclipse), tools for assisting on framework understanding tend to be integrated into these self-sustainable platforms, producing solutions that are multi-faceted and present different and varied approaches to accommodate different user needs. The combination and personalization of these tools, offer flexibility to adjust the environments to the specific needs of particular users in particular tasks.

IV. Collaborative Software Environments

Software projects usually involve a team or multiple teams that have to work together. For some time now, there has been a concern on how to coordinate these teams of developers to be able to efficiently work together. Research areas such as Groupware and Computer-Supported Collaborative Work rose to address collaboration supported by software. The Collaborative Software Engineering domain deals with collaboration within the software development process. The next sections address these research areas in further detail.

A. Groupware and CSCW

Many credit Peter and Trudy Johnson-Lenz for coining the term "groupware" in 1978. They defined it as: "intentional group processes plus software to support them". This definition, however, was not widely accepted as it has narrowed the scope of group work to a set of processes.

Another attempt to provide a definition came from Johansen [69]: "Groupware… a generic term for specialized computer aids that are designed for the use of collaborative work groups. Typically, these groups are small project-oriented teams that have important task and tight deadlines. Groupware can involve software hardware, services, and/or group process support". This definition also didn't take, as it would exclude categories of products that were not designed specifically for supporting work groups, like email or shared databases. Besides that, it also focuses on small teams, which is also restrictive.

To broaden the scope, Ellis et al. [39] proposed to define groupware as: "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment". Although less restrictive, this definition was considered too broad. Despite excluding multi-user systems (such as time-sharing systems where users don't share the same goal), it would include shared database systems. Many argue that these systems cannot be considered groupware because they provide the illusion that every user has independent access, alas, they are not "group-aware."

In general, as Grudin points out in [57] groupware means different things to different people. According to Nunamaker et al. [100], groupware is defined as "any technology specifically used to make a group more productive". Coleman states [28], "Groupware is an umbrella term for the technologies that support person-to-person collaboration; groupware can be anything from email to electronic meeting systems to workflow". These definitions although quite broad capture almost all the products and projects that are identified as groupware.

The common denominator in all the above definitions is the notion of group work. Groupware is designed to support teams of people working together. As such, groupware provides a new focus in software technology from human – computer to human – human interaction. Human interactions have three key elements: communication, collaboration and

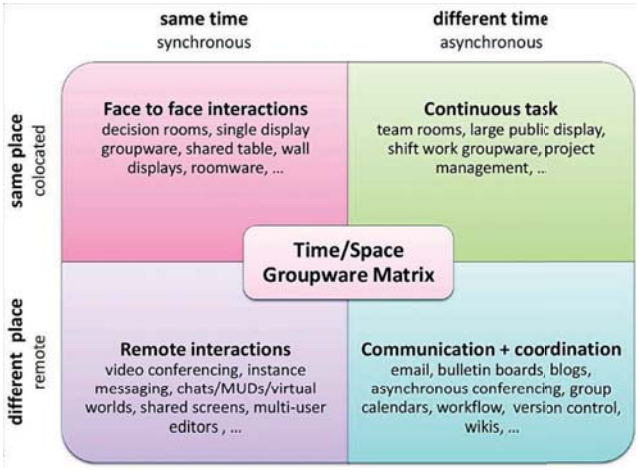


Figure 3 - Groupware Matrix (extracted from [69][10])

coordination. The goal of groupware is to assist groups in communicating, in collaborating and in coordinating their activities [39], and has been focusing on these issues for years.

The fact that most groupware tools failed to be widely adopted made clear the need for a better understanding of how groups of people work together. A new research area emerged called: "Computer-Supported Collaborative Work (CSCW)".

Iren Greif of MIT and Paul Cashman of Digital Equipment Corporation, who organized a workshop in 1984 for people interested in how groups work, coined the term CSCW. Since then, this new field attracted a lot of interest. Amongst the various definitions, Wilson's seems to have captured the scope of CSCW [148]: "CSCW [is] a generic term, which combines the understanding of the way people work in groups with the enabling technologies of computer networking, and associated hardware, software, services and techniques." Greenberg [54] adds: "CSCW is the scientific discipline that motivates and validates groupware design. It is the study and theory of how people work together, and how computer and related technologies affect group behavior."

CSCW collects researchers from a variety of specializations – computer science, cognitive science, psychology, sociology, anthropology, ethnography, management, and information systems – each contributing a different perspective and methodology for acquiring knowledge of groups and for suggesting how the group's work could be supported.

CSCW led to a better understanding of groups and made clear that group relationships are not based only on communication, collaboration and co-ordination. As pointed out by Kling [77]: "In practice, many working relationships can be multivalent with and mix elements of co-operation, conflict, conviviality, competition, collaboration, commitment, caution, control, coercion, co-ordination and combat."

CSCW researchers that design and build systems try to address core concepts in novel ways. These concepts have largely been derived through the analysis of systems designed by researchers in the CSCW community, or through studies of existing systems and the most addressed are:

- *Awareness*. Individuals working together need to be able to gain some level of shared knowledge about each other's activities [33].
- *Articulation work*. Cooperating individuals must somehow be able to partition work into units, divide it amongst themselves and, after the work is performed, reintegrate it [126].
- *Appropriation (or tailorability)*. How an individual or group adapts a technology to their own particular situation; the technology may appropriate in a manner completely unintended by the designers [34].

However, the complexity of the domain makes it difficult to produce conclusive results. The success of CSCW systems is often so contingent on the peculiarities of the social context that it is hard to generalize. Consequently, CSCW systems that are based on the design of successful ones may fail to be appropriated in other seemingly similar contexts for a variety of reasons that are nearly impossible to identify a priori [56].

In [2], Ackerman describe CSCW's main intellectual contribution has the effort to close the social-technical gap between what we know we must support socially and what we can support technically. He states that systems lack nuance, flexibility and ambiguity, clearly properties inherent to Human activity. Therefore, the social aspects must be taken into account when designing systems for these to be increasingly effective.

In [109], Weber et al. contributed with a taxonomy that defines and describes criteria for identifying CSCW systems and serves as a basis for defining CSCW system requirements. The criteria are divided into three major groups:

- *Application*. From an application viewpoint, certain tasks are generically present in many scenarios, from general-purpose tasks such as brainstorming, note taking and shared agenda features to more dedicated domains where there is the need for tailored tools. To the user, a CSCW system appears complete only when specialized and generic tools are integrated.
- *Functional*. A CSCW system relates functional features with the social aspects of teamwork. Each functionality has an impact on the work behavior and efficiency of the entire group using the system. Issues such as interaction, coordination, distribution, user-specific reactions, visualization and data hiding must be taken into consideration. However, the psychological, social, and cultural processes active within groups of collaborators are the real keys to the acceptance and success of CSCW Systems.
- *Technical*. This criteria comprises hardware, software and network support. It divides the architecture of a CSCW system into four classes of classes or features: (1) input, (2) output, (3) application, and (4) data. Each can be centralized or replicated.

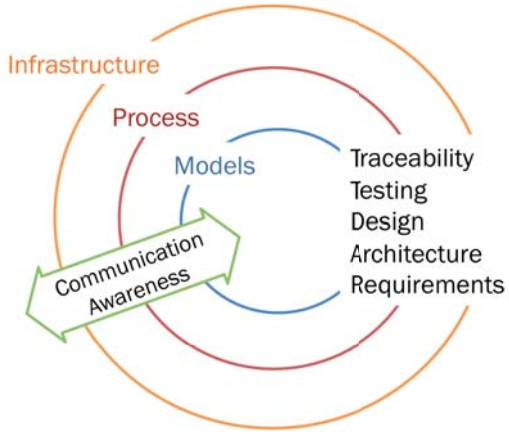


Figure 4 - Collaborative Software Engineering Model

For all of these groups, concerns such as flexibility, transparency, collaboration and sharing are addressed and guidelines for supporting them are presented.

Another approach to conceptualizing groupware and a CSCW system, states that its context can be considered along two dimensions: first, whether collaboration is co-located or geographically distributed, and second, whether individuals collaborate synchronously (same time) or asynchronously (not depending on others to be around at the same time). This approach can be seen in Figure 3 and was first introduced by Johansen [69] in 1988, also appearing in [10].

As the research continues, both groupware and CSCW fields still face challenges. The current trends evolve mostly in the following directions:

1) Mobile technologies.

With the emergence of new mobile technologies and the increasing connectivity users enjoy, the importance of having light, easy-to-use and accessible groupware features is growing.

2) Web 2.0.

With the advent of concepts of the so-called second-generation web or "Web 2.0", collaboration and contextual-connectivity become even more present in our day-to-day activities. From blogs to wikis, social software is booming and its capabilities should be harnessed to improve group work.

3) Strong commercial interest.

Major commercial competitors such as Microsoft, Google, IBM, amongst others, are releasing solutions into the market at an increasing rate. This must come as an incentive to continue researching into these ever-increasing fields of interest.

4) Delocalization of groups.

Teams and groups are becoming more and more delocalized. Work stops at one side of the planet and starts contiguously on the other side. Communication and synchronism become critical for a adequate and effective flow of work.

B. Collaborative Software Engineering

Software engineering projects are inherently cooperative, requiring many software engineers to coordinate their efforts to produce a large software system [146]. As such, this effort encompasses the development of a shared understanding surrounding multiple artefacts, each embodying its own model, over the entire development process. Figure 4 depicts that effective communication and awareness are crosscutting concerns across, not only the phases of software development but its models, process and infrastructure.

Collaboration techniques in software engineering have evolved to address our limitations: humans are slow and error-prone, especially when working at high-levels of abstraction; our natural language is expressive but ambiguous; our memory skips the details of large projects and we can't keep track of what everyone is doing.

Software engineering collaboration has multiple goals spanning the entire lifecycle of development:

a) Establish the scope and capabilities of a project.

Engineers must work with the users and stakeholders of a software project to describe what it should do at both a high level, and at the level of detailed requirements. How this collaboration takes place can have profound impact on a project, ranging from the up-front negotiation of the waterfall model, to the iterative style of evolutionary prototyping [90].

b) Converge towards a final architecture and design.

System architects and designers must negotiate, create alliances, and engage domain experts to ensure convergence on a single system architecture and design [55].

c) Manage dependencies among activities, artefacts, and organizations.

This encompasses a wide range of collaborative activities, including typical management of subdividing work into tasks, ordering them, monitoring, assessing, and controlling the plan of activities [85].

d) Reduce dependencies amongst engineers.

An important mechanism for managing dependencies is to reduce them where possible, thereby reducing the need for collaboration. Defining per-developer workspaces helps reducing dependencies in development time.

e) Identify, record and resolve errors.

Errors and ambiguities exist in all software artefacts, and many approaches have been developed to find and record them. Collaborative techniques such as inspections, reviews, beta testing and bug tracking assist on mitigating these problems and tracking the quality of the software.

f) Record organizational memory.

In any long running collaborative project, people may join and leave. Part of the work of collaboration is recording what people know, so that project participants can learn this knowledge now, and in the future [3]. SCM change logs are one form of organizational memory in software projects, as are project repositories of documentation. Process models also record organizational memory, describing best practices for how to develop software.

Collaboration in software engineering can be unstructured, where occasional and sporadic informal conversations occur concerning a piece of software anywhere in the project's lifecycle. It can also be structured, where the focus goes to various formal and semi-formal artefacts (requirement specifications, architecture diagrams, UML diagrams, source-code, bug reports, etc.) Software engineering collaboration can thus be understood as artefact-based, or model-based collaboration, where the focus of activity is on the production of new models, the creation of shared meaning around the models, and elimination of error and ambiguity within the models. Without the structure and semantics provided by the model, it would be more difficult to recognize differences in understanding among collaborators.

This focus on model-oriented collaboration embedded within a larger process is what distinguishes collaboration research in software engineering from broader collaboration research, which tends to address artefact-neutral coordination technologies and toolkits.

Software engineers have developed a wide range of model-oriented technologies to support collaborative work on their projects. These technologies span the entire lifecycle, including collaborative requirements tools [16][136], collaborative UML diagram creation, software configuration management systems and bug tracking systems [137].

Process modelling and enactment systems have been created to help manage the entire lifecycle, supporting managers and developers in assignment of work, monitoring current progress, and improving processes [17] [81]. In the commercial sphere, there are many examples of project management software, including Microsoft Project [92] and Rational Method Composer [108]. Several efforts have created standard interfaces or repositories for software project artefacts, including WebDAV/DeltaV [35][147] and PCTE [144]. Web-based integrated development environments serve to integrate a range of model-based (SCM, bug tracking systems) and unstructured (discussion list, web pages) collaboration technologies.

1) Tools, environments and infrastructure

Tool support developed specifically for collaboration in software engineering falls into four broad categories:

a) Model-based collaboration tools.

Software engineering involves the creation of multiple artifacts. These range from the end product and the source code to all the models, diagrams and specifications that cover all the phases of the software development process. Each artifact has its own semantics, with a variable degree of formality, and creating them is an inherently collaborative activity. Systems designed to support the collaborative creation and editing of specific artifacts are really supporting the creation of specific models, and hence support the model-based collaboration. Collaboration tools exist to suppose the creation of every kind of model found in typical software engineering practice.

b) Process centred collaboration.

A software process model structures steps, roles and artifacts to create during software development. Typically, engineers reduce the amount of overhead coordination to initiate the project, tackling more quickly with the project at hand, rather than negotiating the entire project structure. Overtime, as experience grows, the net effect is to reduce the amount of coordination work required within a project by regularizing points of collaboration, as well as to increase predictability of future activity. Process centered software development environments have facilities for writing software process models in a process modeling language, then executing these models in the context of the environment. For example, the environment can manage the assignment of tasks to engineers, monitor their completion, and automatically invoke appropriate tools. Some examples of such systems are Arcadia [72], Oz [12], Marvel [13], ConversationBuilder [73], and Endeavours [17].

c) Collaboration awareness.

Software engineering is a human-driven and human-intensive activity. Most medium- to large-scale projects involve multiple software developers that may or may not be co-located. In recent years, there has been much work in developing collaborative development environments that provide support for coordination and communication during software development [66]. A key issue in any collaborative is awareness, or "knowing what is going on" [40]. More precisely, awareness is "an understanding of the activities of others, which provides a context for [one's] own activity" [33]. Awareness encompasses knowing who else is working on the project, what they are doing, which artifacts they are or were manipulating, and how their work may impact other work. In distributed collaborative work, maintaining awareness is considerably more difficult. Research areas ranging from software visualization to reverse engineering have been developing tools and techniques to provide awareness during software development. Seesoft [37], Palantir [116], Lighthouse [122] and Jazz [66] are but a few. A more extensive survey and comparison study can be found at [134].

d) Collaboration infrastructure.

Various infrastructure technologies make it possible for engineers to work collaboratively. Software tool integration technologies make it possible for software tools to coordinate their work. Major forms of tool integration include data integration (ensuring that tools can exchange data), control integration (ensuring that tools are aware of activities of other tools and can take action based on that knowledge). For example, nowadays, most IDEs know when a source-file is saved after editing and store it on a central repository (data integration) or SCM, then automatically call the proper compiler (control integration). Tools like Eclipse, Visual Studio, Marvel and WebDAV already implement these behaviors. Whether through calling other external tools based on the context of the task or coordinating between integrated tools, these environments already bring a sustainable collaboration between engineers and theirs development tasks.

2) Trends and future research directions

There are still several areas to be addressed for improving collaboration in software engineering, which may reveal the future trends on this domain of expertise.

a) Integrating web and desktop environments.

The migration of development tools to the web is increasing, now that the user interface is becoming more sophisticated (thanks to AJAX and its overall adoption) and the processing power of browsers is higher. UML and source code editing are no longer relegated only to desktop applications, whereas in the past, the web could not support such features. Despite this trend, there is a longstanding practice surrounding the use of integrated development environments (Visual Studio, Eclipse, JBuilder, etc.), which are not going to be displaced by completely web-based environments. Instead, future projects are likely to adopt a mixture of web-based and desktop tools, for which interfacing open standards between the desktop IDE's and the web-based services should be created. Although not an easy task, these open standards would allow a more seamless interaction with the complex information a software project creates.

b) Broader participation in design.

Currently, software customers are engaged in the development process during requirements elicitation, but then become not so engaged for the requirements analysis, design and coding phases, only to reconnect again for the final phase of testing. Broadened participation by customers in the requirements analysis, design, coding and early testing phases would keep them engaged during these middle stages, allowing a more actively assurance that their direct needs are met. By increasing the participation of the direct end users, software engineers can reduce the risk that the final product does not meet the needs of customer organizations. Surely, a balance between completely open-sourced projects and a fine-grained proprietary closed-source model available for the customer to refine has to be made. Nevertheless, a participatory development model would allow customers a better tailoring of the software to their needs. The trend toward providing support for distributed development teams in a wide range of development tools makes a broader engagement possible. Open source SCM tools like Subversion, as well as web-based requirements tools and problem tracking tools make it possible to coordinate globally distributed teams.

c) Capturing rationale argumentation.

One of the strongest design criteria used in software engineering is design for change, which inherently involves making predictions about the future. As a result, the design process is not just and engineer making rational decisions from a set of facts, but instead is a predictive process in which multiple engineers argue over current facts and future potentials. Architecture and design are argumentative processes in which engineers resolve differences of prediction and interpretation to develop models of a software system's structure. Since only one vision will prevail, the process of architecture and design is simultaneously cooperative and competitive. Providing collaborative tools to support engineers in the recording and visualization of architecture and design argumentation structures would do a better job of capturing the nuances and tradeoffs involved in creating large systems. They would also better convey the assumptions that went into a particular decision, making it easier for succeeding engineers to know when they can safely change a system's design.

d) Using novel communication and presence technologies.

Software engineers have a long track record of integrating new communication technologies into their development processes. Email, instant messaging and web-based applications are very commonly used in today's projects to coordinate work and be aware of whether other developers are currently active (present). As a result, engineers would be expected to adopt emerging communication and presence technologies if they offer advantages over current tools. For instance, networked collaborative 3d game worlds are such an emerging technology that spawned "software immersion environments". Second Life is an example of using such a 3D world to develop software, as their team uses its own platform to do so. There is a range of research issues inherent to the use of 3D virtual environments as a collaboration infrastructure, for example, how to synchronize physical and virtual worlds. Ultimately, the utility of adopting a 3D virtual world needs careful examination, as the benefits of the technology need to clearly exceed the costs. It is currently very unclear that this is true.

e) Improved assessment of collaboration technology.

Assessing the impact of the introduction of new technology into a project is difficult, and usually subjective. Estimation in software development is a difficult task, which hinders the objective assessment of collaboration technology. Without the uncovering of the pros and cons of specific collaboration tools, forward progress in the field of software collaboration support tools is hard to measure. There is a lack of studies how already introduced tools (instant messaging, Internet-aware SCM tools, email, bug tracking systems, etc.) that quantify the benefits received from using these collaboration tools. Developing improved methods for assessing the impact of collaboration tools would boost research in these areas by increasing confidence in positive results, and making it easier to convince teams to adopt new technologies.

V. OPEN ISSUES

Program Comprehension deals with understanding programs and software artefacts. Framework Understanding focuses on a specific kind of software artefact: a framework. This understanding is often made resorting only to information on the artefact itself and accompanying documentation. More and more, software is developed collaboratively. Can this "collaboration" help in framework understanding?

From the state-of-the-art review, a number of open research issues arise. An insight of the most relevant ones follows, as a means to focus the reader to the intended scope:

a) Frameworks are often hard to understand and use.

The difficulty of understanding frameworks is a serious inhibitor of effective framework reuse. This is mainly due to framework designs being usually very complex, and thus hard to communicate. The framework design is: (1) very abstract, to factor out commonality; (2) incomplete, requiring additional classes to create a working application; (3) more flexible than needed by the application at hand; (4) obscure, in the sense that it usually hides existing dependencies and interactions between classes. The learning curve becomes steep, requiring a considerable amount of effort to understand and learn how to use a framework.

b) Framework documentation is often outdated and inaccurate.

Good documentation significantly improves the process of learning and understanding new frameworks. By guiding users on the customization process and by explicitly showing the framework design principles and details, effective documentation contributes to make frameworks easy to reuse. Despite these reasons, framework documentation is still regarded as of low importance within the framework development process. Most commonly during maintenance or evolution phases, documentation is used to assist on these tasks but its update is often discarded or neglected. Moreover, it is still hard, costly and tiresome to define and write good quality documentation for a framework. Good documentation should be easy to use, support different audiences and provides multiple views through different types of documents and notations. The difficulty of producing contents for these requirements may hinder its applicability and demotes its importance within the development process.

c) Programmers (both experts and novices) recurrently tackle with understanding problems.

Every time a software developer needs to re-use a piece of code, whether it's a snippet, class, library or framework, she goes over the entire cognitive process of analyzing, understanding and capturing the relevant information she needs. Depending on the purpose of the task at hand (learning, teaching, communicating, using), the format (quality, clarity, structure, abstraction level, etc.) of the code, and the experience of the programmer (expert or novice) the understanding process may go through various approaches (top-down, bottom-up, etc.), not always leading to the desired outcome in a straight forward manner. Choosing the adequate understanding process should not be difficult, and changing from one to another should be feasible without much overhead.

d) The process of understanding a framework is not properly dealt with.

The palette of tools available to the framework learner scarcely deals with specific aspects of framework understanding. Without questioning its local and highly focused solutions, each tool aids in a specific aspect, whether capturing high-level design artifacts, browsing the code for hot-spots, or helping on producing sustainable output formats. Alas, the framework user has to navigate through a plethora of tools trying to figure out where the relevant information might be.

e) Different tools provide sparse results with variable quality.

By itself, each tool has its own problems and limitations, thus producing quality-questionable results. For instance, many of the problems design recovery (reverse engineering) tools have, tend to converge to selection of results (elimination of false positives) and semantic overlapping (same result can have several meanings). With such discrepancy amongst results, it becomes difficult to ascertain tool efficiency and compare results regarding precision and recall.

f) Collective knowledge of the development team is often not harnessed at its best.

Software development is a highly social process. It has been perceived that, when trying to understand a piece of code, developers turn first to the code itself and, when that fails, to their social network, that is, the team. This behavior, not only happens during code understanding, but also throughout the whole understanding process. Nevertheless, it is not easy to go for the team. Firstly, it is not clear who to address for clarification, for there is a lack of awareness of what other members of the team are doing or how do they relate to the work done. Secondly, the fields of expertise are not clear or stated, leading to wasteful interruptions of the wrong people. Thirdly and most often, the team or the experts are not available for consulting or rebuke their fellow colleagues due to interruption. Interrupted developers lose track of parts of their mental model, resulting in laborious reconstruction or bugs and discouraging more frequent interruptions.

g) Implicit developers' knowledge is not captured and shared as effectively as it could be if well supported.

Developers go to great lengths to create and maintain rich mental models of design and code that are rarely permanently recorded. Very often, developers, without referencing written material, can talk in detail about their product's architecture, how the architecture is implemented, who owns what parts, the history of the code, to-dos, wish-lists, and meta-information about the code. For the most part this knowledge is never written down, except in transient forms such as sketches on a whiteboard. The bottom-line problem here is that "Lots of [useful] information is kept in peoples' heads" [80].

VI. KEY RESEARCH QUESTIONS

From the open issues presented before, a few research questions revolve around a major question that is considered central to this research work:

1) How to improve framework understanding?

a) What kind of information do developers try to capture first? What makes them decide?

b) What are the actual goals of the framework learner?

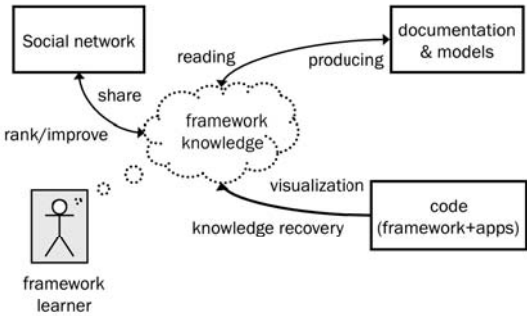


Figure 6 - Framework learning environment

c) Are there any typical and repeated behaviours developers apply when trying to learn how to use a framework?

d) How can tools assist the learning process?

e) What kind of information is presented to framework learners that they mostly look for? What do they look for that isn't there?

f) What is missing from existing development environments to assist on framework understanding?

In this paper, the authors address mainly questions *d)* and *f)*, and believe that to improve framework understanding, tools should be collaborative and specific knowledge should be captured and presented to the developers. The next section will address how they intend to pursue that.

VII. IMPROVING THROUGH COLLABORATION

Teams collaborate to develop software. But not all of the relevant knowledge is recorded for later use. Developers tackle recurrently with understanding and learning issues, especially if teams rotate their members often. Team members take tacit knowledge with them that decays with time and that proves useful later on. That knowledge, if permanent and available, could save time when dealing again with the system. What if that knowledge could be shared with other developers, novice or expert? The idea is to make that knowledge available within the development environment. Therefore, (re) learning about the system (frameworks, in this case) should benefit from knowing how its was learnt in the first place.

A. *Supporting the learning process*

Learning how to use a framework is not a trivial task. The learner is usually engaged in a *process* composed of a series of activities. This process has *best practices* that can be followed to improve its outcome. These practices could be actively applied and improved having *tools* to support them. These three levels are detailed next and depicted in Figure 6.

1) Process

In this particular domain, there is range of activities that may characterize the developer's behavior while trying to understand a framework. These activities may fall into three categories:

a) Code

Figure 5 - Learning environment support levels

"Where is all that we need to know"(?) The problem is that what we need to know is not explicitly in front of us. Furthermore, frameworks make it particularly difficult to find what we need to know. As an example, recovering design knowledge implicit in the code is a recurring practice to help clarify the framework's structure and purpose. The questions reside on what kind of design artifacts, to what kind of audience, and how to store and present the results so that they are useful.

b) Documentation.

When the developer wants to learn how to use a framework (or any reusable software artifact, for that matter), she goes for the documentation, if it exists. But, is there always documentation? And is that documentation clear, well suited and complete? Does it have all the answers? There are known ways of producing good documentation for frameworks [6][7][8]. The issue is nurturing the developers to easily produce and access that documentation, even during the learning process.

c) Social network.

When all else fails, the developer loses her self-sufficiency as a learner and resorts to her "contacts", that is, strong candidates to bear knowledge that might help her. Call it team, peers, social network, buddies or any other term, there is knowledge that one can't find anywhere else but on people's minds. It is called intrinsic knowledge. Getting this knowledge is intrusive. There should be ways of harnessing this knowledge without such intrusiveness.

Putting it short, a framework learner looks at the code, reads the documentation, visualizes information and asks her colleagues for help, as going through a learning process of understanding how to use the framework. Figure 5 (extracted from [5]) depicts this scenario.

2) Best Practices

Associated with the learning process, there is a series of good practices on how to deal with each stage of the learning process. These are presented in [45]. A learning environment should support and nurture these practices.

3) Supporting Tools

Depending on several factors (learner's experience, existing artifacts, learning goal, etc.) the learning process to undertake may resort to different practices and paths. What works for some, might not work for others, and may even vary between frameworks. Novices and experts will take different paths.

Yet, in a truly collaborative environment, where, at first, there is no distinction between who is expert and who is

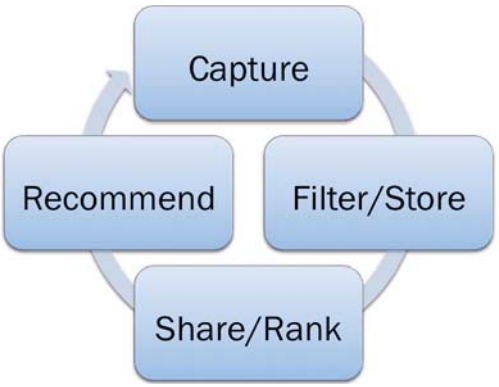


Figure 7 - Supporting steps to improve the learning process

novice, sharing experiences and advising the global community proves useful [135]. The importance given to an advice or counsel is measured by its actual applicability. You became experienced and expert by giving valid and helpful feedback into the community.

By supporting this sharing of knowledge, the learners may benefit from their collective intelligence, thus improving their own learning processes. Therefore, the supporting tools should be prepared to capture this knowledge, share it and assist other learners in their tasks.

B. Tool requirements

Teams turn into communities mainly due to high member rotation, high project preemption and the widely spread of frameworks. The strength of this community relies mainly on its ability to withhold valuable knowledge, filtering out what its not important. The issue is providing an effective infrastructure to share this information amongst its members without too much effort and to allow a "natural" selection of what is actually valuable.

Providing such an environment would have the following requirements:

a) Seamless integration into a IDE.

Tools and features to support the learning process should be available within the development environment as a means to enforce usage, without disrupting the normal way a developer works. When presented with possible solutions, it should be straightforward how to proceed within the IDE to apply those solutions.

b) Non-intrusive / non-interruptive.

Ideally, capturing the developer's intrinsic knowledge should be implicit. That is, the developer should not be asked to explicitly provide any information regarding that knowledge to the system. In practice, a satisfactory solution would be to notify the system we are trying to learn how to do some task and signal reaching that goal. Bottom-line, it should be as non-intrusive as possible.

c) Context-aware.

The tools should be aware of the context where the developer is learning and provide information that makes sense within that context.

d) Web-aware.

Not only should the environment seek knowledge within its own boundaries (its knowledge-base), it should also be prepared to go to the web in a contextual manner.

e) Descriptive, not prescriptive.

The system should not tell the developer how to proceed, but instead should give possible directions on how to solve the task at hand.

f) Shared knowledge-base.

The environment should store and share all the relevant knowledge that helps the framework learning process. Not only the documentation artifacts and source-code, but the captured knowledge that helps guiding the developer throughout the process.

g) Learning Path

Different developers learn in different ways. The environment should be able to deal with the learning profile of its users, considering aspects such as visualization of information and easy personalization of contents.

The learning process would be supported relying on a four-step cycle shown in Figure 7. The purpose would be to capture the learning steps taken by the learner. Whether she looks at the code first, goes for documentation, explores certain artifacts, and recovers others, until she reaches a satisfying conclusion. This path would then be recorded, stored and shared. "Sharing" means that other learners may reuse it or get assistance through it to guide them on their own learning path. If the shared knowledge really helped them, then they should rank it or improve it. As the collected knowledge keeps improving (through sharing, usage and ranking), the best learning strategies will be recommended to recurrent learners and thus improving their learning process.

Candidate existing environments are Eclipse, Jazz Team Concert and Visual Studio due to their extensible nature and pluggable architecture. These environments have a notion of context or process-awareness, yet they miss the learning context. The idea would be to insert the notion of a learning process and provide tools to assist in that process, supporting the steps depicted in Figure 7. The tools should be built as plug-ins to the collaborative environment, introducing a learning context and accompanying the learner throughout the process.

VIII. CONCLUSION

Frameworks are good software artifacts for reuse. Nevertheless they are complex, thus hard to learn. Most of the tools that may help in this task don't encompass the social nature of software development. In distress, learners tend to look for help at their colleagues, often disrupting their work. Supporting the social side of software development by raising awareness and capturing intrinsic knowledge helps improving the learning of software, namely, frameworks.

A set of requirements for tools to harness framework learning knowledge and assisting in the process of learning should: allow for seamless integration into an IDE; be non intrusive or interruptive; be context and web aware; be

descriptive instead of prescriptive; share a common knowledge base of evolving learning knowledge and capture the learning path taken by developers.

Providing a collaborative environment where learning knowledge can be captured, shared, ranked and recommended to recurrent learners, both expert and novice, in a non-intrusive way, aims at improving framework understanding.

REFERENCES

[1] N.Flores, "Patterns and Tools for improving Framework Understanding: a Collaborative Approach", SEDES Doctoral Symposium, ICSEA - The Fourth International Conference on Software Engineering Advances, Porto, Portugal, September 2009.

[2] M. Ackerman, (2000). "The Intellectual Challenge of CSCW: The gap between social requirements and technical feasibility". Human-Computer Interaction 15: 179–203

[3] M. S. Ackerman and D. W. McDonald (2000), "Collaborative Support for Informal Information in Collective Memory Systems", in Information Systems Frontiers, vol. 2, o. 3/4, pp. 333-347, 2000.

[4] Adoption-Centric Software Engineering Project site. Computer Science Department at University of Victoria, Canada URL:http://www.acse.cs.uvic.ca/index.html. Accessed at 30-06-2010

[5] A.Aguiar, "Framework Documentation – A Minimalist Approach", PhD Thesis, FEUP September 2003.

[6] A.Aguiar and G.David, "Patterns for Documenting Frameworks – Part III", PLoP'2006, Portland, Oregon, USA, October 2006.

[7] A.Aguiar and G.David, "Patterns for Documenting Frameworks – Part II", EuroPLoP'2006, Irsee, Germany, July 2006.

[8] A.Aguiar and G.David, "Patterns for Documenting Frameworks – Part I", VikingPLoP'2005, Helsinki, Finland, September 2005.

[9] R.Baecker and A.Marcus, "Human Factors and Typography for More Readable Programs". ACM Press, Addison-Wesley Publishing Company, 1990

[10] R.M. Baecker et al., (1995). "Readings in human-computer interaction: toward the year 2000". Morgan Kaufmann Publishers.

[11] T. Ball and S.G. Eick, "Software visualization in the large", IEEE Computer, 29, 4, pp.33-43, 1996

[12] I. Z. Ben-Shaul (1994), "Oz: A Decentralized Process Centered Environment (PhD Thesis)," in Department of Computer Science: Columbia University, Dec 1994.

[13] I. Z. Ben-Shaul, G. E. Kaiser, and G. T. Heineman (1992), "An Architecture for Multi-user Software Development Environments," in ACM SIGSOFT 92: 5th Symposium on Software Development Environments, Tyson's Corner, Virginia, 1992, pp. 149-158.

[14] C.F. Bertholf and J.Scholtz, "Program Comprehension of Literate Programs by Novice Programmers", Empirical Studies of Programmers: 5th Workshop, 1993

[15] T.J.Biggerstaff, B.W Mitbander, and D.Webster, "The concept assignment problem in program understanding", Proceedings of the 15th International conference on Software Engineering, pp.482-498, 1993.

[16] B. Boehm and A. Egyed (1998), "Software Requirements Negotiation: Some Lessons Learned", in the 20th International Conference on Software Engineering (ICSE'98), Japan, 1998, pp.503-507

[17] G. A. Bolcer and R. N. Taylor (1996), "Endeavors: a Process System Integration Infrastructure," in 4th International Conference on the Software Process (ICSP'96), Brighton, UK, 1996, pp. 76-89.

[18] J.Bosch, P.Molin, M.Mattsson, and P.O. Bengtsson, "Framework – Problems and Experiences" Building Application Frameworks, M.Fayad, D.Schmidt, R.Johnson, Wiley, 1999.

[19] M.S.K.Brade, M.Guzdial, and E.Soloway, "Whorf: A visualization tool for software maintenance". Proceedings 1992 IEEE Workshop on Visual Languages, pp.148-154, 1992

[20] J.M.Brant, "HOTDRAW", MsC Thesis, University of Illinois, 1995

[21] R. Brooks, "Towards a theory of the comprehension of computer programs", International Journal of Man-Machine Studies, pp. 543-554, vol.18, 1983.

[22] M.H.Brown and R.Brooks, "ZEUS: A system for algorithm animation and multi-view editing". Proceedings of the IEEE 1991 Workshop on Visual Languages, pp.4-9, 1991

[23] M.Bruch, T.Schäfer, and M.Mezini, "FrUiT: IDE Support for Framework Understanding", OOPSLA Eclipse Technology Exchange, 2006

[24] R.I.Bull and M-A. Storey, "Towards Visualization Support for the Eclipse Modeling Framework", A Research-Industry Technology Exchange at EclipseCon, 2005.

[25] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. "Pattern oriented software architecture - a system of patterns". John Wiley and Sons, 1996.

[26] G.Butler, R.Keller, and H.Milli, "A Framework for Framework Documentation", Computing Surveys, special Symposium Issue on Object-Oriented .Application Framework, ACM, 2000.

[27] T.Cheng, S.Hupfer, S.Ross, and J.Patterson (2007). "Social Software Development Environments", Dr.Dobb's Journal. Janury 11th , 2007.

[28] K. Coleman (1995), "Groupware technology and Applications", Prentice Hall PTR 1995

[29] J.R. Cordy, T.R. Dean, A.J.Malton, and K.A. Schneider, "Source Transformation in Software Engineering using the TXL Transformation System", Journal of Information and Software Technology, vol(44)13, pp. 827-837, 2002.

[30] D.Cubranic, G.C. Murphy, J.Singer, and K.S.Booth, " Hipikat: A project memory for software development", IEEE Transactions on Software Engineering 31, 6 (Jun. 2005), pp. 446-465.

[31] R. DeLine, A.Khella, M.Czerwinski, and G.Robertson, "Towards Understanding Programs through Wear-based Filtering", Softvis, 2005

[32] F. Détienne, "Software Design – Cognitive Aspects", Springer Practitioner Series 2001.

[33] P. Dourish and V. Bellotti, (1992). "Awareness and coordination in shared workspaces". Proceedings of the 1992 ACM conference on Computer-supported cooperative work: 107-114, ACM Press New York, NY, USA.

[34] P. Dourish, (2003). "The Appropriation of Interactive Technologies: Some Lessons from Placeless Documents". *Computer Supported Cooperative Work* **12**: 465–490. Kluwer Academic Publishers.

[35] L. Dusseault (2003), WebDAV: Next-Generation Collaborative Web Authoring, Prentice Hall PTR, 2003.

[36] Eclipse project site. URL: http://www.eclipse.org. Accessed at 30-06-2010.

[37] S. G. Eick, J. L. Steffen and, E. E. Sumner Jr. (1992) "SeeSoft – a tool for visualizing line oriented software statistics." IEEE Transactions on Software Engineering 28, 4, 396-412.

[38] T.Eisenbarth, R. Koschke, and D. Simon, "Aiding Program Comprehension by Static and Dynamic Feature Analysis", Proceedings of the IEEE International Conference on Software Maintenance, 2001

[39] C. A. Ellis, S. J. Gibbs, and G. L. Rein (1993), "Groupware some issues and experiences". In: Baecker, Ronald M. Readings in groupware and computer-supported cooperative work. San Francisco: Morgan Kaufmann, 1993. p. 9-28.

[40] M. Endsley (1995), "Toward a theory of situation awareness in dynamic systems". Human Factors 37, 1, 32-64.

[41] K. Erdös, and H.M. Sneed, "Partial Comprehension of Complex Programs (enough to perform maintenance)", Proceedings of the 6th

International Workshop on Program Comprehension, pp. 98-105, 1998.

[42] G.Fairbanks, D.Garlan and W. Scherlis, "Design Fragments Make Using Frameworks Easier", OOPSLA 2006 .

[43] M.Fayad, D.Schimdt, and R.Johnson, "Building Application Frameworks", Wiley 1999.

[44] N.Flores and A.Aguiar, "JFREEDOM: a Reverse Engineering Tool to Recover Framework Design", Proceedings of the 1st International Workshop on Object-Oriented Reengineering, ECOOP'05, 2005.

[45] N. Flores and A.Aguiar, (2008) "Patterns for Framework Understanding", in Proc. of th 15th Pattern Languages of Programming Conference (PLoP'08).

[46] M.Fontoura, W.Pree, and B.Rumpe, "The UML Profile for Framework Architectures", Addison-Wesley Professional 2001.

[47] J. Froehlich and P. Dourich, "Unifying artifacts and activities in a visual tool for distributed software development teams", Proceedings of the 26th International Conference on Software Engineering, pp. 387-396, 2004.

[48] G. Froehlich, H.Hoover, L.Lui, and P.Sorenson, "Hooking into Object-Oriented Application Frameworks", Proceedings of the 19th International Conference on Software Engineering, pp.491-501, 1997

[49] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns — Elements of reusable object-oriented software". Addison-Wesley, 1995.

[50] D.Gangopadhyay and S.Mitra, "Understanding Frameworks by Exploration of Exemplars", Proceedings of CASE-95, IEEE Computer Society, pp. 90-99, 1995

[51] J.J.Garrett. "Ajax: A New Approach to Web Applications". Adaptive Path, February 18th 2005. URL: http://www.adaptivepath.com/publications/essays/archives/000385.php. Accessed at 30-06-2010.

[52] D. M. German, "Decentralized open source global software development, the GNOME experience," Journal of Software Process: Improvement and Practice, vol. 8,no. 4, pp. 201–215, 2004.

[53] T.R.G. Green and M. Petre, "When Visual Programs are Harder to Read than Textual Programs", Human-Computer Interaction: Tasks and Organization, Proceedings (ECCE)-6 (6th European Conference Cognitive Ergonomics), 1992

[54] S. Greenberg (1991), "Computer-supported Co-operative Work and Groupware", Academic Press Ltd., London, 1991.

[55] R. Grinter, (1999), "Systems Architecture: Product Designing and Social Engineering," in ACM Conference on Work Activities Coordination and Collaboration (WACC'99), San Francisco, California, 1999, pp. 11-18.

[56] J. Grudin, (1988). "Why CSCW applications fail: problems in the design and evaluation of organization of organizational interfaces". Proceedings of the 1988 ACM conference on Computer-supported cooperative work: 85-93, ACM Press New York, NY, USA.

[57] J. Grudin, (1994), "Computer-Supported Co-operative Work: History and Focus", Computer, Vol. 27, No. 5, May 1994

[58] C. Gutwim, R.Penner, and K. Schneider, "Group Awareness in Distributed Software Development", ACM CSCW, pp. 72 – 81, 2004

[59] M.Hakala, J. Hautamäki, K.Koskimies, J.Paakki, A.Viljamaa, and J.Viljamaa, "Annotating reusable software architectures with specialization patterns", Proceedings of the Working IEEE/IFIPConference on Software Architecture (WICSA'01), pp. 171, 2001

[60] I.Hammouda and K.Koskimies, "A pattern-based j2ee application development environment", Nordic Journal of Computing 9(3), pp. 248-260, 2002

[61] J.Hannemann and G. Kiczales, "Design pattern implementation in java and aspectj", Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages and applications, pp.161-173, 2002.

[62] R.Helm, I.Holland, and D.Gangopadhyay, "Contracts: specifying behavioral compositions in object-oriented systems", Proceedings of the European conference on object-oriented programming (ECOOP'90), pp. 169-180, 1990

[63] T. Hendrix, J.H. Cross II, L. Barowski, and K.Mathias. "Tool support for reverse engineering multi-lingual software". Proceedings of the 4th Working Conference on Reverse Engineering (WCRE'97), pp. 136-143, 1997

[64] L. Hohmann, "Journey of the Software Professional: The Sociology of Software Development", 1996.

[65] D.Hou and H.J. Hoover, "Towards specifying constraints for object-oriented frameworks", Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research, page 5, IBM Press, 2001.

[66] S. Hupfer, L.-T Cheng, S. Ross, and J. Patterson, "Introducing collaboration into an application development environment", Proceedings of the ACM Conference on Computer Supported Cooperative Work, pp. 444-454, 2004.

[67] Imagix 4D. Imagix Corporation. URL: http://www.imagix .com/index.html.

[68] S.Isoda, T.Shimomura, and Y. Ono, "VIPS: A visual debugger", IEEE Software, May 1987

[69] R. Johansen (1988) "Groupware: Computer Support for Business Teams" The Free Press.

[70] R.Johnson, "Documenting Frameworks using Patterns", Proceedings of the OOPSLA'92, SIGPLAN notices, 27(10), pp.63-76. 1992

[71] R.Johnson, "Components, framework, patterns. SIGSOFT Software Engineering Notes, 22(3), pp. 10-17, 1997

[72] R. Kadia (1992), "Issues Encountered in Building a Flexible Software Development Environment," in ACM SIGSOFT 92: 5th Symposium on Software Development Environments, Tyson's Corner, Virginia, 1992, pp. 169-180.

[73] S. M. Kaplan, W. J. Tolone, A. M. Carroll, D. P. Bogia, and C. Bignoli (1992), "Supporting Collaborative Software Development with ConversationBuilder," in ACM SIGSOFT 92: 5th Symposium on Software Development Environments, Tyson's Corner, Virginia, 1992, pp. 11-20.

[74] M.Kersten and G.Murphy, "Mylar: a degree-of-interest model for IDE's", International Conference on Aspect Oriented Software Development, pp.159-168, 2005

[75] D.Kirk, M.Roper, and M.Wood, "Identifying and Addressing Problems in Framework Reuse", Proceedings of the 13th International Workshop on Program Comprehension (IPWC'05), pp. 77-86, 2005

[76] G. Kizcales, J.Lamping, A.Mendhekar, C.Maeda, C.V.Lopes, J-M Loingtier, and J. Irwin. "Aspect Oriented Programming", Proceedings of the European Conference on Object-Oriented Programming (ECOOP), June 1997.

[77] R. Kling (1991) "Co-operation, Co-ordination and Control in Computer-Supported Work", CACM, vol. 34, no. 12, December 1991.

[78] G.E.Krasner, S.T.Pope,"A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80", Journal of Object-Oriented Programming 1,3, pp. 26-49, 1988

[79] M.Lanza and S.Ducasse, "A Categorization of Classes based on Visualization of their Internal Structure: the Class Blueprint".Proceedings for OOPSLA 2001, pp.300-311, ACM Press, 2001

[80] T.D.LaToza, G.Venolia, and R.DeLine (2006), "Maintaining Mental Models: A Study of Developer Working Habits". Proc. of the International Conference of Software Engineering (ICSE'06), Shanghai, China.

[81] B. S. Lerner, L. J. Osterweil, Stanley M. Sutton Jr., and A. Wise (1998), "Programming Process Coordination in Little-JIL Toward the Harmonious Functioning of Parts for Effective Results," in European Workshop on Software Process Technology, 1998.

[82] S.Letovsky, "Cognitive processes in program comprehension", Empirical Studies of Programmers, pp.58-79, 1986

[83] P.Linos, P.Aubet, L.Dumas, Y.Helleboid, P.Lejeune, and P.Tulula. "Visualizing program dependencies: An experimental study" Software-Practice and Experience, 24(4):387-403, 1994

[84] D.C Littman, J.Pinto, S.Letovsky, and E.Soloway, "Mental models and software maintenance", Empirical Studies of Programmers, pp. 80-98, 1986

[85] T. W. Malone and K. Crowston (1994), "The Interdisciplinary Study of Coordination," in ACM Computing Surveys (CSUR), vol 26, no 1, pp. 87-119, 1994.

[86] D. Mandelin, L. Xu, R. Bodik and D.Kimelman, "Mining Jungloids: Helping to Navigate the API Jungle", Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, pp 48-61, 2005.

[87] A. Marcus, L. Feng, J.I. Maletic, "Comprehension of Software Analysis Data Using 3D Visualization", Proceedings of the IEEE International Workshop on Program Comprehension (IWPC2003), pp.105-114, 2003

[88] A. von Maryhauser and A. Vans. "Program comprehension during software maintenance and evolution" IEEE Computer pp. 44-55, August 1995.

[89] A. von Maryhauser and A. Vans. "From code understanding needs to reverse engineering tool capabilities" Proceedings of CASE'93, pp. 230-239, 1993.

[90] S. McConnell (1996), "Lifecycle Planning," in Rapid Development: Taming Wild Software Schedules Redmond, WA: Microsoft Press, 1996.

[91] A.Mendelson and J.Sametinger. "Reverse Engineering by visualizing and querying", Software – Concepts and Tools, 16:170-182, 1995

[92] Microsoft Corporation (2007), "Microsoft Office Project Standard 2007 Product Guide," April 2006, Available at: http://download.microsoft.com/download/d/f/b/dfb0e645-3e5a-4fe8-8ea6-1c9e86d6139a/ProjectStandard2007ProductGuide.doc. Accessed at 30-06-2010

[93] Microsoft Visual Studio (2008) website. http://msdn.microsoft.com/en-us/vstudio/default.aspx. Accessed on 30-06-2010

[94] A.Mockus, R.Fielding, and J.D. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla", ACM Transactions of Software Engineering and Methodology, 11, 3, pp.309-346, 2002

[95] S.Moser and O. Nierstrasz, The Effect of Object-Oriented Frameworks on Productivity", IEEE Computer, pp.45-51, 1996

[96] H.Muller and K. Klashinsky. "Rigi – A system for programming-in-the-large", Proceedings of the 10th International Conference on Software Engineering (ICSE'10), pp.80-86, 1988.

[97] G.C. Murphy, D.Notkin, and K.Sullivan, "Software Reflexion Models: Bridging the Gap Between Source and High-Level Models", Proceedings of Foundations of Software Engineering, pp. 18-28, 1995.

[98] A.Murray and T.Lethbridge, "On Generating Cognitive Patterns of Software Comprehension", Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research, pp.200-211, 2005.

[99] NATO, Software Engineering Conference, Garmisch, Germany, 7-11 October 1968.

[100] J. F. Nunamaker, R. O. Briggs, and D. D. Mittleman (1995). "Electronic meeting systems: Ten years of lessons learned." In D. Coleman, & R. Khanna (Eds.), Groupware: Technology and Applications (pp. 149–192). Englewood Cliffs, NJ: Prentice-Hall, 1995.

[101] C. O'Reilly, D.Bustard, and P.Morrow, "The War Room Command Console (Shared Visualizations for Inclusive Team Coordination)", Softvis, 2005.

[102] N.Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs", Cognitive Psychology, pp. 295-341, vol 19, 1987.

[103] D.A. Penny, "The Software Landscape: A Visual Formalism for Programming-in-the-Large", PhD Thesis, University of Toronto, 1992.

[104] M.Petre, A. Blackwell, and T.Green, "Cognitive questions in software visualization". Software Visualization: Programming as a Multi-Media Experience, pp. 453-480. MIT Press, 1997

[105] M.Petre, "Why looking isn't always seeing: readership skills and graphical programming", Communications of the ACM, vol. 38, pp. 33-44, 1995

[106] W.Pree, "Design Patterns for Object-Oriented Software Development", Addison-Wesley, 1994

[107] V.Rajlich, N.Damskinos, and P.Linos, "VIFOR: A tool for software maintenance". Software-Practice and Experience, 20(1):67-77, 1990

[108] Rational Software Corporation (2003), "Rational RequisitePro User's Guide," June 2003, http://www.ibm.com/developerworks/rational/library/content/03July/getstart/RP/ReqProMain.pdf. Accessed on 30-06-2010.

[109] W. Reinhard, J. Schweitzer, G. Volksen, and M. Weber, (1994) "CSCW tools: concepts and architectures," Computer , vol.27, no.5, pp.28-36, May 1994

[110] S.Reiss, "Pecan: Program development systems that support multiple views", IEEE Transactions on Software Engineering, SE-11(3): 276-285, 1985

[111] S.Reiss, "An overview of BLOOM", Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program analysis for software tools and engineering, pp.2-5, 2001.

[112] D.Riehle, "Framework Design: A Role Modelling Approach", PhD Thesis, Swiss Federal Institute of Technology, 2000

[113] R.S. Rist, "plans in programming: Definition, Demonstration, and Development", Empirical Studies of Programmers, 1st Workshop, 1986.

[114] M.P. Robillard and G.Murphy, "FEAT: A tool for locating, describing, and analyzing concerns in source code", Proceedings of the 25th International Conference on Software Engineering, pp. 822-823, 2003

[115] G.-C.Roman, K.C. Cox, C.D. Wilcox, and J.Y.Plun. "Pavane: A system for declarative visualization of concurrent computations". Techinical Report WUCS-91-26, Washington University, St. Louis, 1991.

[116] A. Sarma, Z. Noroozi, and A. van der Hoek (2003). "Palantír – raising awareness among configuration management workspaces". In Proc. Of the 25th International Conference on Software Engineering, 444-454

[117] D.A. Scanlan, "Structured flowcharts outperform pseudocode: An experimental comparison", IEEE Trans. Soft. Eng., 1989

[118] F.Schull, F.Lanubile, and V.Basil, "Investigating Reading Techniques for Object-Oriented Framework Learning", IEEE TSE, vol.26, nº.11, 2000

[119] B. Shneiderman and R.Mayer, "Syntatic/semantic interactions in programmer behavior: A model and experimental results". International Journal of Computer and Information Science, pp. 219-238, 8(3), 1979

[120] B. Shneiderman, "Measuring computer program quality and comprehension", InternationalJournal of Man-Machine Studies, vol. 9, pp. 465-478, 1977

[121] B. Shneiderman, R.Mayer, D.McKay, and P.Heller, "Experiment investigations of the utility of detailed flowcharts in programming", Communications of the ACM, vol. 20, pp. 373-381, 1977

[122] I. A. da Silva, P. H. Chen, C. V. der Westhuizen, R. M. Ripley and A. van der Hoek (2006)" Lighthouse: Coordination through Emerging Design".In Proc. of the 2006 OOPSLA workshop on eclipse technology eXchange.

[123] J. Singer, T.Lethbridge, N.Vinson, and N. Anquetil, "An Examination of Software Engineering Work Practices", Proceedings of CASCON'97, pp. 209-233, 1997

[124] J. Singer, R. Elves, and M-A. Storey, "NavTracks Demonstration: Supporting Navigation in Software Space", International Workshop on Program Comprehension, 2005

[125] P.Schorn, A. Brungger, and M. de Lorenzi, "The XYZ Geobench: Animation of geometric algorithms. Animations for Geometric Algorithms: A Video Review, Digital Systems Research Center, Palo Alto, California, 1992

[126] K. Schmidt and L. Bannon, (1992). "Taking CSCW seriously". Computer Supported Cooperative Work 1: 7–40.

[127] E. Soloway, J.Pinto, S.Letovsky, D.Littman, and R.Lampert, "Designing documentation to compensate for delocalized plans", Communication of the ACM, 31(11): 1259-1267, 1988.

[128] E. Soloway and K. Erlich, "Empirical studies of programming knowledge", IEEE Transactions on Software Engineering, pp. 595-609, SE-10(5), September 1984.

[129] M-A Storey, "Theories, Methods and Tools in Program Comprehension: Past, Present and Future." Proceedings of the 13th IEEE International Workshop on Program Comprehension (IWPC). St. Louis, MO, pp. 181-191, IEEE Computer Society Press 2005.

[130] M-A Storey, F.Fracchia, and H.Muller. "Cognitive design elements to support the construction of a mental model during software visualization". Proceedings of the 5th International Workshop on Program Comprehension (IWPC'97), Dearborn, Michigan, pp. 17-28, May, 1997

[131] M-A Storey, F.Fracchia, and S.Carpendale, "A top down approach to algorithm animation" Techincal Report CMPT 94-05, Simon Frasier University, Brunaby, B.C., Canada, 1994

[132] M-A Storey, "Designing a Software Exploration Tool Using a Cognitive Framework of Design Elements", Software Visualization, 2003.

[133] M-A Storey, J. Michaud, M. Mindel, M. Sanseverino, D. Damian, D,Myers, D.Geman and E.Hargreaves, "Improving the Usability of Eclipse for Novice Programmers", Eclipse Technology eXchange (eTX) Workshop at OOPSLA 2003.

[134] M-A. D. Storey, D. Cubranic, and D.M. German (2005), "On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework". In Proc. of the 2005 ACM symposium on Software Visualization, 193-202.

[135] J. Surowiecki, (2004) "The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations", Anchor Publishing.

[136] Rational DOORS, IBM (2008), Nov. 12, 2008. Available at: http://www-01.ibm.com/software/awdtools/doors/productline/. Accessed at 30-06-2010

[137] The Bugzilla Team (2008) , "The Bugzilla Guide - 3.3 Development Release". Mar 5, 2008. Available at: http://www.bugzilla.org/docs/tip/en/pdf/Bugzilla-Guide.pdf. Accessed at 30-06-2010

[138] W.F. Tichy, N. Habermann, and L. Pretchelt (1993). "Summary of the dagstuhl workshop on future directions in software engineering: February 17-21, 1992, schlo dagstuhl. ACM SIGSOFT Software Engineering Notes, 18(1):35-48

[139] S.Tilley and D.B.Smith, "Coming Attractions in Program Understanding", Technical Report CMU/SEI-96-TR-019, 1996

[140] S.Tilley, S.Paul, and D.Smith. "Towards a framework for program understanding". WPC'96: 4th Workshop on Program Comprehension, Berlin, Germany, pp. 19-28, March 1996

[141] T.Tourwé, "Automated Support for Framework-Based Software Evolution", PhD Thesis, Vrije Universiteit, 2002

[142] T.Tourwé and T.Mens, "Automated Support for Framework-Based Software Evolution", Proceedings of the International Conference on Software Maintenance, page 148, 2003

[143] I.Vessey, "Expertise in debugging computer programs: A process analysis", International Journal of Man-Machine Studies, pp. 459-494, vol 23, 1985.

[144] L. Wakeman and J. Jowett (2005), "PCTE: The Standard for Open Repositories": Prentice Hall, 1993. International Conference on Software Engineering Research, Management and Applications (SERA'05), Mount Pleasant, Michigan, USA, 2005, pp. 86-93.

[145] A. Walenstein, "Observing and Measuring Cognitive Support: Steps Toward Systematic Tool Evaluation and Engineering", 11th International Workshop on Program Comprehension (IWPC'03), pp. 185-195, May 2003.

[146] J. Whitehead (2007). "Collaboration in Software Engineering: A Roadmap". In *2007 Future of Software Engineering* (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 214-225.

[147] E. J. Whitehead, Jr. and Y. Y. Goland, (1999) "WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web," in 6th European Conference on Computer Supported Cooperative Work (ECSCW'99), Copenhagen, Denmark, 1999, pp. 291-310.

[148] P.Wilson, (1991). "Computer Supported Cooperative Work: An Introduction". Kluwer Academic Pub, 1991.

[149] K. Wong. "The Reverse Engineering Notebook", Ph.D Thesis, University of Victoria, 2000.

[150] I. Zayour and T.C. Lethbridge, "A Cognitive and User Centric Based Approach For Reverse Engineering Tool Design", Proceedings of the CASCON 2000.

[151] U.Zdun and P.Avgeriou, "Modelling Architectural Patterns Using Architectural Primitives", OOPSLA 2005

[152] M.V. Zelkowitz and D.R. Wallace (1998). "Experimental models for validating technology". IEEE Computer, 31(5):23-31

Automatic Identification of Cohesive Structures within Modularity Reengineering

Anja Bog Oleksandr Panchenko Kai Spichale Alexander Zeier
Hasso Plattner Institute for Software Systems Engineering
University of Potsdam
August-Bebel-Str. 88, 14482 Potsdam, Germany
Email: {anja.bog, panchenko, kai.spichale, zeier}@hpi.uni-potsdam.de

***Abstract*—The quality of software systems depends heavily on the quality of their structure, which affects maintainability and readability. To improve the quality of structure, a system can be restructured. This paper describes a restructuring process, which uses a combination of strongly connected component analysis, dominance analysis, and intra-modular similarity clustering to identify and preserve structures that have been thoughtfully placed together, but would be separated by pure metric-based or similarity-based techniques. The use of the proposed method allows a significant reduction of the number of components that should be moved. Therefore, the number of false movements is alleviated. The proposed approach was implemented in a prototype and illustrated by statistics and examples from 18 open source Java projects. A coherence metric is introduced to further improve restructuring results.**

***Keywords*-Source code organization, Restructuring, reverse engineering, and reengineering, Metrics**

I. Introduction

Each time that a software element (e.g., method, class, package) is added, the developer has to decide where this element has to be placed. It is likely that the developer chooses a suboptimal position because of the limited ability of humans to cope with the increasing complexity of software systems. Besides this, any source code change could introduce new dependencies among software elements, which might adversely affect the system structure. Dependencies could also vanish, which allows creating simplified configurations. This paper is an extended version of our previous work [1], integrating the proposed pre-processing techniques into the entire process of restructuring and adding discussions about further techniques to enhance the results, i.e., cohesion.

In this paper we present an approach for generating restructuring advice to improve the physical structure [2] of software systems. Restructuring advice comprises moving misplaced software elements, whereas dependencies among software elements are kept unchanged. Thus, restructuring advice leads to another system configuration.

The proposed approach includes a preprocessing phase and a restructuring phase. In the restructuring phase, several alternative configurations of the original system are created and compared to each other based on coupling, cohesion, and coherence. However, not all configurations that lead to better values of these metrics are acceptable. Not heeding design decisions of the original system and only improving metric values, may pull apart cohesive structures consisting of elements that were thoughtfully placed together. Therefore, given configurations must not be ignored as they capture well-considered design decisions. The preprocessing phase identifies such structures that should be preserved during restructuring and helps to distinguish between intended and unaware decisions.

Restructuring advice is created as the result of the preprocessing and restructuring phase. In the following steps this advice is validated by developers and eligible restructuring advice can be implemented. For the preprocessing phase we propose techniques that are applied to identify intended cohesive structures and to mark them for preservation during restructuring. Since the techniques used in this paper are based purely on structural analysis of the software system, semantical meanings of the elements are not taken into account and are out of scope for this paper. The results of the preprocessing phase are further enhanced by applying the cohesion metric in the restructuring phase in order to choose an optimal system configuration.

The following section introduces the graph structure used as the basis for the restructuring algorithms. Section III relates the approach to existing research. An overview of the proposed reengineering process is given in Section IV. The subsequent section focuses on the preprocessing phase of the reengineering process detailing the steps and algorithms, which are applied in order to create restructuring advice. Section VI discusses a further possibility to improve the restructuring results by introducing coherence metric. Afterwards, a short overview of our implementation to create restructuring advice is given in Section VII. Section VIII concludes the paper and gives an overview of possible directions for future work.

II. Module Dependency Graph

The proposed restructuring approach is independent of any programming language. To accomplish this objective, the described techniques are based on the Module Dependency Graph (MDG) [3] that has three types of elements: *components*, *modules*, and *dependencies*. A *component* represents an atomic software element whose internal structure is not considered at this level of granularity. Calls

between components are represented by dependencies. Each pair of distinct components can be linked by at most one *dependency*. The components and their dependencies form a directed graph. *Modules* are disjoint sets of components. Figure 1 shows the elements of an MDG. Notice that the inter-modular dependency between b and c implies a module dependency between M_1 and M_2.

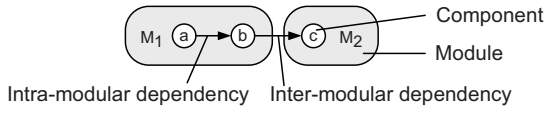


Figure 1. MDG Elements

A graph is defined as a pair (V, E), where V is the vertex set and E is the edge set. However, the graph described above must be extended to satisfy the need to reflect modularization properties. Therefore a partition is defined. A partition of a set X is a set of nonempty subsets of X such that every element x in X is in exactly one of these subsets. Thus, an MDG is a triple (V, E, M), where (V, E) is a directed graph and M represents modules and is a partition of V.

The MDG can be applied at different levels of granularity. Java applications can be modeled as follows: Java classes are represented by *components* and packages by *modules*. Experiments have also been executed on a larger system developed with the SAP [4] component model that divides software projects into development components (DCs) to organize the software in comprehensible and reusable units. Further, a software component (SC) combines DCs to larger units for delivery and deployment. The DCs are modeled by *components* and SCs by *modules*.

Traditionally, various metrics have been used to assess the quality of the MDG. Most popular metrics for coupling and cohesion are used as optimization criteria for metric-based refactoring. Coupling of a module *m* [5, p. 520] is the degree of dependence between *m* and other modules of the MDG and is represented by the number of afferent and efferent dependencies. Cohesion of the module *m* [5, p. 524] is the measure of the strength of structural connections of components inside *m* and is calculated as the number of actual dependencies divided by the number of maximal possible dependencies within a module. The module, that has only one component, has a cohesion value equal to one.

III. Related Work

Several techniques for automating the decomposition of software systems into subsystems and improving their structure exist. In this section, categories of techniques, concrete techniques, and their fields of application are presented. Tool-driven reengineering techniques are aimed at architecture reconstruction and software restructuring. Architecture reconstruction captures component recovery and program understanding of single systems and product lines [6]. Software restructuring aims at improving the physical design of existing code [7]. Due to the high number of software elements and relations among them, maintaining and improving the structural quality must be supported and automated by tools. As we are interested in improving the structure of software systems, we are focusing on software restructuring techniques in the following. Design structure matrices [8] and reflexion models [2] provide means to model the structure of software systems and thereby gain insights to support their maintenance and evolution. Furthermore, (semi-)automated techniques exist, that extract abstractions from software artifacts to make software systems more understandable, e.g., Storey et al. [9] developed the interactive, visual tool Rigi that helps understanding software systems.

Beyond modeling the structure of a software system, subsystem decomposition techniques help to provide proposals for improving its structure. Respective techniques are categorized by their underlying technologies into clustering techniques, graph-based techniques, and multi-approach techniques [10]. Clustering techniques utilize similarity measures for components and modules to group the most similar ones. Graph-based techniques model relevant properties of subsystems as graph properties and optimize them. Multi-approach techniques use a mixture of techniques from the aforementioned categories.

Concerning clustering techniques, Hutchens and Basili [7] proposed an algorithm that clusters procedures by measuring the interaction between pairs of procedures. Schwanke's tool Arch [11] clusters similar software elements based on their common and distinct references. Girard, Koschke, and Schied [12] extended Schwanke's similarity metric to cluster functions, types, and variables into atomic elements.

K-cut modularization proposed by Jermaine [13] is an example for a graph-based technique. This method decomposes a software system into modules in such a way as to attempt the minimization of inter-module connections. As a result, modules with high cohesion and low coupling are identified. The problem of software decomposition is formulated as the k-cut problem in graph theory. The computation of the k-cut of a graph is an NP-hard problem, however, efficient approximations exist [14]. K-cut modularization is most appropriate for monolithic procedural systems.

Regarding multi-approach techniques, Mitchell and Mancoridis [15] developed Bunch, a tool that identifies subsystems based on maximizing cluster cohesion, while minimizing inter-cluster coupling. Tzerpos and Holt [16] developed the algorithm ACDC that recognizes subsystem patterns and places software elements based on lowering coupling.

However, none of the mentioned techniques was explicitly developed for providing restructuring advice for misplaced components. There is no technique that detects subsystem patterns to preserve existing structures. ACDC detects subsystem patterns to create a skeleton, but the

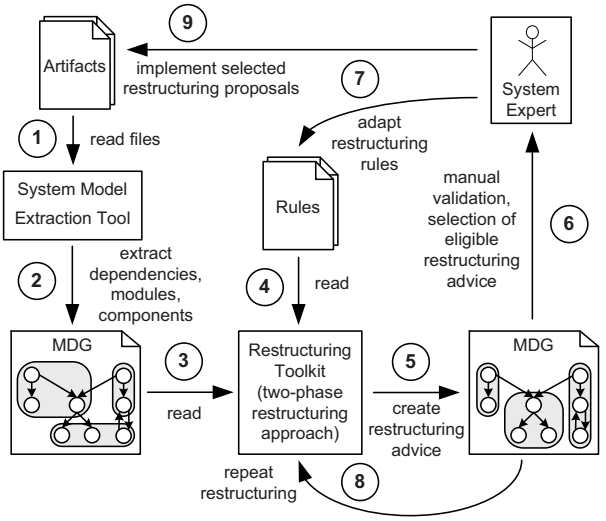


Figure 2. Reengineering Process

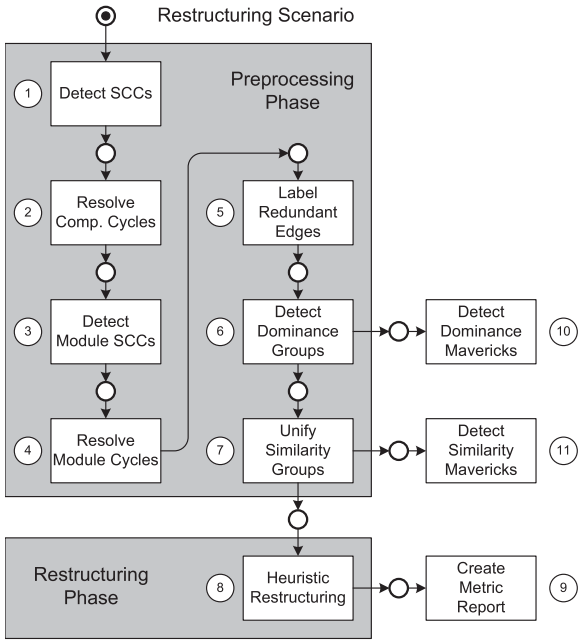


Figure 3. The Two-phase Restructuring Approach

identified subsystems are not restricted to the given configuration. Furthermore, a subsystem decomposition technique is needed that also resolves cyclic module dependencies. This need is based on the Acyclic Dependency Principle, a design principle formulated by Robert C. Martin [17] that indicates that dependencies between software elements of the granularity of release must not form cycles. Applied to the MDG, this design principle stipulates that any cyclic dependencies between modules have to be resolved. Our empirical investigation shows that, although this design principle is widely accepted, the most systems lack of proper structure.

Techniques that are merely based on maximizing cluster cohesion and minimizing inter-cluster coupling cannot create acceptable results because the proposed configuration often requires too many component moves. High cohesion and low coupling are commonly agreed to be attributes of good design. Although, a configuration with optimal metric values does not inevitably imply an optimal design. Furthermore, clustering techniques based solely on similarity cannot reasonably place all software elements because of too low similarity values.

In the following section we will introduce our reengineering process that detects and preserves constructs that have been consciously placed together.

IV. Reengineering Process

Figure 2 shows the steps and their sequence within a simplified version of the entire reengineering process. (1) The process starts with analyzing the physical artifacts (e.g., source code, deployment descriptors, configuration files) of a software system. (2) Tools automatically extract data about the software elements and the dependencies among them to create a system model in the form of an MDG. (3) The created MDG is the input data for the used restructuring techniques. (4) Rules can be defined to limit possible restructuring proposals that contradict intended design decisions. Such rules comprise modification suppressions for software elements. By this means a component can be bound to a module in such a way that it cannot be moved into another module. (5) Finally, graph theory and clustering techniques are applied to propose restructuring advice. (6) Not all proposals might be suitable to the intended design. Therefore the proposals must be validated and selected by a system expert. (7) If the restructuring proposals are not satisfying, the rules can be adapted. (8) After changing the rules, the analysis can be repeated. (9) The process is finished when the approved restructuring proposals are implemented.

The detailed activities of the two-phase restructuring approach step that is proposed in this paper are shown in Figure 3. The restructuring approach comprises a number of individual graph-based techniques as well as clustering techniques that are applied one after another to the MDG. The selection and order of these techniques depend on the intended purpose, which is in our case restructuring of an existing software system in order to improve the overall structure.

(1) In the first step, all cyclic dependencies on component level are detected in the form of strongly connected components (SCCs). One separate or several overlapping dependency cycles constitute a strongly connected component. Cyclic dependencies are useful information for restructuring insofar as placing all components of an SCC into the

same module reduces the inter-modular coupling. (2) In the second step, the detected component SCCs are collapsed and placed into appropriate modules. Collapsing SCCs pulls together all interconnected components. By this means the dependency structure becomes acyclic at component level. Further, cyclic module dependencies are resolved if the interconnected components were originally in different modules. (3) Next, cyclic module dependencies are detected. Even if the dependency graph is acyclic on component level after collapsing all SCCs, cyclic module dependencies can exist. Cyclic dependencies among modules are architectural flaws that have to be detected and removed. (4) Cyclic module dependencies are resolved by moving components from one module to another. (5) For dominance analysis, redundant edges have to be labeled. (6) Dominance subgraphs can then be detected and collapsed without introducing new cyclic dependencies. (7) In the last step of the preprocessing phase, similar components are unified by clustering them within the same module.

(8) In the second phase, component moves between different modules are proposed by heuristic restructuring in order to improve metric values. (9) At the end a metric report is created that compares the original configuration with the proposed configuration based on metrics.

(10) Detection of dominance mavericks shows misplaced components based on dominance analysis. The results of dominance mavericks detection are not integrated into the final restructuring advice as the number of false positives is high according to our experiments. Nevertheless, these results should be reviewed by an expert of the analyzed system and integrated manually if applicable. (11) Similarity mavericks detection analyzes components that are misplaced based on similarity. Results are not part of the final restructuring advice, either, but should be reviewed by an expert as they might reveal further structural improvement.

In the following section, more detail about the steps within the preprocessing phase is provided.

V. Preprocessing Phase

The purpose of the preprocessing phase is to (1) resolve cyclic module dependencies and to (2) identify cohesive structures with dominance analysis and intra-modular similarity clustering.

Removing cyclic dependencies in a software system increases maintainability and extensibility as will be explained in this section. Additionally, acyclic graphs are a prerequisite for the dominance analysis in the following step.

The goal of identifying cohesive structures is to distinguish between thoughtfully intended and unaware decisions to position components in order to improve the final reengineering results. Dominance analysis detects connected components, and similarity clustering identifies elements with similar structure.

Table I
Analyzed projects

Name	Source
Apache Ant 1.7.1	http://ant.apache.org/
CruiseControl 2.7.3	http://cruisecontrol.sourceforge.net/
Eclipse Ganymede SR1	http://www.eclipse.org/ganymede/
Apache Geronimo 2.1.2	http://geronimo.apache.org/
Hibernate 3.3.0	http://www.hibernate.org/
JBoss 4.2.3 jdk6	http://www.jboss.org/
JDepend 2.9	http://clarkware.com/software/JDepend.html
J2SE 5.0 JDK 1.5.0_09	http://java.sun.com/javase/
JRuby 1.1.4	http://jruby.codehaus.org/
JUnit 4.5	http://www.junit.org/
Apache Logging Services for Java 1.2.15	http://logging.apache.org/
Apache Maven 2.0.9	http://maven.apache.org/
NetBeans 6.1 (Base IDE)	http://www.netbeans.org/
PicoContainer 2.5.1	http://www.picocontainer.org/
Saxon 9.1.0.1	http://saxon.sourceforge.net/
Spring Framework 2.5.5	http://www.springsource.org/
Apache Tomcat 6.0.18	http://tomcat.apache.org/
Xalan-j 2.7.1	http://xml.apache.org/xalan-j/

The examples given in this paper are selected after performing an analysis of 18 open source Java projects. The usefulness of the approach is exemplified by statistics. The list of the selected projects is given in Table I.

To automate the analysis, a tool has been implemented. The tool uses Classycle[18] to extract runtime dependencies among Java classes. JAR files are analyzed by Classycle and, as a result, the MDG is created in XML format. The algorithms used in the proposed approach have been implemented to work with the MDG in this format.

A. Resolving Cyclic Module Dependencies

Cyclic dependencies form SCCs. An SCC of a digraph G is a maximal strongly connected subdigraph of G. A digraph is strongly connected if there is a directed walk from each vertex to each other vertex [19].

The components of an SCC can be part of several modules. If this is the case, cyclic module dependencies are created. To resolve these cyclic module dependencies, the SCCs are collapsed. Possible locations of a collapsed SCC are the modules that contain at least one component that is part of the SCC. The module that implies the lowest coupling is chosen.

Even if the dependency graph is acyclic on component level after collapsing all SCCs, cyclic module dependencies can exist. Alternative modifications to remove these pseudo-cyclic dependencies are *component moving*, *module splitting*, and *module merging*.

As the name states, in component moving a component is moved from one module to another to resolve the pseudo-cycle. Component moving may result in an empty module, which has to be deleted. In module splitting a selected module is split into two separate modules in such a way

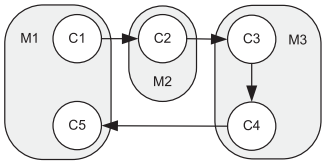


Figure 4. Pseudo-cyclic Dependencies on Component Level

that the pseudo-cyclic dependencies are removed. However, not every module that is part of the cycle can be split so that the intended effect occurs. Module splitting can be interpreted as a special case of component moving, as it requires the movement of components into a newly created or existing module. Consequently, the conditions under which this strategy is applied are the same as in component moving. Module merging is the reverse modification of module splitting. Modules are unified with the objective of turning their undesired inter-modular dependencies into intra-modular dependencies. Module merging is not selective, i.e., modules are merged in their entirety. Therefore, component moving and module splitting are far more fine-grained than merging.

To determine which components are part of a cycle on module level that is not cyclic on component level, the order of the modules has to be determined and which components are the cause of it. Each module in a cycle has the role of a successor and predecessor to another module. Therefore, for each module pair (p, s) two component sets called predecessor subset P and successor subset S are defined, where p is the predecessor and s the successor. "$\rightarrow$" denotes dependencies between components. The sets are defined as follows:

$$P(p,s) := \{v \mid v \in p \wedge \exists t \in s : \exists\{r_1, ..., r_n\} \subseteq p : v \rightarrow r_1 \rightarrow ... \rightarrow r_n \rightarrow t\}$$
$$S(p,s) := \{v \mid v \in s \wedge \exists r \in p : \exists\{t_1, ..., t_n\} \subseteq s : r \rightarrow t_1 \rightarrow ... \rightarrow t_n \rightarrow v\}$$

The predecessor subset contains all components $v \in p$ that directly or indirectly depend on a component $t \in s$ not considering paths including other modules. Similarily, the successor subset is the subset of s containing components v that are directly or indirectly referenced by the components in p.

The algorithm for resolving pseudo-cyclic dependencies is based on the fact that the inter-component dependencies are acyclic. Consequently, there exists at least on non-empty set of components that can be moved to resolve the cyclic dependencies on module level.

Figure 4 shows an example containing pseudo-cyclic dependencies on module level. To determine which component subsets can be moved to resolve these dependencies, predecessor subset and successor subset are analyzed for each of the modules, see Table II.

If the predecessor subset of a module overlaps with its successor subset, then neither the predecessor nor the successor subset can be moved to break the pseudo-cyclic dependencies. In the example, only C_1 and C_5 could effectively be moved. Hence, we can identify the following options:

- Moving $\{C_1\}$: The component subset $\{C_1\}$ is the predecessor subset of M_1 with regard to M_2. If $\{C_1\}$ is moved to M_2, the dependencies on module level become acyclic.
- Moving $\{C_5\}$: The successor subset of M_1 with regard to M_3 is $\{C_5\}$. Moving this subset to M_3 is another valid solution.
- Splitting M_1: One of the identified disjoint subsets can be moved into a separate module.

If multiple solutions exist, the solution is chosen that requires the smallest number of component moves and creates the configuration with the lowest coupling. Typically, software systems contain a large number of cycles [20]. If several cycles overlap, the algorithm has to be applied iteratively.

According to Fowler [21], cycles in dependency structures should be avoided as they provoke situations, where every change of one module breeds other changes that come back to the original module entering a vicious circle of change propagation. Systems become tightly coupled by cyclic dependencies and fiercely resist decomposition.

Drawbacks of cyclic dependencies are: (1) higher complexity, since modules cannot be understood independently. The goal of modularization is to divide a complex system into simpler modules that can be independently developed, maintained, and understood [22], whereas tight coupling, caused by cyclic dependencies diminishes the ability to understand modules in isolation [23, p. 85]; (2) less flexibility and extensibility is a result of cyclic dependencies as the program is harder to understand because of increased complexity, and coupled components can be affected by changes. Cycles make it harder to accurately assess and manage the impact of changes to the system.

Cyclic dependency analysis is an important aspect of the proposed approach because 31.5% of the components and 53.7% of the modules in the analyzed projects are involved in cyclic dependencies. Melton and Tempero's empirical study [20] confirms the high amount of cyclic dependencies between classes and packages, which was also discovered in our analysis: 52% of the component level SCCs remain inside a module. Consequently 48% of the component level SCCs are distributed over more than one module and cause cyclic dependencies among modules.

A large number of cyclic dependencies requires many component movements to resolve cycles, which results in complex refactorings at the beginning of the process. In

Module	Predecessor subset	Successor subset	Intersection of subsets
M_1	$P(M_1, M_2) = \{C_1\}$	$S(M_3, M_1) = \{C_5\}$	$\emptyset$
M_2	$P(M_2, M_3) = \{C_2\}$	$S(M_1, M_2) = \{C_2\}$	$\{C_2\}$
M_3	$P(M_3, M_1) = \{C_3, C_4\}$	$S(M_2, M_3) = \{C_3, C_4\}$	$\{C_3, C_4\}$

Table II. Predecessor subset, successor subset and subset intersections of the example given in Figure 4

this case, human intervention is needed to continue with the reengineering process.

B. Dominance Analysis

Components often reference underlying components that provide specific functions, which cannot be understood or reused individually. If an underlying component is an essential part of the referencing component, then referenced and referencing components must not be separated by any restructuring attempt.

Figure 5 (A) shows a client using a facade, a unified interface hiding a complex subsystem. The facade and the covered components must be reckoned as one unit to prevent dispersing this coherent structure.

Figure 5 (B) shows two clients depending on some utility components. When `Client2` was developed, its common utility functions were extracted to the component `CommonUtil`. `Client1` can use `CommonUtil` without referencing `Client2`. The component `SpecialUtil` emerged when the developers of `Client1` decided to encapsulate some functions. But no other component depends on `SpecialUtil`. `Client1` and `SpecialUtil` belong together and must not be separated. Nevertheless, if `SpecialUtil` was developed as a reusable component, a rule could be defined to enable the separation of both components.

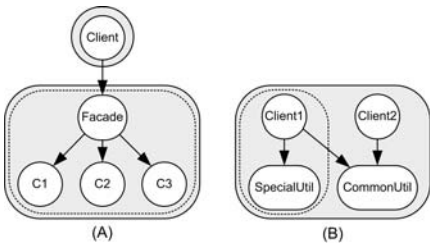


Figure 5. Examples of Dominance Subgraphs

Dominance analysis is the process of identifying intra-modular subgraphs that can be collapsed without introducing cyclic dependencies. The first step, is collapsing all SCCs as mentioned above. This step is common to other proposed approaches [24], [25], because the algorithms for transitive closure used for dominance analysis require acyclic graphs as input. Next, all redundant dependencies are removed. An edge e part of a directed graph G is said to be redundant iff e can be removed without changing the transitive closure of G [26]. Then, the algorithm goes through all vertices v and examines whether v qualifies as dominated vertex. The vertex v is said to be dominated iff there exists exactly one vertex d that is linked to v by an edge (d, v). Dominator vertex and dominated vertex form a dominance pair if they are part of the same module. One separate or several overlapping dominance pairs constitute a dominance subgraph. The dominance subgraph detection is repeated until no further dominance pairs can be detected.

Figure 6 shows an example of dominance analysis. The SCC $\{e, f\}$ detected in part (A) is collapsed in part (B). The dotted edges in part (B) denote redundant dependencies. In part (C) the redundant dependencies are filtered and three dominance pairs are found that form two collapsed dominance subgraphs in part (D).

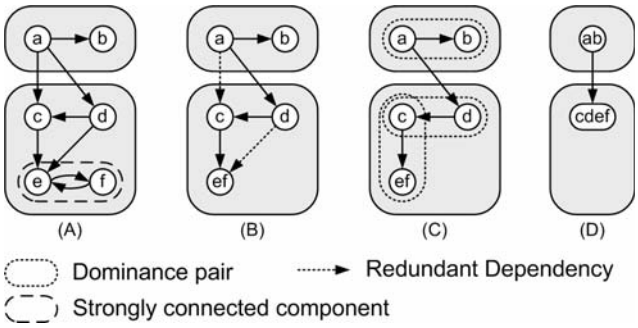


Figure 6. Steps of Dominance Analysis

Only components within the same module should be united, otherwise too many components would be pulled together. Since dominance subgraphs are not spread over multiple modules, subsequent restructuring attempts must either move complete subgraphs or keep them unchanged in their modules.

Other proposed dominance analyses [16], [24] are restricted to rooted (sub-)trees and unsuitable to detect nested dominance subgraphs due to redundant edges.

During preprocessing 32.1% of the analyzed classes could be assigned to dominance subgraphs. There are 1.82 dominance subgraphs per package. Based on a manual review, the identified dominance subgraphs are accurate and expedient without exception. Figure 7 shows an intra-modular dominance subgraph detected in the J2SE JDK. The Java classes `Timer`, `TimerThread`, `TaskQueue`, and `TimerTask`, which are part of the `java.util` package, form a dominance subgraph. When the system is restructured these classes should be kept together because `Timer` and

`TimerTask` are always referenced together by classes positioned in other packages. `TimerThread` and `TaskQueue` are only used by `Timer`, and therefore they should not be separated from `Timer`.

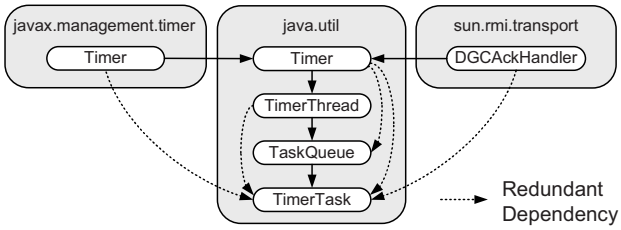


Figure 7. Detected Dominance Subgraph

C. Intra-modular Similarity Clustering

Structural similarity clustering allows comparing components based on afferent and efferent dependencies. Two patterns can be distinguished: support library pattern and facade pattern. Figure 8 (A) shows the support library pattern. The gray component is a support library that is used frequently. Figure 8 (B) shows the facade pattern. The gray component is the facade depending on a number of other components. In both cases, the white components resemble one another structurally although the dependencies of the support library and facade may be irrelevant for positioning. Therefore, it can be useful to remove these dependencies from consideration.

Clustering algorithms [27] group similar entities together. In order to quantify the similarity of entities a similarity measure is necessary. Schwanke [11] proposes a similarity measure to compare two procedures. This measure is applied to the MDG to compare components. By this means clusters of similar components that are part of the same module can be identified. These clusters are cohesive structures that are sustained during restructuring.

Figure 9 shows the similar components b and c that would be separated by metric-based restructuring techniques without similarity clustering. Part (A) shows the initial MDG. The similarity cluster $\{b, c\}$ is marked by a shaded oval. Without this cluster, b an c would be separated to improve metric values as shown in part (B). The metric values for the original configuration are: Coupling(M_1 and M_3) = 2, Coupling(M_2) = 4, Cohesion(M_1 and M_3) = 1, Cohesion(M_2) = 0. The alternative configuration created by a pure metric-based approach would have: Coupling(M_1

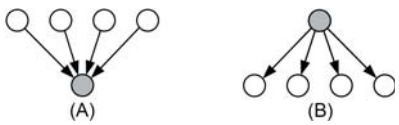


Figure 8. Similarity Clustering Motivating Example

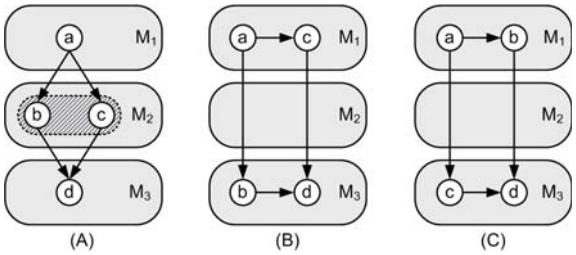


Figure 9. Intra-modular Similarity Clustering

and M_3) = 2, Cohesion(M_1 and M_3) = 0.5. Part (C) shows an alternative configuration with equal metric values. Therefore, using structural clusters prevents pulling apart similar components.

The similarity measure is based on features that are derived from the afferent and efferent dependencies of the components. Let a be a component that depends on the component b, then a has the feature "is-predecessor-of-b" and b has the feature "is-successor-of-a". Important features occur seldom, while common features emerge frequently. For example, the dependencies to a logging component are of little importance because they occur frequently throughout the system. Schwanke proposes to use the Shannon information content [28] from information theory as weighting factor for features. The formula for the weight of a feature used in this project is:

$$weight = -1 * log_2 \frac{\#feature\ references}{\#components - 1}$$

The components are clustered as follows: first an undirected graph is created. Each component is represented by a distinct vertex. At the beginning the graph has no edges. Then pairs of similar vertices are connected if the components they represent are part of the same module and if the similarity value reaches the similarity threshold, which has been detected in experiments as 0.8. At the end, the connected components of the graph are detected. Each connected component represents a cluster of similar components.

The collapsed dominance subgraphs and SCCs can affect the similarity of components. Therefore, similarity must be measured based on graphs without collapsed subgraphs.

The experiments show that only 61.3% of the classes are in the same package as their most similar peer. Therefore, restructuring a system by means of clustering the most similar components causes a high number of component moves and is therefore not acceptable. Seen from a different point of view, the positioning of 38.7% of the classes might be justified by other arguments, which are not detectable by similarity clustering.

Similarity clustering is a useful tool for detecting structures that should be maintained during restructuring. 12.3% of the analyzed classes could be assigned to intra-modular

similarity clusters using a high similarity threshold to limit the number of false-positive findings.

VI. Restructuring Phase

Although the main focus of this paper is preserving cohesive structures during the preprocessing phase, the accuracy of the restructuring phase can be improved as well. Besides using well known and widely accepted coupling and cohesion metrics, this paper introduces a third metric – coherence – as a further optimization criterion.

A. Coherence Metric Definition

Cohesion refers to the relatedness of a module's internal structure. We argue that an external viewpoint should also be used to analyze how the elements of a module contribute to a common purpose or objective. Therefore, we propose the metric coherence, which characterizes the functional cohesion of a module from an external viewpoint.

For a module m the function $Clients$ defines the components that are not part of m and that depend on components in m.

$$Clients(m) := \{c | c \in V \wedge c \notin m \wedge \exists a \in m : (c, a) \in E\}$$

Let m be a module and c a component not in m. The function ref specifies the components in m, which are used by c.

$$ref(c, m) := \{a | a \in m \wedge (c, a) \in E\}$$

The Jaccard Coeffcient [29, ch. 7] is used as a binary similarity measure to compare the usage patterns of the module's external clients. Let A and B be sample sets by which two entities are compared, then the Jaccard Coefficient is

$$S_{Jaccard} := \frac{|A \cap B|}{|A \cap B| + |A \triangle B|}$$

with the symmetric difference: $A \triangle B := (A \setminus B) \cup (B \setminus A)$. Coherence for a module m is defined as the sum of Jaccard Coeffcients applied to the module's clients:

$$Coherence(m) := \frac{\sum_c |ref(a, m) \cap ref(b, m)|}{\sum_c |ref(a, m) \cap ref(b, m)| + \sum_c |ref(a, m) \triangle ref(b, m)|}$$

with $c := \{a, b\} \subset Clients(m), a \neq b$. Coherence quantifies the similarity of usage patterns of the module's external clients. All clients of module m are pairwise compared using sets of referenced components in m.

Figure 10 shows three modules with varying coherence. The modules M_1, M_2, and M_3 are equal, but are used differently. Module M_1 has two clients each referencing to a different component in M_1. By means of the above proposed formula $Coherence(M_1) = 0/(0 + 2) = 0$. The value 0 corresponds to our intuitive comprehension of coherence because the clients use disjoint parts of M_1. If all elements of a module would contribute to one and the same purpose or objective, the clients would depend on component subsets with high intersection.

Module M_2 has three clients. `Client1` and `Client3` depend on different components. `Client2` depends on both components from m. In this case, coherence has a low value, but not zero, $Coherence(M_2) = (1+0+1)/((1+0+1)+(2+1+1)) = 1/3$.

Both clients of M_3 have the same usage pattern. In this case the module provides a coherent set of functions to other elements in the system. Consequently, coherence has the highest possible value, $Coherence(M_3) = 2/(2+0) = 1$.

A similar idea of using clients of a module for measuring the strength of its internal connections has been used in the Lack of Coherence in Clients (LCIC) metric [30]. LCIC has been used for identifying candidates for refactoring. The main difference between the coherence metric presented in this paper and LCIC is that LCIC uses the same approach as the Lack of Cohesion on Methods (LCOM) metric [31] while the coherence metric is based on a similarity measure. We argue that in contrast to the LCIC our metric depend less on the size of the module.

B. Coherence Metric Properties

This section validates the cohesion metric according to a property set similar to the set of properties proposed by Briand et al. [32] that must be satisfied by coupling and cohesion metrics.

Non-negativity and normalization property requires the existence of a real number Max such that the coherence of a module belongs to an interval $[0; Max]$. The metric coherence has the range $[0; 1]$.

Zero value property requires coherence to be zero, if the usage patterns of clients have nothing in common. Coherence is not defined for modules without clients. If a module has one client, the coherence is 1 per definition. Let us assume a module m has n clients, where $n = |Clients(m)| \geq 1$. If all clients depend on different components in m, then $|\forall a, b \in Clients(m), a \neq b : ref(a, m) \cap ref(b, m) = \emptyset$ and consequently $Coherence(m) = 0$.

Monotonicity property means that the coherence of a module is not decreased by adding an inter-modular dependency between a client and a component that is already

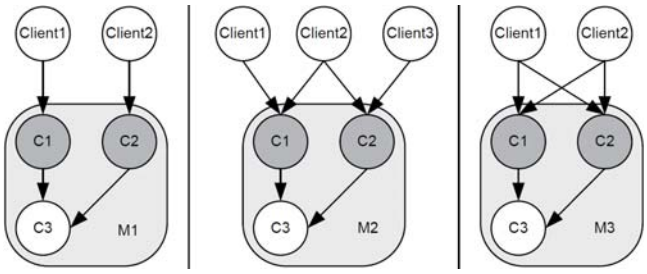


Figure 10. Illustration of Coherence Metric

referenced by one ore more other clients of this module. Let $\Gamma = (V, E, M)$ be an MDG, $m \in M$ a module, and $t \in m$ a component. Let $c_1, c_2 \in Clients(m)$ be two different clients of m. $(c_1, t) \notin E, (c_2, t) \in E, E' := E \cup \{c_1, t)\}$, and $\Gamma' = (V, E', M)$ is a second MDG. This property is satisfied, if the coherence of m in Γ is not greater than the coherence of m in Γ'. If in Γ the coherence of m is a/b with $a \geq 0$ and $b > 0$, then in Γ' the coherence of m is $(a + x)/b$ with $x \geq 1$ because there is at least one more common dependency leading to t. Adding a dependency (c_1, t) increases the similarities of the usage patterns of the clients of m.

Coherent modules property means that the coherence of a module created by merging two other modules having different clients is not greater than the maximum coherence of the two original modules. Let $\Gamma = (V, E, M)$ be an MDG. Let $m_1, m_2 \in M$ be two different, non-empty modules, $m_1 \neq m_2$, $|m_1|, |m_2| > 0$. Further, there is a module $m = m_1 \cup m_2, m \notin M$. Let $M' := M \cup \{m\} \setminus \{m_1, m_2\}$ be a set of modules and $\Gamma' = (V, E, M')$ an MDG. The coherence of m_1 and m_2 is:

$$Coherence(m_1) = \frac{r_1}{r_1 + d_1} \; Coherence(m_2) = \frac{r_2}{r_2 + d_2}$$

where $r_1, d_1, r_2, d_2 \in \mathbb{N}$ and $r_1 + d_1, r_2 + d_2 > 0$. In order to validate this property we have to show:

$$max\{Coherence(m_1), Coherence(m_2)\} \geq Coherence(m)$$

Let us assume

$$\begin{aligned} & Coherence(m_1) \geq Coherence(m_2) \\ \Leftrightarrow & \frac{r_1}{r_1 + d_1} \geq \frac{r_2}{r_2 + d_2} \\ \Leftrightarrow & r_1(r_2 + d_2) \geq r_2(r_1 + d_1) \end{aligned}$$

Let $x \geq 1$ be the number of usage difference of the clients that were added when merging m_1 and m_2. If $Coherence(m_1) \geq Coherence(m_2)$, then it is sufficient to show that

$$\begin{aligned} & Coherence(m_1) \geq Coherence(m) \\ \Leftrightarrow & \frac{r_1}{r_1 + d_1} \geq \frac{r_1 + r_2}{r_1 + r_2 + d_1 + d_2 + x} \\ \Leftrightarrow & r_1^2 + r_1 r_2 + d_1 r_1 + d_2 r_1 + r_1 x \geq r_1^2 + r_1 r_2 + d_1 r_1 + d_1 r_2 \\ \Leftrightarrow & r_1(r_2 + d_2) + r_1 x \geq r_2(r_1 + d_1) \end{aligned}$$

C. Coherence Metric Values

A manual inspection of the coherence metric values confirmed its plausibility. For example, the package `org.apache.tools.ant.taskdefs` is a conglomeration of different, partially related classes. This package has no clearly defined function, but containing all Ant tasks. The coupling is 978, cohesion 0.01, coherence 0.08. Other packages such as `org.apache.tools.tar` include a set of related classes. Its coupling is 8, cohesion is 0.2, and coherence is 0.62.

The Spearman's rank correlation coeffcient, p, was used to measure the pairwise correlation between the module size and coherence, and between cohesion and coherence. There is a very significant ($p\text{-}value \ll 0.05$), medium negative correlation between size and coherence (p = –0.52). Further, there is a very significant ($p\text{-}value \ll 0.05$), medium positive correlation between cohesion and coherence (p = 0.42).

The distribution of the cohesion and coherence values of the analyzed Ant projects is provided in Figure 11. The coherence metric shows a wider spectrum than cohesion. As a result of this stronger distinction of the projects regarding their coherence value, we argue, that although both metrics correlate, coherence can complement coupling and cohesion as a further optimization criteria.

VII. The Automatic Restructuring Toolkit

As a basis for the validation of our proposition, we developed a framework called "Automatic Restructuring Toolkit" (ART). This section comprises a short description of ART and its characteristic implementation aspects.

ART is a framework providing a collection of individual restructuring techniques, e.g. "detect SCCs" or "resolve component cycles" as introduced above. Figure 12 gives an overview of the most important modules within ART. All artifacts needed to analyze the structure of a software system are provided as XML documents, which are transferred into the internal format by the XML Handler and the Data Loader. The ArtGraph is an MDG that has been created from source code with the help of external tools, e.g., Classycle in case of Java code. The core engine of ART contains the restructuring techniques, which can be combined with each other via tasks to create restructuring proposals according to specific restructuring processes such as the process shown in Figure 3. The output of each task is the input of its following task. Intermediate results are stored, since they can be helpful to understand the final results.

The task-based approach allows changing the execution order of techniques without re-compilations. Therefore, techniques can be added or replaced, and configured with diverse parameters leaving the entire process of restructuring flexible. The composition of techniques can be accomplished through the exposed Java API, or by Ant scripting.

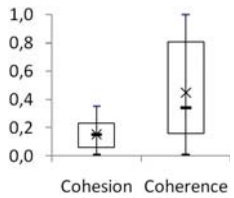


Figure 11. Cohesion and Coherence Metric Values

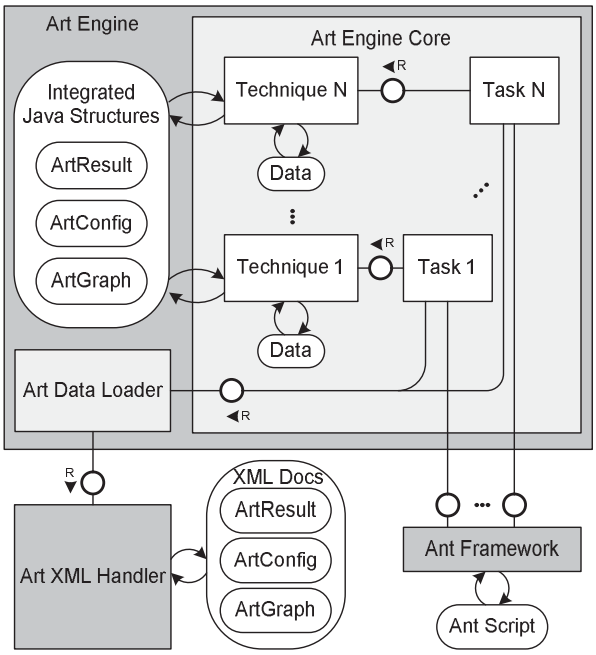


Figure 12. ART Architecture Overview

VIII. Conclusion and Future Work

Instead of radical changes, manageable changes are proposed by the presented approach. Existing cohesive structures are identified in the preprocessing phase and preserved during restructuring.

Eighteen Java open source projects have been analyzed for this work. The analysis shows that each module could be split on an average into 3.1 modules without introducing new inter-modular dependencies. 76% of all dependencies are inter-modular. Consequently, pure metric-based techniques would propose many component moves and split up those modules not changing the values for coupling, but improving the cohesion values.

Since only 61.3% of the components are in the same module with their most similar peer, pure similarity-based techniques would also propose comprehensive changes.

The results verify the usefulness of the proposed approach. During preprocessing 32.1% of the analyzed classes could be assigned to dominance subgraphs and 12.3% could be assigned to similarity clusters for preserving these structures during restructuring, thereby proposing less radical change.

The approach, however, does not include statements about the actual usage during runtime. Cases may exist where the usage patterns implying components to be similar on the basis of a structural analysis seldom or never occur during the runtime of the system. Runtime analysis and validating the above techniques from this point of view is a stream for future work.

More empirical research is necessary to analyze to what extent preserving cohesive structures supports or impedes finding better configurations. Future tests will show whether size and quality of the intra-modular similarity clusters can be improved with an extended similarity measure [12].

In future work the restructuring rules will be extended and combined with logical architectures mapped onto the physical artifacts of the analyzed systems to reduce the level of uncertainty of restructuring proposals.

A similar approach can be used during development of new software to identify positions for a new component while the rest of the system is kept unchanged.

Another field of future work lies in assessing different versions of a software system with our proposed approach, hereby validating the approach and the design decisions made during the evolution of the system.

Although no empirical evidence about the usefulness of the coherence metric has been investigated in this paper, we believe that introducing this additional criteria will allow preserving more cohesive structures. An experiment to test this hypothesis is part of future work.

Our results show that pure structural analysis can significantly contribute to the improvement of source code structure. From our point of view including analysis of the components' semantic meaning may even lead to further enhanced restructuring results. The validation of this assumption is subject of future work.

Acknowledgments

We thank Pieter Bloemendaal and Jakob Spies for their insightful inputs and suggestions concerning this work.

References

[1] K. Spichale, O. Panchenko, A. Bog, and A. Zeier, "Preserving Cohesive Structures for Tool-based Modularity Reengineering," in *Proceedings of the Fourth International Conference on Software Engineering Advances, ICSEA'09*, Porto, Portugal, September 2009.

[2] G. C. Murphy, D. Notkin, and K. J. Sullivan, "Software Reflexion Models: Bridging the Gap between Design and Implementation," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 364–380, 2001.

[3] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner, "Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures," in *Proceedings of the IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 1999, p. 50.

[4] "SAP – Business Managemant Spftware Solutions Applications and Services," http://www.sap.com, accessed July 1st, 2010.

[5] H. Zuse, *A Framework of Software Measurement*. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1997.

[6] R. Koschke, "Atomic Architectural Component Recovery for Understanding and Evolution," Ph.D. dissertation, University of Stuttgart, 2000.

[7] D. H. Hutchens and V. R. Basili, "System Structure Analysis: Clustering with Data Bindings," *IEEE Transactions on Software Engineering*, vol. 11, no. 8, pp. 749–757, 1985.

[8] S. Huynh, Y. Cai, Y. Song, and K. Sullivan, "Automatic Modularity Conformance Checking," in *Proceedings of the International Conference on Software Engineering*. ACM, 2008, pp. 411–420.

[9] M.-A. D. Storey, K. Wong, H. A. Müller, P. Fong, D. Hooper, and K. Hopkins, "On Designing an Experiment to Evaluate a Reverse Engineering Tool," in *Proceedings of the 3rd Working Conference on Reverse Engineering*. IEEE CS Press, 1996, pp. 31–40.

[10] J.-F. Girard, "ADORE- AR: Software Architecture Reconstruction with Partitioning and Clustering," Ph.D. dissertation, Univ. of Kaiserslautern, CS Dept., 2006.

[11] R. W. Schwanke, "An Intelligent Tool For Re-engineering Software Modularity," in *Proc. of the Int. Conference on Software Engineering*. IEEE CS Press, 1991, pp. 83–92.

[12] J.-F. Girard, R. Koschke, and G. Schied, "A Metric-Based Approach to Detect Abstract Data Types and State Encapsulations," *Automated Software Engineering*, vol. 6, no. 4, pp. 357–386, 1999.

[13] C. Jermaine, "Computing Program Modularizations Using the k-cut Method," in *Proceedings of the 6th Working Conference on Reverse Engineering*. Los Alamitos, CA, USA: IEEE CS Press, 1999, pp. 224–234.

[14] O. Goldschmidt and D. Hochbaum, *Polynomial algorithm for the k-cut problem*. Los Alamitos, CA, USA: IEEE Computer Society, 1988, vol. 0.

[15] B. S. Mitchell and S. Mancoridis, "On the Automatic Modularization of Software Systems Using the Bunch Tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193–208, 2006.

[16] V. Tzerpos and R. C. Holt, "ACDC: An Algorithm for Comprehension-Driven Clustering," in *Proceedings of the 7th Working Conference on Reverse Engeneering*. IEEE CS Press, 2000, pp. 258–267.

[17] R. C. Martin. (2000) Design Principles and Design Patterns. http://www.objectmentor.com.

[18] "Classycle: Analysing Tools for Java Class and Package Dependencies," http://classycle.sourceforge.net, accessed July 1st, 2010.

[19] J. L. Gross and J. Yellen, *Handbook of Graph Theory*. CRC Press, 2004.

[20] H. Melton and E. Tempero, "An Empirical Study of Cycles among Classes in Java," *Empirical Software Engineering*, vol. 12, no. 4, pp. 389–415, 2007.

[21] M. Fowler, "Reducing Coupling," *IEEE Software*, vol. 18, no. 4, pp. 102–104, 2001.

[22] D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[23] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a discipline of Computer Program and Systems Design*. Raleigh, NC, USA: Prentice-Hall, Inc., 1979.

[24] J.-F. Girard and R. Koschke, "Finding Components in a Hierarchy of Modules: A Step Towards Architectural Understanding," in *Proceedings of the International Conference on Software Maintenance*. IEEE CS Press, 1997, pp. 58–65.

[25] A. Cimitile and G. Visaggio, "Software Salvaging and the Call Dominance Tree," *Journal of Systems and Software*, vol. 28, no. 2, pp. 117–127, 1995.

[26] A. V. Aho, M. R. Garey, and J. D. Ullman, "The Transitive Reduction of a Directed Graph," *SIAM Journal*, vol. 1, no. 2, 1972.

[27] T. A. Wiggerts, "Using Clustering Algorithms in Legacy Systems Remodularization," in *Proceedings of the 4th Working Conference on Reverse Engineering*. IEEE CS Press, 1997, pp. 33–43.

[28] R. G. Gallager, *Information Theory and Reliable Communication*. John Wiley & Sons, Inc., 1968.

[29] M. Falk, F. Marohn, and B. Tewes, *Foundations of Statistical Analyses and Applications with SAS*. Basel, Swiss: Birkhäuser, 2002.

[30] S. Mäkelä and V. Leppänen, "A Software Metric for Coherence of Class Roles in Java Programs," in *Proceedings of the 5th international symposium on Principles and practice of programming in Java*. New York, NY, USA: ACM, 2007, pp. 51–60.

[31] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[32] L. C. Briand, S. Morasca, and V. R. Basili, "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, 1996.

Requirement-driven Scenario-based Testing Using Formal Stepwise Development

Qaisar A. Malik, Linas Laibinis, Dragoş Truşcan, and Johan Lilius
Turku Centre for Computer Science and Dept. of Information Technologies,
Åbo Akademi University, Turku, Finland.
Email: {Qaisar.Malik, Linas.Laibinis, Dragos.Truscan, Johan.Lilius}@abo.fi

Abstract

This article presents a scenario-based testing approach, in which user-defined abstract testing scenarios of the SUT are automatically refined based on formal specifications of the system under test (SUT). The latter are specified in a stepwise manner using the Event-B formalism until a sufficiently refined specification is obtained, which is then used to generate a Java implementation template of the system. The development of the specification is driven by the requirements of the system which are traced throughout the development and testing process. Abstract testing scenarios, provided by the user, are automatically refined following the same refinement steps used for the system specification. The sufficiently refined scenarios are then transformed into executable Java Unit Testing (JUnit) test cases, which are executed against the Java implementation of the SUT. During the described process, the requirements linked to the testing scenarios are propagated to JUnit tests. The main advantage of the proposed approach that it allows the developer to evaluate which requirements have been validated and to trace back the failed tests to corresponding elements of the formal specifications.

Index Terms

Scenario-based testing; Requirements Traceability; Event-B; Formal Refinement; JUnit;

1. Introduction

Formal development ensures that the developed systems are correct-by-construction. However, development of large and complex systems by formal methods exhibit several limitation including computation time and efforts it takes for the verification, and handling of low-level implementation details [1]. In general practice, formal methods are used to verify specifications of the system abstractly and implementation of the system is hand-coded while following formal specifications. In such cases, the implementation is written in an informal programming language. Since the implementation is no longer *correct-by-construction*, the resulting implementation needs to be tested.

Traditionally, testing has been performed manually by the tester carefully examining the implementation under test and then designing test cases. As the software became more complex, the resulting test cases have grown in numbers and complexity. This naturally has lead to the need to automate the testing process. Today, there exist several testing approaches that automate the testing process either completely or partially. These approaches try to achieve their goal by applying different means, i.e., code templates, scripts, formal and semi-formal software models etc. However, these approaches do not distinguish between different parts of the system that might be more or less important for overall system correctness. Therefore, it is important to test the functionality of the system according to user's requirements.

In this paper, we propose a testing methodology which uses user-specified testing scenarios in order to generate test cases. Our scenario-based testing approach can be seen as a kind of model-based testing where tests are generated from user-provided testing scenarios. The focus of this testing approach is on explicit identification of important behavior of the system that should be tested. The proposed methodology uses formal models of the system along with user-provided testing scenarios. These formal models and scenarios are mapped using the requirements. Later on, the formal models and scenarios are translated to Java and JUnit artifacts, respectively, while the requirements are also propagated to the JUnit test cases. The advantage of propagating requirements to executable test cases is that upon a a test case failure, it is possible to back-trace the failed requirement(s) to the corresponding parts in the model.

The work we present in this paper builds on and extends our previous work [9], [10] on scenario-based testing, where we have used formal models of the SUT based on the Event-B formalism. We have also proposed a collection of formal refinement techniques that can be used in the context of test generation. In this article, we elaborate our previous results and as well as compliment them by building a requirement traceability support. This allows us to keep track on how requirements are addressed by the specification at different abstraction levels and how they are propagated to the generated test cases.

To summarize, our proposed methodology encompasses the following:

- inclusion of requirements in the formal specification process and propagation of requirements to tests;
- traceability of requirements from tests back to formal specifications;
- identification of abstract test cases from formal scenario specifications;
- generation of Java templates of the system from sufficiently refined Event-B specifications;
- generation of JUnit tests from abstract test cases in the Communicating Sequential Processes (CSP) notation.

The organization of the paper is as follows. Section 2 provides necessary background on the modeling and programming languages used in this paper. In Section 3, we look in detail at the scenario-based testing process and present extended guidelines for modeling of Event-B specifications and of testing scenarios along with requirements. Section 4 gives overview of the tools we used for modeling, testing and measuring test coverage. In Section 5, we analyze and discuss the benefits and short-comings of our approach. Section 6, presents some related work in the area of research. Finally, Section 7 concludes the paper.

2. Background

In this section, we give overview of the languages and techniques we use for our scenario-based methodology.

2.1. Overview of Event-B

The Event-B [3] is a recent extension of the classical B-method [4] formalism. Event-B is particularly well-suited for modeling event-based systems. The common examples of event-based systems are reactive systems, embedded systems, network protocols, web-applications and graphical user interfaces. The language of the B-method and Event-B is based on set theory and predicate calculus.

As an example of an Event-B model, consider the following model (also known as machine) *M* with a context *C*. A context is considered as the static part of the Event-B specifications. It contains constants, sets and properties (axioms) related to these. On the other hand, an Event-B machine describes the dynamic part of the specification in the form of events (state transitions).
The context has the following general form.

CONTEXT C
SETS $sets$
CONSTANTS $constants$
AXIOMS $axioms$
END

A context is uniquely defined by its name in the **CONTEXT** clause. The **CONSTANTS** and **SETS** clauses define constants and sets respectively. The **AXIOMS** clause describes the properties of constants and sets in terms of set-theoretic expressions.
An Event-B machine has the following general form.

MACHINE M
SEES C
VARIABLES v
INVARIANT I
EVENTS
$INITIALISATION \ = \ \ldots$
$E_1 \ = \ \ldots$
$\ldots$
$E_N \ = \ \ldots$
END

The machine is uniquely defined by its name in the **MACHINE** clause. The **VARIABLES** clause defines state variables, which are then initialized in the **INITIALISATION** event. The variables are strongly typed by constraining predicates of the machine invariant *I* given in the **INVARIANT** clause. In addition, the invariant can define other essential system properties that should be preserved during system execution. The operations of event-based systems are atomic and are defined in the **EVENT** clause. An event is defined in one of two possible ways

$$E \ = \ \textbf{WHEN} \ g \ \textbf{THEN} \ S \ \textbf{END}$$

$$E \ = \ \textbf{ANY} \ i \ \textbf{WHERE} \ G(i) \ \textbf{THEN} \ S \ \textbf{END}$$

where *g* is a predicate over the state variables *v*, and the body *S* is an Event-B statement specifying how the variables *v* are affected by execution of the event. The second form, with the **ANY** construct, represents a parameterized event where *i* is the parameter (or a local variable) and *G(i)* restricts *i*. The occurrence of the events represents the observable behavior of the system. The event guard (e.g., *g* or *G(i)*) defines the condition under which event is enabled.

The occurrence of events represents the observable behavior of the system. The condition under which the action can be executed is defined by the guards. An event is known to be *enabled* when the guards evaluate to *true*. An event execution is supposed to take no time and no two events can occur simultaneously. When some events are enabled, one of them is chosen non-deterministically and its action is executed on the model state. When all events are disabled, i.e. their guards evaluate to false, the discrete system deadlocks. Then previous step is repeated to see if any events are enabled for execution.

The actions of an event can be either a deterministic assignment to the variables of the system or a non-deterministic assignment from a given set or according to a given post-condition. The semantics of actions are defined by their before-after (BA) predicates, where a BA predicate is a relation between *before* and *after* values of the event variables. BA predicates for specific cases of Event-B actions are given in Figure 1.

Action	Before-after (BA) predicate	Explanation
$x := F(x,y)$	$x' = F(x,y) \wedge y' = y$	*standard assignment*
$x :\in Set$	$\exists t.\ (t \in Set \wedge x' = t) \wedge y' = y$	*non-deterministic assignment from set*
$x :\mid P(x,y,x')$	$\exists t.\ (P(x,y,t) \wedge x' = t) \wedge y' = y$	*non-deterministic assignment by given post-condition*

Figure 1. The actions and before-after predicate

In Figure 1, x and y are disjoint lists of state variables, and x', y' represent their values in the after state. The $F(x,y)$ represents a function that provides a deterministic value for x' while y does not change its value. The Set represents any defined set while $P(x,y,x')$ is a post-condition relating initial values of x and y to the final value x'. The $:\in$ and $:\mid$ represent non-deterministic assignment operators operating on sets and predicates respectively.

To check consistency of an Event B machine, we should verify two types of properties: event feasibility and invariant preservation. Formally,

$$Inv(x,y) \wedge g_e(x,y) \ \Rightarrow\ \exists v'.\ BA_e(x,y,x')$$
$$Inv(x,y) \wedge g_e(x,y) \wedge BA_e(x,y,x') \ \Rightarrow\ Inv(x',y)$$

The main development methodology of Event B is *refinement* – the process of transforming an abstract specification to gradually introduce implementation details while preserving its correctness. Refinement allows us to reduce non-determinism present in an abstract model as well as introduce new concrete variables and events. The connection between the newly introduced variables and the abstract variables that they replace is formally defined in the invariant of the refined model. For a refinement step to be valid, every possible execution of the refined machine must correspond to some execution of the abstract machine.

Further details about modeling and verification in Event-B can be found in [3].

2.2. Overview of Communicating Sequential Processes (CSP)

In the following, we present a brief overview of Communicating Sequential Processes (CSP) [6] which is needed to model scenarios in our approach. In CSP, a system is modeled as a process, which interacts with the environment via a number of events whereas the occurrence of events is atomic.

In CSP, there are two basic processes: $STOP$ is a deadlocked process, and $SKIP$ is the terminating process. The process $a \rightarrow P$ can perform an event a and then behave as P. There are two choice operators used in CSP, namely, *external choice* ($\Box$) and *internal choice* ($\sqcap$) operators. In the case of external choice, $P_1 \Box P_2$, either process P_1 or P_2 is executed based on which event occurs first. On the other hand, the internal choice operator is used to model non-determinism, e.g., $P_1 \sqcap P_2$ can arbitrarily choose to behave as either P_1 or P_2.

The processes can be combined together in parallel or in sequence. For sequential composition, $';'$ operator is used. For instance, $P_1; P_2$ ensures that P_1 process executes before P_2. By parallel composition, we allow processes to interact/communicate with each other through the events they engage in. For parallel composition, $\|$ operator is used, e.g., $P_1 \| P_2$. Further details about CSP operators, its semantics and refinements can be read from [6].

2.3. Unit Testing

Unit testing aims at testing units of the program code, e.g., methods or modules, separately from each other. This kind of testing is performed by writing programming methods (called *unit tests*) that invoke the corresponding implementation methods under test. A unit test provides the needed input to the unit under test and evaluates its output before assigning any verdict about its success or failure. Unit testing ensures that the functionality of individual units are tested before these units are integrated to form a larger system.

In order to facilitate unit testing during the system development, unit testing frameworks have been developed for almost every programming language. A unit testing framework provides helper methods, reporting and debugging features to aid unit testing. In our scenario-based approach, we use Java Unit Testing (JUnit) [7] frameworks.

3. Scenario-based Testing Process

In our scenario-based testing process (Figure 2), the system is specified in a stepwise manner using the Event-B formalism until a *sufficiently refined specification* is obtained. The formal models are refined manually based on a set of guidelines which we will discuss in the following section. The sufficiently refined specification is then used to generate a Java implementation template of the system. The development of the specification is driven by the requirements of the system which are traced throughout the process, including to the generated Java code.

The testing scenarios are gradually developed from requirements. The first abstract scenario is provided by the user. This scenario represents a valid behavior of the abstract model present on the same level of abstraction. In short, we say that the abstract model *conforms to* or formally *satisfies* the abstract scenario. Later on, we refine this abstract scenario along the refinement chain of the system models until a sufficiently detailed scenario is obtained. In fact, this detailed scenario represents an abstract test case.

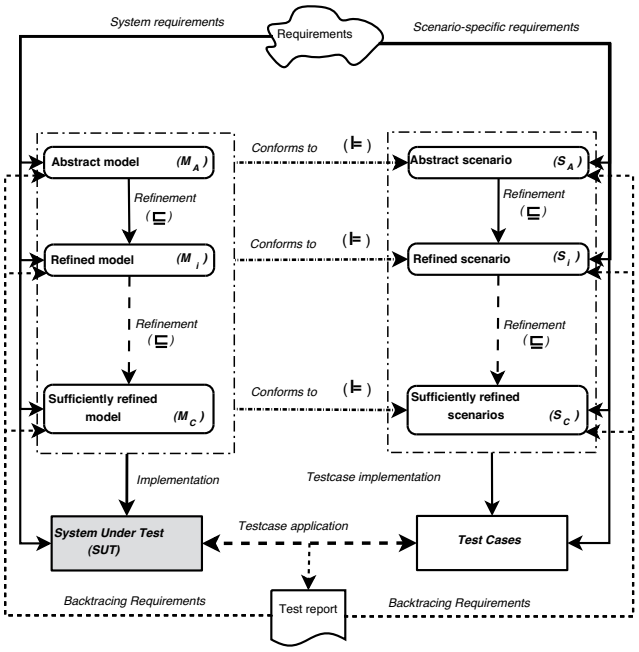


Figure 2. Overview of our scenario-based approach

3.1. Modeling Requirements

The sufficiently refined scenarios are then transformed into executable JUnit test cases, which are executed against the Java implementation. During the process, requirements linked to the testing scenarios are propagated to JUnit tests, where they are used for producing a test report. The approach allows us to evaluate which requirements have been validated and to trace back the failed tests to the formal specifications.

Usually the software systems are built according to informal requirements provided by user. The link between informal requirements and formal models is quite important in software development. The requirements are used for creating the initial specification of the SUT and also for refining this specification on the next level of abstraction.

In our approach, a stand-alone document specifying the requirements of the SUT in a structured manner is used. In this document, the requirements are described using *ID*, *Category*, *Title*, *Priority*, and *Description*, as shown in Figure 3. The hierarchy of the requirements is implemented using the requirement ID. For instance, requirement $\texttt{REQ1}-1$ is a sub-requirement of requirement $\texttt{REQ}-1$.

Throughout this paper we will use excerpts from a *Hotel Booking System*. For the sake of the understanding, we will briefly go through the main functionalities of the system which will be used for exemplification later on in the paper. The four main functional requirements of the system are: the system should allow the user to search for a room in the room database ($\texttt{REQ}-1$), to reserve the room ($\texttt{REQ}-2$), to allow him to pay for the reserved room ($\texttt{REQ}-3$) or to cancel an existing reservation ($\texttt{REQ}-4$).
The requirement $\texttt{REQ}-1$ is described as

Requirement : $\texttt{REQ}-1$
The system should be able to find a room of given type if it is available in the database and connection to the database is successfully established. In case of failed connection, an exception is reported.

Each requirement can be divided into several sub-requirements. For instance, ($\texttt{REQ1}-1$) and ($\texttt{REQ1}-2$) are given in the following.

Requirement : $\texttt{REQ1}-1$
The system should be able to find a room of given type if it is available in the database and connection to the database is successfully established.

Requirement : $\texttt{REQ1}-2$
The system should return an error message if connection to the database is not established successfully.

The requirement ($\texttt{REQ1}-1$), is further divided as

Requirement : $\texttt{REQ1.1}-1$
The system should be able to accept room type as an input.

Requirement : $\texttt{REQ1.1}-2$
The system should be able to connect to the database.

Requirement : $\texttt{REQ1.1}-3$
The system should be able to retrieve results.

These sub-requirements serve as basis for refining the Event-B model.

3.2. Using Event-B for Scenario-based Testing

In our approach, we create formal descriptions of the SUT starting from the requirements as shown in Figure 2. Subsequent refinements of the specification are preformed based on the sub-requirements of a given requirement. In order to be able to generate executable test cases, one needs to have available sufficient information regarding the inputs and outputs of the system. For this purpose, we structure the information about the inputs and outputs based on set of guidelines, following the basic refinement types we suggested in [10]. These basic refinement types are also

Requirement ID	Category	Title	Pr...	D	Description
REQ-1	Functional	FindRoom	1		The system should be able to find a room of given type if it is av
REQ1-1	Functional	FindSuccess	1		The system should be able to accept room type as an input.
REQ1-2	Functional	FindException	1		The system should return an error message if connection to the
REQ1.1-1	Functional	ValidRoomType	1		The system should be able to accept room type as an input.
REQ1.1-2	Functional	ConnectionSuccess	1		The system should be able to connect to the database.
REQ1.1-3	Functional	RetrieveSuccess	1		The system should be able to retrieve results.
REQ-2	Functional	ReserveRoom	1		The system should allow the user to reserve an available room.

Figure 3. Requirements specification excerpt

referred to as *controlled refinements*. The guidelines are used in a similar way for the development of both Event-B models and corresponding user scenarios.

3.2.1. Classification of Events. In order to identify information about the inputs and outputs of the system we classify the Event-B event types into *input*, *output*, and *internal* events, as follows:

Definition 1: The Events. Set of all events in the system, denoted by Σ, is divided into following subsets of:

- *Input* events denoted by ε^I
- *Output* events denoted by ε^O
- *Internal* events denoted by ε^τ

□

The *input events*, ε^I, accept inputs from user or environment. Apart from their input behavior, these events may take part in the normal functioning of the system. However, the input events do not produce externally visible output. The *output events* ε^O produce externally visible outputs. Finally, the *internal events* do not take part in any input/output activity. These events however, may produce intermediate results used by the events in ε^I and ε^O. The motivation of this classification is explained in next section, where we further divide our system into logical functional units.

3.2.2. Logical Units. As we develop our system in a stepwise manner, the main functional units of a system are already identified at the abstract level. Each of these abstract functional units are modeled as a separate logical unit, called *IOUnit*, in our Event-B models.

Definition 2. An IOUnit, U, consists of a finite sequence of events and has the following form.

$$U =< \varepsilon^I, \varepsilon^{\tau+}, \varepsilon^O >$$

Here ε^I and ε^O denote the input and output events respectively, and $\varepsilon^{\tau+}$ represents one or more occurrences of *internal* events.

□

It can be observed from the above definition that an *IOUnit* consists of the sequence of events occurring in such an order that the first event in the unit is always an *input* event and the last event is always an *output* event, with possibly one or more *internal* events in between. Moreover, an *IOUnit* can not contain more than one input or output event.

An *IOUnit* takes input and produces output, as the presence of the input and output events indicates. The classification of events, defined previously, helps us in identifying the inputs and outputs of each unit, and when combined, of the whole system. The motivation for this approach is the following. The developer of the SUT may decide to implement the system independently of the structure of an Event-B model. Indeed, it is sometimes hard to construct the strict one-to-one mapping between the events of the model and corresponding programming language units. For example, two events in a model can be merged to form one programming-language operation, or the functionality of an event in the model may get divided across multiple operations or classes in the implementation. However, for successful execution of the system, the interfaces of the model and implementation, i.e., the sequence of the inputs and outputs, should remain the same.

3.2.3. Example. Reserving a room in such the hotel booking system can be modeled as a sequence of events that occur in a specific order. On the abstract level, we may have only a few events, representing some particular functionalities of the system. For example, if we model requirements $\texttt{REQ} - 1$ to $\texttt{REQ} - 4$, each top-level user requirement will be implemented as one IOUnit. Consequently, there are four main IOUnits namely, *Finding* a room, *Reserving* it, *Paying* for it, and *Canceling* a reserved room. After we structure our model according to the guidelines described in Section 3.2.1, the resulting events and their sequence of execution can be seen in Figure 4(a).

As it can be observed, the main functional events are wrapped with the input and output events. For example,

the *Find* event is wrapped around with the *InputForFind* and *OutputForFind* events, where *InputForFind* and *Output-ForFind* are the input and output events, respectively.

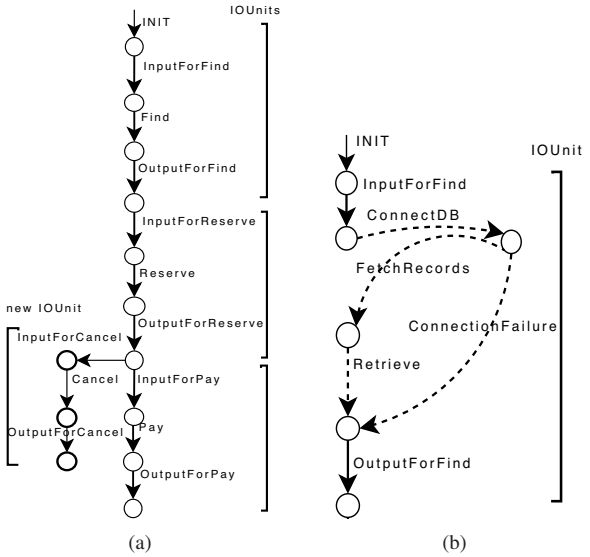


Figure 4. (a) Abstract System (b) Refined System

Within an IOUnit, we treat our main functional events as *internal events* (e.g., *Find, Reserve, Pay* and *Cancel*). Such events can be further refined, in one or more steps, consequently adding more *internal* events within the input-output unit. The refinement is performed according to the sub-requirements of the requirement that was the source of the IOUnit. For instance, the *Find* IOUnit in Figure 4(a) has been refined by applying successive refinements based on the requirement $\texttt{REQ} - 1$ and its sub-requirements, introduced earlier in Section 3.1, into four *internal* events depicted graphically with dashed line pattern in Figure 4(b).

The complete Event-B specifications of this example have been developed and proved using the RODIN [5] platform. In the final refined system, there was a total of 42 proof obligations. Out of these, 38 proof obligations were automatically discharged by the tool, while the remaining 4 needed manual assistance.

3.3. Modeling the Testing Scenarios

As previously mentioned, we use CSP to represent testing scenarios. The advantage of using CSP is twofold. First, a CSP expression is a convenient way to express several scenarios in a compact form. Second, since we develop our system in a controlled way, i.e. using the basic refinement transformations, we can associate these Event-B refinements with syntactic transformations of the corresponding CSP expressions. For instance, the abstract scenario S_A in Figure 2 is refined into scenario S_i, while considering the controlled refinement steps involved in refining the abstract model M_A to the refined model M_i. Similarly, this process continues until we get a sufficiently detailed, concrete testing scenario S_C to which the model M_C conforms.

3.3.1. Testing Scenarios. We define a testing scenario as a finite sequence of events occurring in some particular order. Since we have grouped the events in the form of logical IOUnits, our scenarios will also include a finite sequence of IOUnits. This means that the scenarios will include the same events as in the corresponding Event-B model. However, the scenarios must follow the same rules that were set for constructing IOUnits in the previous section, i.e.,

1) The first event in the scenario is always an *input* event;
2) The last event in the scenario is always an *output* event;
3) There can not be two input-type events in the sequence without any output event in between them, i.e., the following sequence in a CSP expression is not allowed;
$$< \cdots \rightarrow \varepsilon^I_k \rightarrow \varepsilon^I_{k+1} \rightarrow \cdots >$$
4) There can not be two output-type events in sequence without any input event in between them, i.e., the following sequence is also not allowed.
$$< \cdots \rightarrow \varepsilon^O_k \rightarrow \varepsilon^O_{k+1} \rightarrow \cdots >$$

Since the scenarios are defined on the abstract level, they lack details about the system inputs and outputs. The input details can be identified from the input event(s) of each IOUnit. For example, if an input event reads three input variables then these three variables become the inputs for the unit that the input event belongs to. The details about the inputs can be retrieved from the Event-B model since the model specifies the type, initial value and invariant properties for all variables.

The expected outputs are generated after the model is animated using the ProB model checker. For a given input of a test case, the ProB can animate the model and return the result, which is then saved as the *expected* output of the test case. This expected output can be then used to compare the values while testing the real implementation. The ProB model checker can only produce output values based on the available abstract values. For example, to test whether a room is available in the *Hotel Booking System*, ProB can check the expected result for a pre-defined set of inputs, while in the actual implementation this result might be retrieved from the database. Therefore, we need to define a mapping relation between the abstract and concrete data types. At the moment this mapping is provided manually. However, it is possible to automate its generation for the commonly used types, e.g., boolean and integers.

3.3.2. Example. In the case of the previously discussed *Hotel Booking System* example, there can be many possible testing scenarios. For example, if we want to test the *room*

finding, *reservation* and *paying* functionality, the corresponding abstract scenario expressed as a CSP expression would be as follows.

$$S_{0(A)} = \texttt{InputForFind?roomType} \rightarrow \texttt{Find} \rightarrow \texttt{OutputForFind!(roomId, anyException)} \rightarrow \texttt{InputForReserve?roomId} \rightarrow \texttt{Reserve} \rightarrow \texttt{OutputForReserve!reserveId} \rightarrow \texttt{InputForPay?reserveId} \rightarrow \texttt{Pay} \rightarrow \texttt{OutputForPay!payId} \rightarrow \texttt{SKIP}$$

After a number of successive refinements of event *Find*, we achieve the following scenario. For keeping the example simple, we only show the refinement of *Find* event which is also shown graphically in Figure 4(b).

$$S_0 = \texttt{InputForFind?roomType} \rightarrow \texttt{ConnectDB} \rightarrow ((\texttt{FetchRecords} \rightarrow \texttt{Retrieve}) \sqcap \texttt{ConnectionFailure})) \rightarrow \texttt{OutputForFind!(roomId, anyException)} \rightarrow \texttt{InputForReserve?roomId} \rightarrow \texttt{Reserve} \rightarrow \texttt{OutputForReserve!reserveId} \rightarrow \texttt{InputForPay?reserveId} \rightarrow \texttt{Pay} \rightarrow \texttt{OutputForPay!payId} \rightarrow \texttt{SKIP}$$

The variable `roomType` is the input for this IOUnit, whereas `roomId`, `anyException` are possible outputs. The variable `anyException` specifies if there was any exception, e.g., a connection failure.

Often, the subsequent event depends on the results of the previous ones. For example, the event `Reserve` takes `roomId` as an input from the previous event. It can be noticed that the refinement of the `Find` event has created two branches, one leading to successful case and the other to a database connection failure exception. When the above scenario is checked for conformance with the ProB model checker, it will be found that one can not proceed to `Reserve` if an exception occurred at the previous step. Therefore, this scenario will be split into two scenarios S_0 and S_1 given in the following.

$$S_0 = \texttt{InputForFind?roomType} \rightarrow \texttt{ConnectDB} \rightarrow \texttt{FetchRecords} \rightarrow \texttt{Retrieve} \rightarrow \texttt{OutputForFind!(roomId, anyException)} \rightarrow \texttt{InputForReserve?roomId} \rightarrow \texttt{Reserve} \rightarrow \texttt{OutputForReserve!reserveId} \rightarrow \texttt{InputForPay?reserveId} \rightarrow \texttt{Pay} \rightarrow \texttt{OutputForPay!payId} \rightarrow \texttt{SKIP}$$

$$S_1 = \texttt{InputForFind?roomType} \rightarrow \texttt{ConnectDB} \rightarrow \texttt{ConnectionFailure} \rightarrow \texttt{OutputForFind!(roomId, anyException)} \rightarrow \texttt{SKIP}$$

These scenarios, when sufficiently refined, are transformed into JUnit tests which will be discussed later in Section 3.5.

In the next section, we will discuss how Event-B model is used to generate an implementation template in Java.

3.4. Generating Java Implementation Templates

Once developed, we use the Event-B models of the SUT to generate Java implementation templates. We start by translating a (sufficiently refined) Event-B model into a Java class. As a result, Event-B events are translated to the corresponding Java methods. For our *Hotel Booking System* example, the excerpts of the respective Event-B machine and its implementation template are shown as follows.

MACHINE BookingSystemRef1
REFINES BookingSystem
SEES BookingContext
VARIABLES
roomType
...
INVARIANTS
...
EVENTS
Initialisation
act5 : roomType := Null_roomType
...
Event InputForFind ≙
Refines InputForFind
any
tt
where
grd1 : tt ∈ *RTYPES*
then
act1 : roomT ype := tt
act2 : inputForFindCompleted := TRUE
end
...
END

An operation in an Event-B specification consists of two parts. The first part contains the pre-condition(s) for the event operation to be enabled, while the second part consists of the actions that the operation performs. For every event in an Event-B model, we create two separate methods in the corresponding Java implementation representing the pre-conditions and actions respectively. The first method, which contains the pre-conditions of an event, returns the evaluation result in the form of a *boolean* value. The name of this method is pre-fixed with the string "guard_". The second method encapsulates the actions of the event. For example, for the *InputForFind* event from our *Hotel Booking System* example, the Java implementation methods are given in the Listing 1. As one can notice, the requirements attached to different IOUnits in Event-B are preserved during the transformation and included in the generated template (see line 19 of Listing 1).

```
1 public class HotelBookingSystem {
2
3     //class-level variables
4     public String roomType;
5     ....
```

```
6
7       public HotelBookingSystem(){
8           //initialization      ...
9       }
10
11      /* PreConditions/Guards for InputForFind event
            */
12      private boolean guard_inputForFind(String
            roomType){
13          return (roomType != null);
14      }
15
16      /* Implementation method for InputForFind
            event*/
17      public boolean inputForFind(String roomType)
18              throws PreConditionViolatedException{
19          //REQ1.1-1
20
21          boolean inputForFindCompleted = false;
22          if(guard_inputForFind(roomType)){
23
24              //actions ...
25
26              this.roomType = roomType;
27              inputForFindCompleted = true;
28          }
29          else{
30              throw new
31              PreConditionViolatedException("For
                    inputForFind");
32          }
33          return inputForFindCompleted;
34      }
35
36      //more Implementation methods for events
37      ....
38  }
39
40  class PreConditionViolatedException extends
        Exception{
41
42      public PreConditionViolatedException(String
            mesg){
43          super(mesg);
44      }
45  }
```

Listing 1. Implementation template example

Each Java implementation method, representing an Event-B event, first evaluates its pre-condition(s) by calling its "guard_" method. If the pre-conditions are evaluated to *false* then the exception `PreConditionViolatedException` is raised, otherwise the actions of the corresponding event are executed. The variables of an Event-B machine are translated into the corresponding class variables in Java. The type information for these variables can be retrieved from the *invariant* clause of the Event-B machine. We assume that a mapping relation between data types in Event-B and Java is provided by the user. For non-primitive data types, Java enumeration (`enum`) type can be used, e.g., to represent a set of finite elements. While most of the Java code can be automatically translated from Event-B constructs, the user can add more code statements according to his/her requirements. This means that the generated class actually constitutes a Java template.

In the next section, we will discuss how testing scenarios are translated into JUnit test cases.

3.5. Generating JUnit test cases from Scenarios

In Section 3.4, we presented the guidelines for generating implementation templates for Java. Once such a template is generated, we can generate the corresponding executable test cases from the scenarios. These test cases are represented as JUnit test methods.

Since our Event-B events are now presented as sequences of *IOUnits*, we write JUnit test cases to test these *IOUnits*. The $Find$ IOUnit from scenario $\mathtt{S}_0$ is represented as an abstract test case $\mathtt{T}_0$ as given in the following.

$$\mathtt{T}_0 = \mathtt{InputForFind?roomType} \rightarrow \mathtt{ConnectDB} \rightarrow \mathtt{FetchRecords} \rightarrow \mathtt{Retrieve} \rightarrow \mathtt{OutputForFind!(roomId, anyException)} \rightarrow \mathtt{SKIP}$$

For scenario $\mathtt{S}_1$, the abstract test case $\mathtt{T}_1$ would be expressed as following.

$$\mathtt{T}_1 = \mathtt{InputForFind?roomType} \rightarrow \mathtt{ConnectDB} \rightarrow \mathtt{ConnectionFailure} \rightarrow \mathtt{OutputForFind!(roomId, anyException)} \rightarrow \mathtt{SKIP}$$

For each of the test cases $\mathtt{T}_0$ and $\mathtt{T}_1$, a separate JUnit test method is implemented. The JUnit test method for $\mathtt{T}_0$ is shown in the Listing 2. In a similar way, JUnit test cases are generated for each IOUnit in the scenario.

```
1  public class HotelBookingSystemTest {
2
3      HotelBookingSystem bSys;
4      ...
5
6      @Before
7      public void setUp() throws Exception {
8          bSys = new HotelBookingSystem();
9      }
10
11     @Test
12     public final void T0(){
13         //REQ1-1
14
15         String roomType = ``Single";
16
17         try{
18             boolean v1,v2,v3,v4,v5;
19             v1 = v2 = v3 = v4 = v5 = false;
20
21             //calling methods of IOUnit
22             v1 = bSys.inputForFind(roomType);
23             v2 = bSys.connectDB();
24             v3 = bSys.fetchRecords();
25             v4 = bSys.retrieve();
26             v5 = bSys.outputForFind();
27
28             //assert statements (verdict)
29             assertTrue(``Successful completion'',
30                 v1 && v2 && v3 && v4 && v5);
31
32             assertTrue(bSys.resultSet.size() > 0);
```

```
33              assertTrue(bSys.anyException == false)
                    ;
34          }
35          catch(PreConditionViolatedException e){
36              fail(e.getMessage());
37          }
38      }
39  }
```

Listing 2. JUnit Test method for T_0

In the test case example shown in Listing 2, there is only one input parameter, i.e., `roomType`. However, in practice, there can be more than one input parameters. Generating all possible values for each parameter and then making all possible combinations of these parameters values may result in combinatorial explosion. In order to handle this problem, the input space partitioning [14] approach is used for test case generation. Information about each input variable is retrieved from the *invariant* clause and the *pre-condition* part of the *input* event. The *pre-conditions* and *invariant* clauses specify the type and possible restrictions (value ranges) for each variable. Using this information, the input space for each parameter is divided into equivalent partitions. Then from each partition, one value is selected to represent the whole partition. Combining the values of different variables from different partitions reduces the total number of input combinations needed for testing.

If a scenario involves multiple *IOUnits* in a sequence and JUnit test case for that sequence is desired, then JUnit test also includes calls to the relevant implementation methods of the the *IOUnit* involved. Moreover, the JUnit assert statements are also appended in the test case.

During the test case generation, the requirements associated to CSP specifications (at this stage called abstract test cases) are propagated to JUnit test cases, as Java comments in the code (see Listing 2–line 13). In addition, each requirement present in the requirement document listed in Figure 3 will be associated with the test cases that covers it. The approach allows one to trace which requirements have been covered and validated during the test execution. The approach will be discussed in more detail in Section 4.

3.6. Backtraceability of Requirements

Once the JUnit tests are run against the SUT a test report is produced. The report will tell which requirements have been covered by the selected set of test cases, which requirements have been left uncovered, and which requirements were not validated. Having the requirements associated to test cases and in the same time to different parts of both Event-B and CSP specifications, allows us to trace at which abstraction level a requirement was introduced and how it reflected in the generated test cases. Based on this analysis, one can identify the source of the error: either in the SUT implementation or an incorrect formalization of the requirement.

4. Tool Support

In this section we will have a brief overview of the tool chain used to support our scenario-based testing process described in Section 3.

Tool support for Event-B modeling and verification is provided by the RODIN platform [5]. RODIN is an Eclipse-based development platform providing effective support for mathematical proof and refinement. The platform is an open-source and further extendible with plugins. RODIN comes along with several useful plugins facilitating smooth and quick development of Event-B specifications. Some of its important features include interactive prover, proof manager, requirement manager and visual modeling. Figure 4 shows a screenshot of RODIN platform with interactive prover.

The consistency of Event B models as well as correctness of refinement steps should be formally demonstrated by discharging *proof obligations*. The RODIN platform automatically generates the required proof obligations and attempts to automatically prove them. Sometimes it requires user assistance by invoking its interactive prover. However, in general the tool achieves high level of automation (usually over 90%) in proving.

The JFeature Eclipse plugin [**?**] ver. 1.2 is used for specifying and managing the requirements of the system. A screenshot was presented in Figure 3. The requirements are specified in the Eclipse GUI and exported to a textual file. The requirements file is then used by the the Requirement plug-in [13] of RODIN for creating associations to Event-B models. In the Requirement plug-in, a parser parses the requirement document and lists individual requirements. Then, any requirement can be selected to be mapped to one or more Event-B elements. This mapping information is stored in a mapping file. Similarly, a separate mapping file is used for storing the mapping between requirements and scenarios. A requirement can be associated with a model element by first selecting a requirement in the requirement manager and then choosing the "Add Association" menu option, which appears after right-clicking the element to be associated. A caption of the Requirement plugin displaying the requirements of the Hotel Booking system is given in Figure 4.

Another important tool in our tool chain is ProB [12] animator and model-checker. Once an initial abstract scenario is provide and expressed in CSP, the generation of the refined testing scenarios is automatic. ProB is used to check conformance between the models and the scenarios. This satisfiability check is performed at each refinement level as was shown earlier in Figure 2. ProB supports execution (animation) of Event-B specifications, guided by CSP expressions. In fact, the available tool support is another motivating reason for representing scenarios as CSP expressions. Otherwise, regular expressions could also have served the purpose.

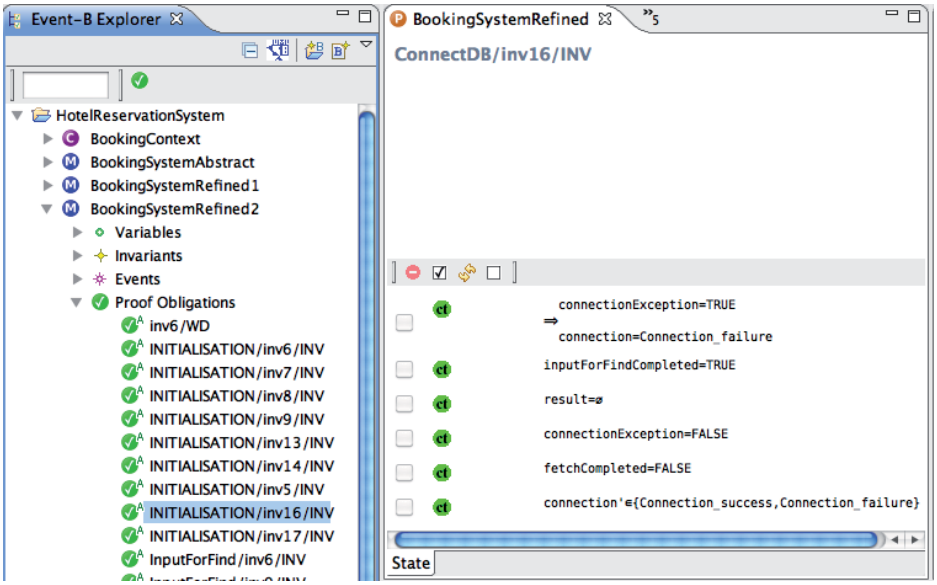


Figure 5. Screenshot of RODIN showing the formal proofs of the Hotel Booking System

Once the JUnit tests are generated they are executed using the JUnit plugin in Eclipse. The code coverage is performed by EclEmma [15], which is a freely available open source Java code coverage tool for Eclipse. With EclEmma, it is also possible to generate the test execution and coverage analysis reports. Figure 4 shows execution and code coverage of a unit test.

As mentioned previously, when the JUnit test cases are generated, the requirement document (listed in Figure 3) is updated such that each requirement is associated with the JUnit tests that cover it. Figure **??** shows a screenshot of the JFeature view in Eclipse. In the top frame, the requirements of the system are linked to one or many JUnit tests. In the left frame, the result of executing the JUnit tests is presented, whereas in the bottom frame, the JFeature test report on how different requirements have been covered by the test execution. In this concrete example, the `TestT1()` test failed, and since it was associated with requirement `REQ1-2 FindException`, the report presents the requirement as broken (with red background). The report presents also the *successful* requirements in green color and the `uncovered` requirements in yellow color.

For back-traceability of requirements for failed test cases, a manual approach is used. Basically, we examine the test reports for failed test cases. With the help of the RODIN Requirement plugin we identify in what parts of the formal specification a requirement was introduced and specified. Alternatively, we examine the Java code of the implantation and debug the code accordingly. Tracing requirements to back to code and formal specifications proved useful in identifying wrongful formalization of requirements or errors in the implementation of the SUT.

5. Discussion

In this section, we analyze our testing approach and discuss some related issues.

The presented test generation process produces test cases in JUnit, which is a well-known and widely used testing framework. The test cases are generated according to the user provided scenarios. More scenarios the user provides, the more code coverage we are likely to achieve. There are several good coverage measuring tools available that can be used with the generated test suites. We have tried EclEmma as described earlier in Section 4.

Furthermore, our approach has the distinguishing advantage that it also accommodates those changes which can not categorized and proved as formal refinement. Referring back to Figure 2, in some cases the model M_i may contain some extra functionalities or features, such as the incorporated fault-tolerance mechanisms, which were omitted or out of scope of the scenario S_A. These *extra features*, denoted by S_{EF}, can be added in the scenario S_i manually. The modified scenario $S_i \cup S_{EF}$ must be checked, by means of the ProB model checker, to satisfy the model M_i. We can then follow the same refinement process, now starting with $S_i \cup S_{EF}$, until we get a sufficiently refined scenario at level of the final model M_C.

Our approach also describes how one can generate Java implementation templates and the corresponding JUnit test cases. However, if for some reason, the user does not want to use the generated template, s/he can still use the JUnit test generation part to test his/her own implementation, provided that s/he has implemented the system keeping the operation interfaces consistent with the already generated JUnit tests.

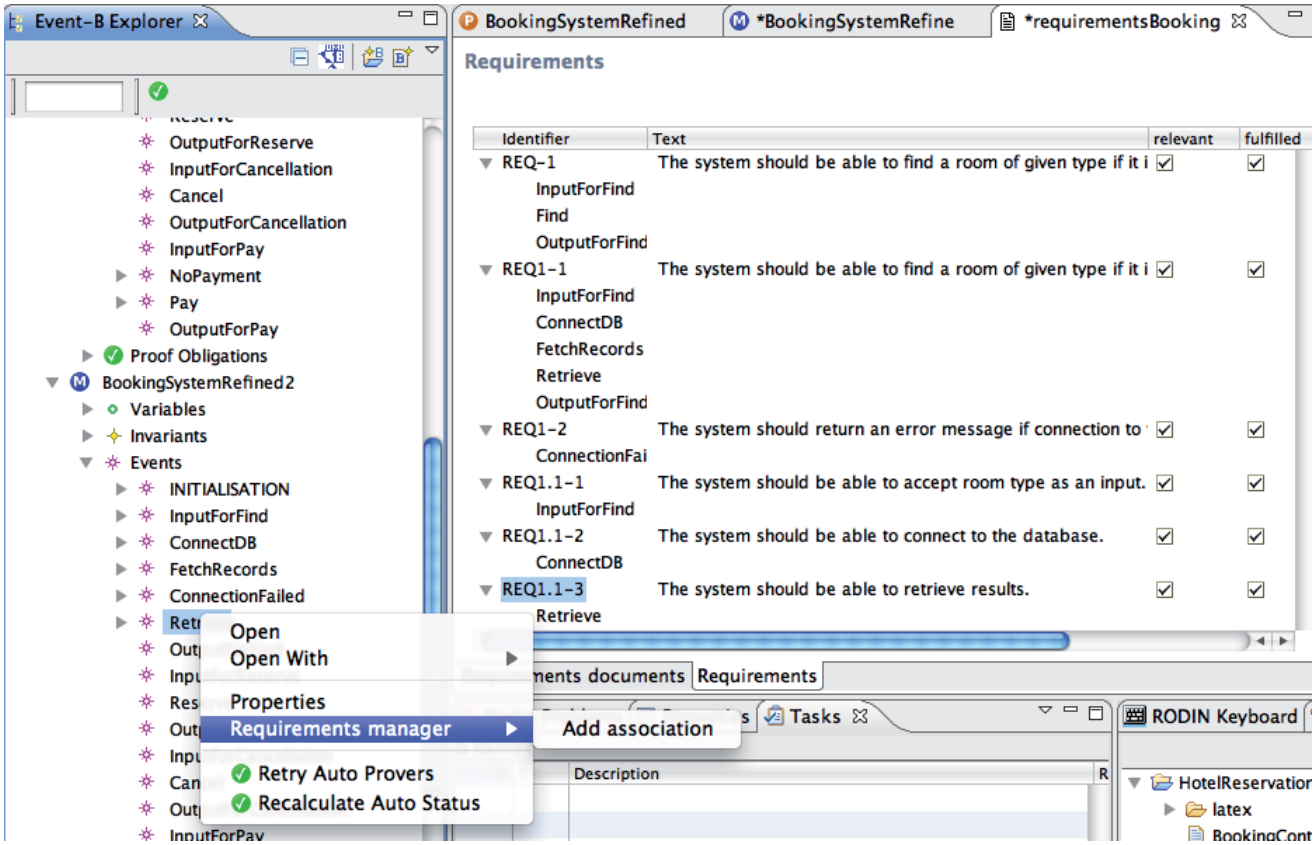


Figure 6. Screenshot of the Requirement plugin in RODIN showing requirements mapping

At the moment, we do not support translation of more complex pre-condition and invariant expressions from Event-B to Java. Namely, the existential and universal quantifiers are not covered. However, this can be achieved by using an approach similar to the one used in JML [16].

We do not explicitly support testing for *negative* scenarios i.e., the behavior that should not exist in the SUT. However, this kind of testing can also be accommodated if we model such *negative* behavior in our Event-B models as events and then provide testing scenarios covering those events. In order to show correctness, the JUnit tests, generated from these negative scenarios, should fail when applied on SUT.

6. Related Work

The jSynoPSys tool [17] performs scenario-based testing using symbolic animation of the B machines. This work defines a scenario-description language used to represent scenarios. However, authors do not provide any guidelines for the refinement of the specifications or scenarios. It is also not mentioned how scenarios will be transformed into executable test cases.

Nogueira et al. in [18] present a test generation approach based on the CSP formalism. The CSP models are constructed from use cases described in a pre-defined subset of natural language. The test scenarios are then incrementally generated as counter-examples for refinement verifications using a model checker. The main difference between their work and our approach is that we use Event-B to represent our system models and use CSP to represent testing scenarios. A model checker in our case is used to check the conformance between models and scenarios.

Stotts et al. in [19] describe a JUnit test generation scheme based on the algebraic semantics of Abstract Data Types (ADTs). The developer codes ADT in Java, while tests are generated for each ADT axiom. One of the advantages of this approach is that the formalism is hidden and the developer only needs to know Java to use this method. However, unlike our approach, in their case it would not be possible to mathematically prove any safety properties or to find deadlocks in the specifications.

In our earlier work [20], we presented the scenario-based testing approach for B models, where we designed an algorithm for constructing test sequences across different refinement [21] models. However, this algorithm is exponential in its nature thus limiting its practical applicability.

In our current approach, ordering of events are enforced

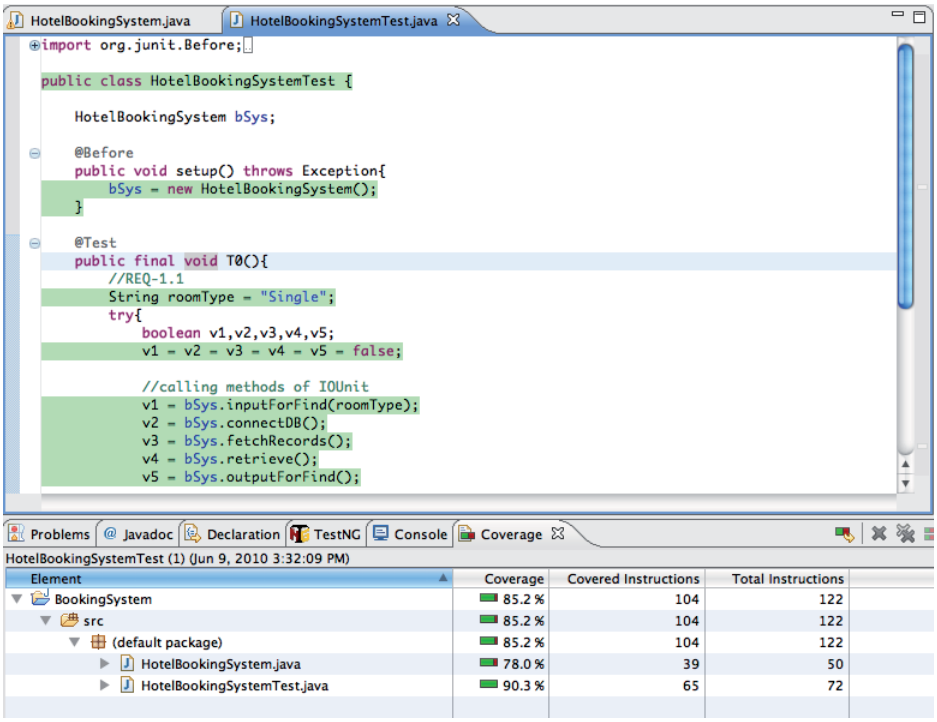


Figure 7. Executing Unit Tests and Measuring Code Coverage

by the guards and actions of the events. In [22], Iliasov has proposed a support of control flow as an explicit event ordering mechanism for Event-B models. The control flow of events resemble the notion of scenarios used by our approach. The difference between the two approaches is that we use a model-checker and an animator to verify existence of these scenarios, whereas in [22], the additional proof obligations are generated and then proved by a theorem prover. The control flow approach can be used as an alternative to the scenario-conformance steps, shown in Figure 2.

7. Conclusion and Future Work

In this paper, we presented a model-based testing approach using user-provided testing scenarios. These scenarios are first validated using a model checker and then used to generate test cases. Additionally, we have provided the guidelines for stepwise development of formal models and automatic refinement of testing scenarios. We also proposed an approach to generate Java language implementation templates from Event-B models. The abstract testing scenarios can then be used to generate executable JUnit test cases. Optionally, user can map informal requirements to the formal model and testing scenarios at different refinement steps. This mapping of informal requirements is extended till concrete test cases so that upon test case failure, these unfulfilled requirements can be back-traced into the model.

We believe that our approach is very scalable. It can help developers and testers to automatically generate large number of executable test cases. Generating these test case by hand would be very laborious and error-prone process.

As future work, we aim at providing graphical representation for the testing scenarios and their refinements. Moreover, at the moment, the mapping between abstract and concrete data types needs to be provided manually by the user. An automatic translation would be very helpful and time-saving in this respect.

In addition to that, we also intend to use the UML-B [23] formalism as the main modeling language in our scenario-based testing approach. UML-B is a new graphical language, which combines certain UML features with Event-B. UML-B is similar to UML but has its own meta model. The main advantages of using UML-B is that it provides an UML-like front-end to Event-B, which make the modeling language familiar to the majority of the developers. Moreover, it provides additional structuring of Event-B models in the form of UML classes and state-machines.

Another direction for our future work is to use the Next Generation Java Testing (TestNG) framework [8] for implementing the test cases. TestNG provides some important extensions to JUnit 4 framework, e.g., parameterized test methods.

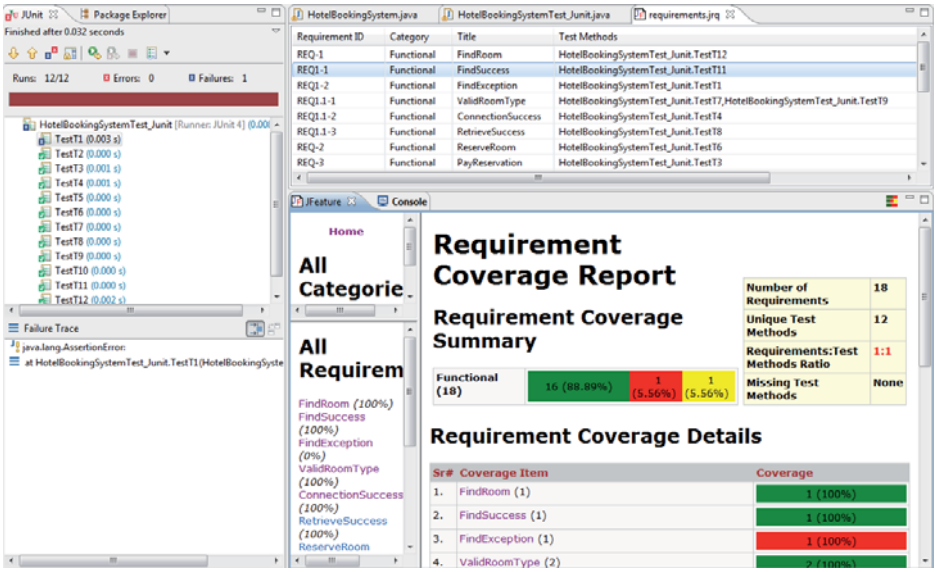


Figure 8. Requirement coverage report in JFeature.

References

[1] M. Hinchey, M. Jackson, P. Cousot, B. Cook, J. P. Bowen, and T. Margaria, "Software engineering and formal methods," *Commun. ACM*, vol. 51, no. 9, pp. 54–59, 2008.

[2] M. Utting and B. Legeard, *Practical Model-Based Testing*. Morgan Kaufmann Publishers, 2006.

[3] J. R. Abrial, *Modeling in Event-B: System and Software Design*. Cambridge University Press, 2010.

[4] J.-R. Abrial., *The B-Book*. Cambridge University Press, 1996.

[5] "Rigorous Open Development Environment for Complex Systems," iST FP6 STREP project, online at http://rodin.cs.ncl.ac.uk/.

[6] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, Inc., 1985.

[7] "JUnit 4," http://www.junit.org.

[8] C. Beust and H. Suleiman, *Next Generation Java Testing: TestNG and Advanced Concepts*. Addison-Wesley, 2007, http://www.testng.org/.

[9] Q. A. Malik, J. Lilius, and L. Laibinis, "Scenario-Based Test Case Generation Using Event-B Models," in *International Conference on Advances in System Testing and Validation Lifecycle (VALID 2009)*. IEEE Computer Society, 2009, pp. 31–37.

[10] Q. A. Malik, J. Lilius, and L. Laibinis, "Model-Based Testing Using Scenarios and Event-B Refinements," in *Methods, Models and Tools for Fault Tolerance, LNCS Vol. 5454*. Springer-Verlag, 2009, pp. 177–195.

[11] A. Roscoe, *The Theory and Practice of Concurrency*. Prentice Hall, 1998 amended 2005.

[12] M. Leuschel and M. Butler, "ProB: A model checker for B." Proc. of FME 2003, Springer-Verlag LNCS 2805, pages 855-874., 2003.

[13] "Requirement Management Plug-in for Rodin Platform," home page : http://wiki.event-b.org/index.php/Category:Requirement_Plugin.

[14] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008.

[15] "EclEmma - Java Code Coverage for Eclipse," http://www.eclemma.org/.

[16] G. T. Leavens and A. L. Baker, "Enhancing the Pre- and Postcondition Technique for More Expressive Specifications," in *In FM99: World Congress on Formal Methods*. Springer, 1999, pp. 1087–1106.

[17] F. Dadeau and R. Tissot, "jSynoPSys – A Scenario-Based Testing Tool based on the Symbolic Animation of B Machines," *Electron. Notes Theor. Comput. Sci.*, vol. 253, no. 2, pp. 117–132, 2009.

[18] S. Nogueira, A. Sampaio, and A. Mota, "Guided Test Generation from CSP Models," in *ICTAC*, 2008, pp. 258–273.

[19] P. D. Stotts, M. Lindsey, and A. Antley, "An Informal Formal Method for Systematic JUnit Test Case Generation," in *XP/Agile Universe*, 2002, pp. 131–143.

[20] M. Satpathy, Q. A. Malik, and J. Lilius, "Synthesis of Scenario Based Test Cases from B Models." in *FATES/RV*, 2006, pp. 133–147.

[21] R.-J. Back and J. von Wright, "Refinement Calculus, Part I: Sequential Nondeterministic Programs," in *REX Workshop*, 1989, pp. 42–66.

[22] A. Iliasov, "On Event-B and Control Flow," *Technical Report in DEPLOY Project*, 2010.

[23] C. Snook and M. Butler, "UML-B: Formal Modeling and Design Aided by UML," *ACM Transaction on Software Engineering Methodologies*, vol. 15, no. 1, pp. 92–122, 2006.

Integrating Quality Modeling in Software Product Lines

Joerg Bartholdt
Corporate Technology
Siemens AG
Munich, Germany
joerg.bartholdt@siemens.com

Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
roy.oberhauser@htw-aalen.de

Andreas Rytina
itemis
Munich, Germany
andreas.rytina@itemis.de

Marcel Medak
FNT GmbH
Ellwangen, Germany
marcel.medak@fnt.de

Abstract— **Due to the large number of possible variants in typical Software Product Lines (SPLs), the modeling of, explicit knowledge of, and predictability of the quality tradeoffs inherent in certain feature selections are critical to the future viability of SPLs. This article presents IQSPLE (Integrated Quality Software Product Line Engineering), an integrated tool-supported modeling approach that evaluates both qualitative and quantitative quality attributes without imposing hierarchical structural constraints. This contributes to better traceability; annotation; constraint enforcement; and quality attribute trade-off analysis - depicting overall product quality impacts on-the-fly. The approach is used in an eHealth SPL scenario, with the results showing that this approach is promising for effectively integrating quality attributes into SPL engineering in conjunction with (UML-based) artifacts.**

Keywords - variability; software product lines; quality modeling; feature modeling

I. INTRODUCTION

SPLE seeks to foster a systematic reuse of software assets for different but similar software products (typically within a domain). The general approach captures the commonalities and variability of the products in the product line and splits the development into domain (commonalities) and application (additional individual features for the final product). Products are created by integrating common artifacts (usually a platform) and optionally configuring them with product-specific artifacts [3][4].

Significant feature-oriented work and methodologies such as Feature-Oriented Domain Analysis (FODA) [5], FeatuRSEB [6], PuLSE [7] are well known for domain analysis and variability modeling for SPLs. However, for a potentially large set of possible variants, a significant aspect yet to be sufficiently addressed is the consequences of choices on the end qualities exhibited by a variant. An SPL engineer is faced with many more quality-related unknowns than a software engineer for a common single application software architecture. While various approaches for combining quality modeling with SPL engineering (SPLE) exist, previous work does not provide an integrated tool-supported approach with both qualitative and quantitative quality attributes (Q-attributes) that are explicitly considered in the variant derivation process without imposing structural constraints such as a hierarchical structure. In this problem space, the tool-supported IQSPLE method contributes trade-off analysis, traceability, annotation, and constraints enforcement of quality attributes during selection. Our previous work in [1] was extended to directly integrate solution space quality modeling with Unified Modeling Language (UML)-based artifact annotation support for variability and quality annotations as well as aggregated quality evaluation capabilities.

Considering the need for trade-off analysis, the distribution of quality attributes can vary significantly in software products, as shown in [8] that studied 24 ATAM (Architecture Tradeoff Analysis Method) evaluations. Such quality attributes are often not fully and systematically captured in prose. Even if formal models like the OMG (Open Management Group) UML-related QoS profile are used, an automatic aggregation ability is requisite to benefit most from a formal description. IQSPLE contributes methods and tools to immediately derive the quality attribute values of a given product instantiation.

Because qualities in SPLs often describe crosscutting concerns, the definition of qualities in the problem space is generally not linked to the solution space, resulting in a lack of traceability. IQSPLE contributes traceability via a formal linking that is used to calculate the quality attributes from the selection of product variations via the properties of assets in the solution space. This also supports the detection of quality issues for certain SPL variants that can be used in narrowing tuning efforts to the relevant solution artifacts.

Typical feature-oriented tooling concentrates on functional features; quality constraints are, if at all, modeled as simple XOR on features and thus remain purely in the problem space. IQSPLE enhances current feature modeling with support for the annotation of solution components with quality properties and arbitrary aggregation functions. By linking to the features, automatic constraint checks on given quality requirements can be executed. To enforce a common understanding and enable automatic calculation in the problem and solution space, a formal quality-capturing model for crosscutting as well as localized quality attributes is necessary.

This paper is structured as follows: Section II describes an e-Health scenario with the ensuing requirements that initiated the research. Section III describes the IQSPLE solution approach, which is then illustrated via an eHealth SPL scenario in Section IV. The solution is evaluated in Section V, with Section VI discussing related work. A conclusion and references follow.

II. E-HEALTH SCENARIO AND REQUIREMENTS

For illustration purposes, a simplified eHealth problem scenario that motivated this research is used. Patients are referred to other clinics due to their specialization (surgery, physical therapy, imaging, etc.). In the past, computed tomography images, clinical findings, etc., were given to the patient in the form of a printout or CD to take to the next treatment. This was error-prone, not all information was necessarily available at the next treatment location, and one was not sure that the data was current.

The eHealth scenario describes a clinic chain that wants to introduce SEPDE (software system for electronic patient data exchange) between organizations. The existing hospital information systems (HIS) are supplemented with SEPDE. The chain consists of ten hospitals where eight have the same HIS product and two have individual solutions.

Figure 1 shows a reduced feature tree of the SEPDE SPL. Integration with existent HIS, which is a key feature of the product line, can be achieved with three different techniques: web services, CORBA or message-based. The message-based approach allows for two different options: A high-throughput, but expensive commercial one or a slower, but license-free open-source variant.

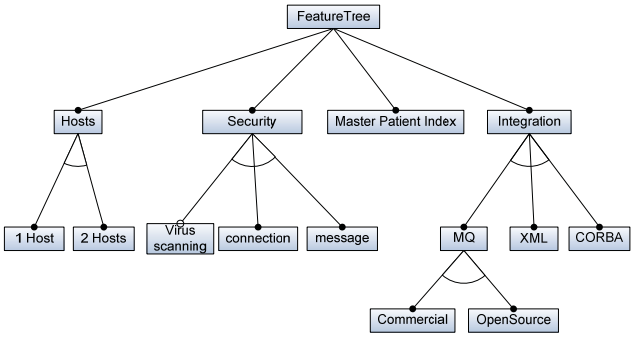


Figure 1. Conventional feature tree.

The development effort for creating the adapter to connect to existing HIS differs: XML-based web services are perhaps easier to develop, but carry greater performance overhead compared to binary protocols. The CORBA-based binary integration makes the final solution better in terms of latency and throughput, but development efforts are higher due to its complexity. A message-based approach increases integration flexibility and scalability, yet development complexity increases due to asynchronicity and lack of object-oriented remote method calls while longer latencies can be expected due to the centralized message broker.

For security and privacy, confidential connection (mapped to SSL connections), message-based encryption and signature (allows secure audit records), and virus scanning is supported. While all decrease the overall performance, the former two are necessary if no VPN exists between the sites; the latter additionally results in ongoing costs for continuous updates.

For higher availability of the system, a two-host-solution can be instantiated. The session state must then be replicated between the hosts so it exists should one of them fail.

This simplified scenario illustrates the relevance of quality attributes (e.g., performance, price, security) to the choice of specific features. The customer may not have exact requirements (e.g., 'use case 15' must be performed in less than 1.5 sec). However, the customer may be able to trade quality attributes against each other or functional features (e.g., 1.5 sec is achievable with the commercial MQ with a 5000€ license, whereas with the open-source solution it is 1.8 sec – which might be acceptable).

Each functional feature influences to some degree all system quality attributes, making the manual tracking of quality attributes difficult. Common feature trees contain functional features that are selectable individually (with a few constraints between each other), while system quality attributes are a crosscutting concern that changes with each (de)selection of a feature. Moreover, the quality correlations are often not expressible in simple constructs. Feature model constraints could (de)select features automatically, causing an entire set of quality attribute values to change at once.

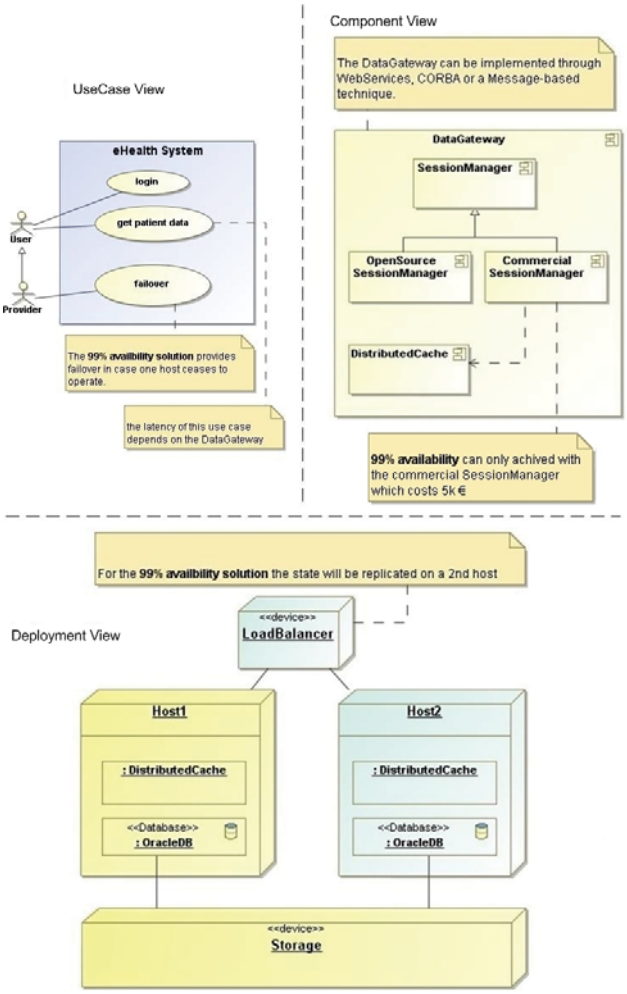


Figure 2. Example models affected by variability.

Qualities of the solution manifest themselves in different views. Many quality constraints can be described on the

component level, e.g., the usage of a specific session manager implementation correlates to the availability of the system, see Figure 2. Others become apparent in the deployment view, e.g., the availability of the solution depends on the number of redundant elements or even the existence of components like a load-balancer, but only make sense when more than one host exists. Certain use cases are valid only or change their nature depending on the features selected or on quality parameters. The resulting use case view can be used to generate development specification documents or user manuals. An additional benefit of a consistent and integrated usage of modeling of the views is the automatic traceability of features in models and specifications.

Considering non-trivial SPLs, the impact on quality attributes is not foreseeable for the SPL engineer, thus there is the need for methodology and tool support. As quality is usually not exactly defined by the customer beforehand and requirements may change, quality support is needed during the selection process.

A. Requirements

The following requirements on the methodology (M-requirements) and tool support (S-requirements) are deduced from the scenario:

M1: *qualitative values*. It must be possible to define quality values such as low, medium, or high in cases where a quantitative expression is not feasible or would be too expensive to measure. The quality attribute must be definable in the solution space as a property of an artifact and in the problem space as a non-functional requirement of the customer. Because non-functional customer requirements depend on design details, it must also be possible to link the qualities between the problem space and the solution space.

M2: *quantitative values*. It must be possible to define quantitative values (e.g., memory footprint, response times) in order to calculate the resulting quality values of the instantiation. Here also, the quality attribute must be definable in the solution as well as the problem space.

M3: *algorithms for calculating the resulting attribute values*. The methodology must support the definition of a calculation algorithm ("aggregation function") for the resulting quality value. This applies to quantitative as well as qualitative values (however, it may be less intuitive for qualitative values).

M4: *presentation as feature*. To support configuration and selection of qualities for a product instance, qualities can be presented in a separate quality tree like features, or alternatively within the feature tree, e.g., when qualities are relatively straightforward and the maintenance of a separate quality tree would seem contrived.

M5: *artifact annotation.* Quality attributes are treated as a first-class modeling citizens, and modeling and other artifacts can be annotated wherever appropriate. The annotation mechanisms shall follow a standardized mechanism to foster reuse and community acceptance.

S1: *calculate the quality values of a given variant*. The quality attributes that result from the selection have to be calculated (ideally on-the-fly), to give immediate feedback and let the users realize what changes in quality values a change in features results in.

S2: *determine the set of possible variants*. Given quality constraints during the selection process, the tooling shall determine the valid variants.

S3: *constrain the selectable features*. Quality attribute requirements shall be definable in advance and during feature selection, with those features not selectable whose selection would impair the required quality.

S4: *visualization of quality values*. From a customer perspective, multiple quality attributes may be of interest and may differ between customers. Thus, the tooling shall support an appropriate yet configurable visualization of the resulting quality values.

S5: *quality modeling integration in solution space*. Consistent, gap-less usage of modeling techniques, especially for quality modeling, leverages available tooling. All views on the solution space of the product line will more or less be influenced by quality attributes. The tooling must handle all of these in a consistent way, thus consistent meta-models must be applied on which transformers, generators, evaluators and viewers rely.

S6: *reuse of artifacts.* Automatic evaluation of quality attributes requires a formal description of quality properties and correlations. These formal descriptions can also be used for other purposes, e.g., for generating product specification documents, manuals, etc.

S7: *traceability support.* Generated artifacts should carry dependent tracing information, e.g., the "administration of multiple hosts" chapter in the manual depends on the selection of a high availability solution.

III. SOLUTION

To address the aforementioned requirements, IQSPLE integrates quality attributes in the solution artifacts, maps the feature and quality selection in the problem domain to the associated solution artifacts, and collects and evaluates the quality attributes. With appropriate aggregation functions, the quality attributes of the product instance can be automatically evaluated and displayed in the selection configuration. The various elements of the IQSPLE methodology are described below.

A. The IQSPLE Process

The process is depicted in Figure 3. SPL domain engineering involves:

1) *Requirements Analysis.* Through the analysis of the problem domain, common and variable feature and quality requirements are collected.
2) *Feature variability and quality variability modeling.* In addition to the typical feature modeling in a feature tree (e.g., using the Compositional Variability Management (CVM) framework, a separate quality tree is used to model the quality attributes and their value types, e.g., memory footprint in MB, latency in 'use case 15' in ms). It is assumed that components can be assigned quality

attribute values based on a specification or measurement. Note that discrete values need not be ordered.

The resulting quality tree can serve as a basis for selecting non-functional requirements. As appropriate, elements in the trees can be linked to other trees (e.g., a selection in the quality tree might deselect certain features in the feature tree) and to the UML models.

For automated synchronization support, UML vendor-specific APIs can be used to allow changes to the (quality-annotated) models to be automatically reflected in the trees (e.g., certain quality options might disappear if no longer supported in the solution models). This supports M1, M2, and M4.

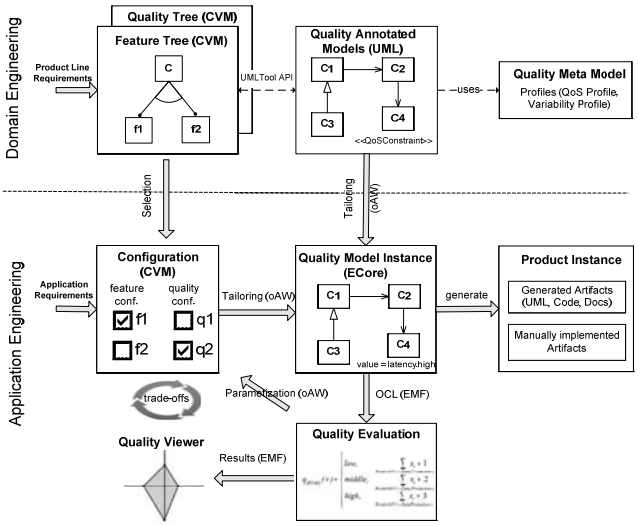


Figure 3. IQSPLE process.

3) *Modeling of solution artifacts including quality-annotations.* A quality meta-model such as the UML QoS Profile, variability profile, etc., is used to allow solution space models (i.e., software artifacts) to be annotated with quality attribute names and values. Note that each element can have multiple quality attributes assigned and these elements can be linked with features (e.g., memory consumption, use-case-specific latency, scalability properties), but not all components must be assigned all attributes. E.g., all components will affect memory, but not all will influence the latency in use case 15. This supports M1, M2, M5, and S5, S7.

SPL application engineering involves:

4) *Configuration.* A product configuration is selected based on feature and quality requirements (e.g., using CVM). This supports S2 and S3.
5) *Quality Model Instance generation.* Based on the configuration information and using UML tooling (e.g., openArchitectureWare (oAW)), a tailored Quality Model Instance is generated (e.g., in ECore).
6) *Quality evaluation.* OCL statements in the Quality Model Instance are used to evaluate the resulting qualities. A quality (aggregation) function is defined to support M3 and S1. To calculate the overall quality of the resulting product instance, the quality attributes of all selected components must be aggregated. In simple cases, this can be a *sum*-operation (e.g., memory footprint, latency) or *min*-operation (e.g., security behavior is as good as the weakest component). Complex operations are also possible, e.g., encryption depends on message length which depends on the selected features, so influence may be expressed in factors such as "encryption increases latency of use case 15 by 50%". For quality-based attributes, the aggregation function may count the number of occurrences, the most frequent wins, or calculates an average over ordered elements.
7) *Quality validation.* Based on the quality viewer, the qualities achieved are validated by the user or, based on tradeoffs, the configuration is adjusted as necessary. This supports S4.
8) *Product Instance Generation.* Once the configuration has been validated, artifacts are generated (e.g., UML models, code, documentation including specifications, and user manuals), and manual artifacts are implemented. This supports S6.

B. Variability

Negative variability. Negative variability starts from a maximal description (e.g., a UML model containing all possible elements of the product line) and deletes the elements that are not connected to selected features [2]. By this reduction, the final model of the selected product instance will be the result.

Depending on the selected features, model elements can be removed to derive different product instances. This is reflected in the model by tagging the different types with the stereotype `<<Variation>>`. The condition for which it is generated for the product instance is defined by the tagged value `{feature = "any feature condition"}`. This indicates to the generation process that the elements associated with the feature condition are generated if the condition evaluates to true, otherwise they are removed.

This is called negative variability since the starting point is a superset of the model definition and the unnecessary elements are stripped away according to the features selected. [2] discussed negative variability in class diagrams to model data structures of product lines and generate the data model for a selected product instance. In order to integrate this approach with quality modeling, this mechanism was extended for other diagram types. E.g., the results of the quality "scalability" will be seen on a deployment diagram that contains one, two, or more hosts. Another example is the deletion of a use case in a use case diagram because a certain feature was deselected. Use case diagrams can form the basis for product specification and manuals, which should also adapt automatically to selected features for the product instance.

Structural variability. Structural variability describes a change in the model dependent on some feature selection [2]. The element is already contained in the model, but its structure (type, cardinality, association) may vary. Structurally changing a UML model is achieved by adding

the stereotype <<modify>> to the elements that should be structurally changed and by setting predefined tagged values. Possible tagged values are, e.g., feature, type, cardinality, name, and initialValue.

In the resulting model, the corresponding property is changed. This can also be used to redirect associations by changing the type of the association.

C. Integrity Constraints

Constraint checking and their languages such as the OCL (Object Constraint Language) in UML are a known and powerful capability for assuring modeling correctness. Since SPLs typically support a large number of variations and quality aspects are typically crosscutting concerns that affect multiple models, constraint capabilities should be applied at the most appropriate points across the tooling.

For instance, feature/quality modeling constraints can be utilized to determine the validity of a certain combination of features or qualities (e.g., CVM provides a proprietary language to specify feature constraints). Instance tailoring constraints can be applied to check conditions (e.g., ensuring that the domain and feature/quality models are consistent) before or during the tailoring process as well as the artifact generation process.

D. Quality Functions and Annotations

In order to enable the evaluation of qualities of a product instance, mathematical functions are used to aggregate quality attributes. These functions are defined as relations between a valid variant *v* and a value *x*, where *x* represents the state of a specific quality.

$$q_i : v \mapsto x \tag{1}$$

To access attribute values of different artifacts, two additional functions are defined. The function *attribE* returns a single attribute value of a specific solution element (e.g., concrete component) and requires a valid variant *v*, a specific element *e* and the intended attribute *a*.

$$attribE : v \times e \times a \mapsto x \tag{2}$$

The second function *attribV* returns a vector of all attribute values of a variant that match the provided attribute name. This provides access to values and can be used if no exceptional conditions must be taken into account.

$$attribV : v \times a \mapsto \vec{x} \tag{3}$$

Note that there are no limitations on using different value ranges for aggregations, as long as a reasonable aggregation function for the quality can be determined. Numbers for calculating memory footprint are just as possible as using low, middle, high values to assess, e.g., security aspects.

As an example, privacy could be defined as follows:

$$q_{privacy}(v) = \begin{cases} low, & \sum_{\vec{x}=attribV(v,DataProtection)} x_i = 1 \\ middle, & \sum_{\vec{x}=attribV(v,DataProtection)} x_i = 2 \\ high, & \sum_{\vec{x}=attribV(v,DataProtection)} x_i = 3 \end{cases} \tag{4}$$

By modeling restrictions (Figure 1) on feature associations as constraints, M4 is supported. A constraint c_i is defined as a relation between a variant *v* and one element of the set *{true, false}*.

$$c_i : v \mapsto \{true, false\} \tag{5}$$

This allows a definition of, e.g., performance requirements based on predefined quality functions.

$$c_{performance}(v) = \begin{cases} true, & q_{latency}(v) \leq 300ms \\ false, & else \end{cases} \tag{6}$$

The constraints are used as a way to filter out remaining variants that violate given requirements. For deselection functionality support for S2 and S3, IQSPLE inspects each feature beneath a selection line in the feature tree and decides if a feature selection violates the given requirements. In order to decide feature selectability, IQSPLE distinguishes three fundamental cases. A feature is:

a) *not selectable* if it does not occur in any remaining variant,
b) *selectable* if it occurs in every variant,
c) or *in combination selectable* if it occurs in some variants but not in all.

Cases *a* and *b* are trivial. However not every consequence of a selection can be predicted, especially if there are still open selections that affect the fulfillment of constraints. Thus, IQSPLE uses Case *c* to indicate that a feature selection possibly can only be made dependent on further feature selections.

An example is shown in Figure 4. To illustrate the process, a constraint is defined that forces a selection of at least five features. Initially all variants are derived that fulfill the constraint. Subfigure (1) shows the root feature tagged in green, which means that at least one variant of the feature tree matches the constraint and that feature f_0 is contained in the set of the valid variants. Green features must always be selected.

In (2) f0 is selected and the selection cut is moved below f0. Here f1 is tagged in green and f2 in red. It is obvious that f2 will always be tagged in red because of the alternative relation to f1, which is an invalid selection since with f2 maximally four features can be selected. Consequently, f1 must be selected as shown in (3). In this case f3 is selectable and f4 in combination selectable. In (4) f3 has been selected and f4, f7 and f8 are in combination selectable. While the

current selection does not fulfill the constraint of at least five selected features, it is recognizable that a combination of the three remaining features would. If the current selection is a valid variant and fulfills the constraint, the selection of a blue feature is optional. For the case that a current selection does not fulfill a constraint, it is necessary to select further blue features.

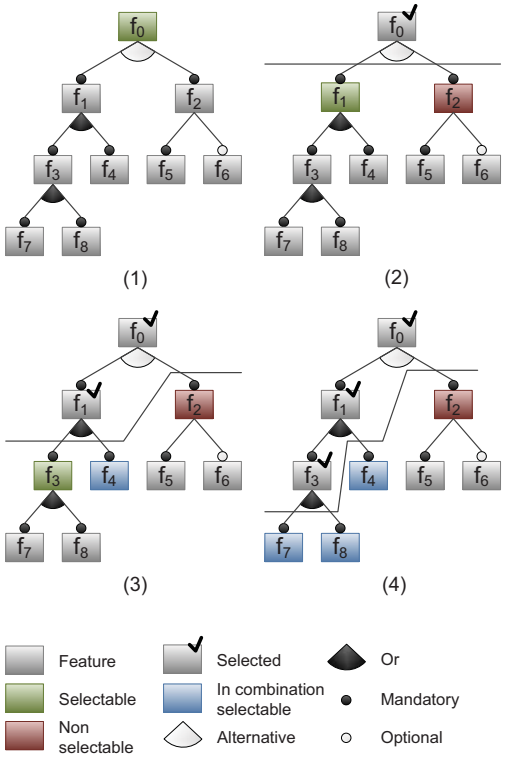


Figure 4. Selection process example.

Thus the IQSPLE process supports the handling of fixed requirements and, depending on the stakeholder's perspective, one can see either which subset of features still fulfill the requirements or if the selection of a certain feature does not.

In the current implementation, all quality functions were defined in OCL due to its expressiveness, UML integration, and standardization, as shown in Figure 5. In the diagram, four Quality of Service (QoS) characteristics are shown for latency, security, costs, and availability. To annotate certain constraints as quality aggregation functions, the OMG "UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms" [9] was extended with the `<<aggregateFunction>>` stereotype. Each `QoSCharacteristic` can have at most one `aggregateFunction`. To allow the retrieval of feature expressions from within a `QoSCharacteristic`, `QoSCharacteristic` inherits from the class `AbstractQoS`. Static methods are defined in the `QoSCharacteristic`, which can be additionally used for the definition of constraints.

The range of attribute values is determined by the `QoSCharacteristic`. For instance, costs are usually positive numbers that are summed, while usability could be either decreased by adding more components to administrate, or increased by a module that presents role-based administration user interfaces.

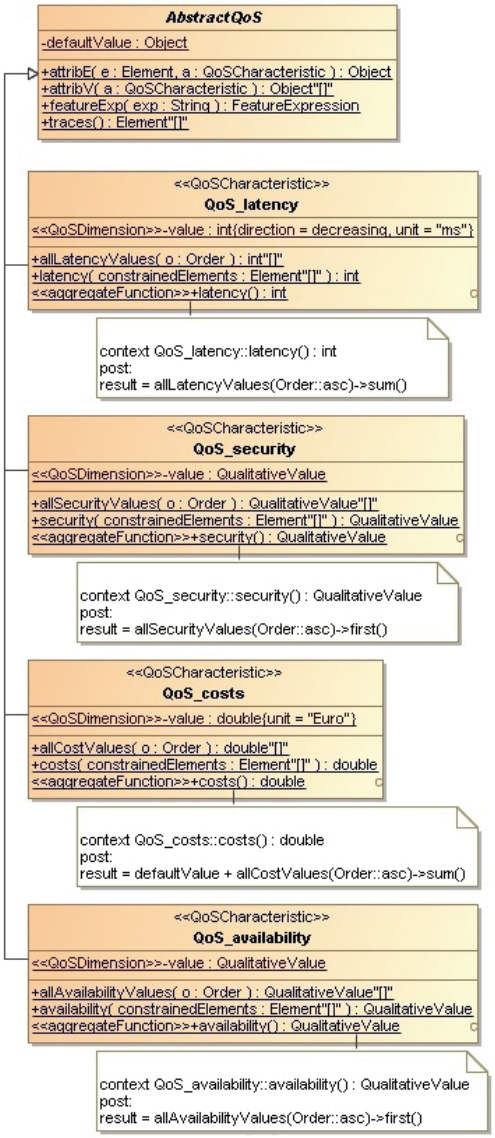


Figure 5. QoS characteristics with OCL quality aggregation functions.

In the OCL code of Listing 1, the aggregation function for latency is shown.

Listing 1

```
context QoS_availability::availability() :
QualitativeValue
post:result = allAvailabilityValues(Order::asc)
              ->first()
-- same query with the generic method attribV:
-- result = attribV(QoS_availability)
--  .oclAsType(QualitativeValue)
```

```
--  ->sortedBy(integerValue())->first()
```

The method `allAvailabilityValues(..)` returns an array of the results of all the availability constraints. The input parameter defines the sorting order of the array. The parameter `Order::asc` is for ascending order. The method call `first()` returns the smallest availability value. Alternatively, the call could have been realized with the method `attribV`, as described in the comments. The return values would then need to be cast to the specific type and sorted.

E. Quality Evaluation

The Quality Evaluator calculates the quality values, which utilizes the quality model instance created by the tailoring process. Impacts on qualities are evaluated by executing the OCL, thus calculating the quality attribute values of the selected artifacts for the product instance and aggregating them to an overall value. To support S4, after the quality value calculation and aggregation process, the assessed variant aggregated qualities are presented to the user as a spider chart as shown in the Quality Editor of Figure 6. This process is triggered every time a user changes a feature selection. The effect of a feature selection on particular qualities can thus be dynamically observed in the change to the chart during selection. This is helpful especially in trade-off situations.

Via this mechanism, the attribute values for a specific variant can be concretely defined in the solution space. For instance, a timeout configuration setting can be dependent on the combined latencies of the selected message component (commercial or open source).

Additionally, in order to ensure that the configured variants achieve the required qualities, during configuration the selectability of certain features and qualities are dynamically grayed-out (unselectable) based not only as hitherto on some abstractly modeled constraint/dependency between feature trees or tree elements, but on the impact evaluation of a specific feature or quality selection on variants (configuration feedback). For example, if a user defines that there is a budget of 10000€ for licenses, then all (aggregated) features are grayed out that are dependent on components that do not meet this requirement. At the time of de(selection) of a feature, the resulting configuration is verified against the quality requirements. In case the configuration does not meet the requirements, the user can choose alternatives. Currently a brute-force algorithm is used to recursively evaluate the remaining variants in the tree. Dependent on the number of variants, the automatic assessment of possible variants can take significant computation time. Therefore, typically not every variant is evaluated, but once a user-configurable limit of valid variants is found, the automatic assessment is halted and the alternatives are suggested to the user. If no valid variants are found within some user-defined time limit, the user is required to select additional features to further limit the variant space and thereby shorten the computation time.

As shown in Figure 6, the Quality Editor presents the aggregated quality values of a variant and provides information about the location and number of quality constraints that affect a certain quality attribute. For example, in Figure 6, the user has selected the quality attribute latency (get patient data) and, in the description which contains static and dynamic text, the user sees that most latency constraints are located in the component view.

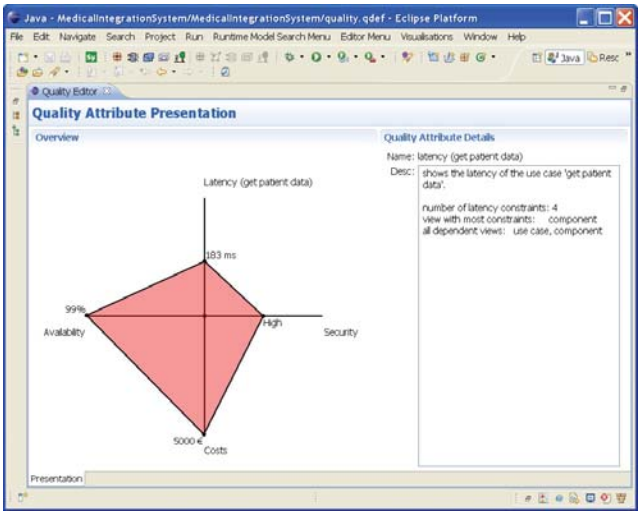


Figure 6. Quality editor.

The attribute values of a model element can also be dynamically calculated or become part of a variant, e.g., as a function of changes in the dependencies and selections.

IV. E-HEALTH SPL EXAMPLE

To illustrate IQSPLE, the E-Health example of Section II will be used applying the process described in section III.A.

1) Requirements Analysis

This is assumed to have been completed in this simplified example.

2) Feature variability and quality variability modeling

The feature tree from Figure 1 is now split into a feature tree (Figure 7) and a quality tree (Figure 8).

The single-host / dual-host features are removed from the feature tree. The customer should not select whether (s)he wants one or two hosts, which was modeled as a feature. Instead, the resulting quality of the solution is defined by the customer, in this case 95% or 99% availability. The dual-host solution is a solution and, as such, not of primary interest to the customer (it might be from an administrative point of view later).

The headline “security” is split. Virus scanning is definitely a customer visible (external) feature and, as such, located in the feature tree. The two other options “connection” and “message-based” were for deciding the security property if the communication can be unprotected, e.g., if the systems are interconnected via VPN. Connection-based security was implemented by using SSL, thus authenticating sender and receiver and encrypting the data in transit, but only a message-based signature results in

auditable messages because the messages together with the signatures can be archived as-is in an audit trail record and the signature can be verified again later. Thus, the customer is primarily interested in the resulting quality properties, e.g., if he has to obey specific regulations for auditable records, but does not care about the actual implementation necessary to achieve this. Security is an example for qualitative values (see requirement M2) with an order ("none" < "confidential" < "auditable").

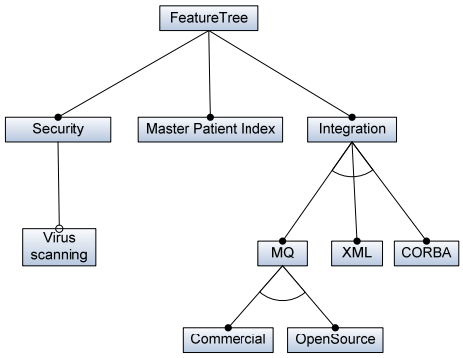


Figure 7. Feature Tree.

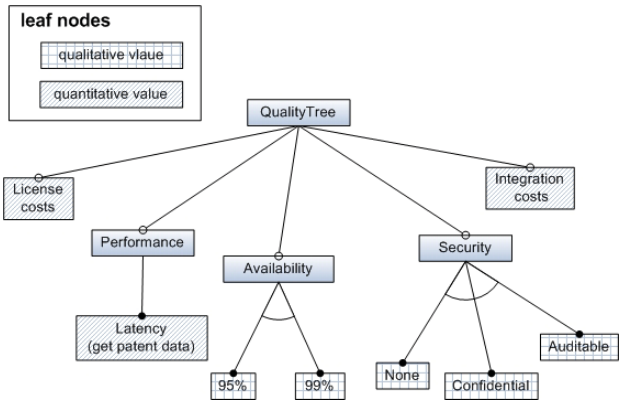


Figure 8. Quality Tree.

Note that a group of features can create additional value, e.g., if the http protocol is used for the user interface and DICOM protocol for image data retrieval, the complete solution is "confidential" only if both protocols provide confidentiality mechanisms, e.g., SSL. The aggregation function would be:

$$q_{security}(v) = \begin{cases} auditable & \text{HTTPcomp.security} == autitable \wedge \\ & \text{DICOMcomp.security} == confidential \\ confidential, & \text{HTTPcomp.security} == confidential \wedge \\ & \text{DICOMcomp.security} == confidential \\ none & else \end{cases}$$

3) Modeling of solution artifact including quality annotation.

The diagrams in Figure 9 through Figure 11 contain a (simplified) super-set of all possible artifacts annotated with constraints on the selected features and qualities. For example, Host2 in the deployment view is marked as `<<Variation>>`. The condition is `{featureExp = "99%"}`, thus the element "Host2" will vanish if the feature is not selected. The same is true for the load balancer. For details on negative variability, see [2].

Additionally, the quality attributes are annotated according to the OMG QoS profile [9] as described in Figure 5.

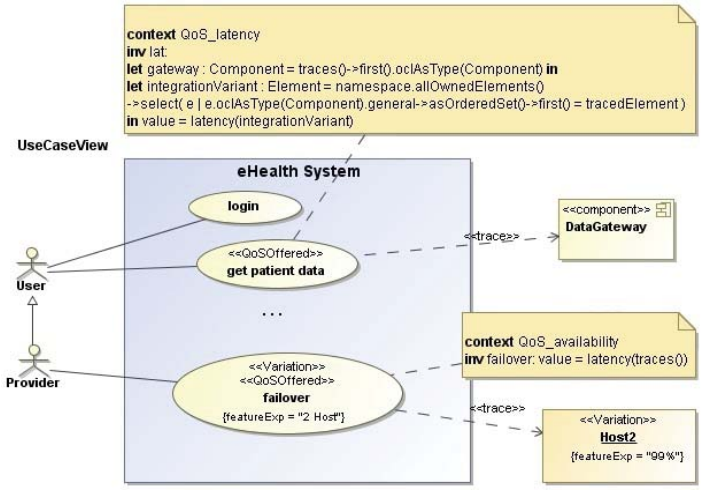


Figure 9. Quality-annotated use case model.

4) Configuration

For a fictitious customer, the product instance as shown in Figure 12 is selected. The customer chooses a message queue as the most flexible technology for integration, which is also the most future-proof variant. To save costs, (s)he selects the open-source implementation. For consolidation of patient records, (s)he wants the new system to have an MPI (Message Passing Interface). He/she selects virus scanning because all systems are connected to the internet, which is how the data exchange is routed.

From a quality perspective, (s)he emphasizes the availability of the system and selects 99%. Since no VPN is in place, (s)he selects confidential data exchange, but (s)he has no requirement for auditable signatures. The customer does not set any other quality constraints in the quality tree as shown in Figure 13.

5) Quality Model instance generation.

This step generates an internal representation of the quality model to be used during the quality evaluation. For performance reasons, this quality model instance only contains the quality constraints, quality aggregate functions, and feature expressions. Based on this simplified model, the Quality Evaluator calculates the quality attribute values for all constrained elements and then aggregates the resulting values to overall quality values.

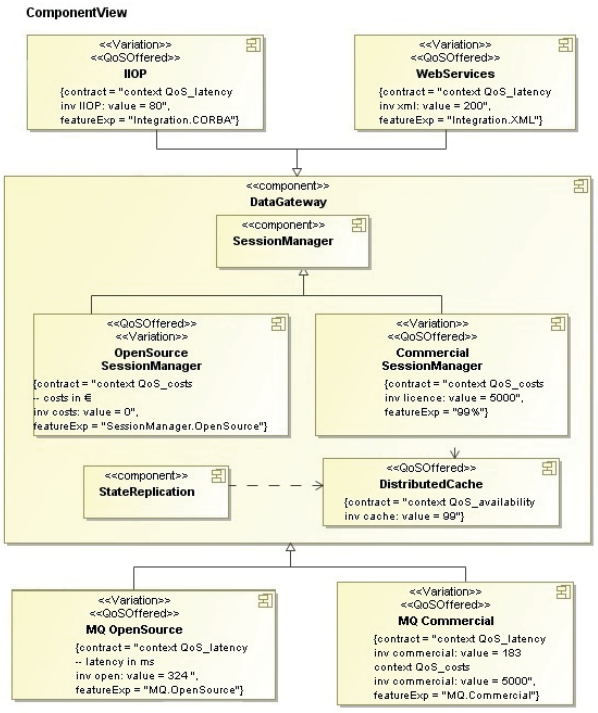


Figure 10. Quality annotated component model.

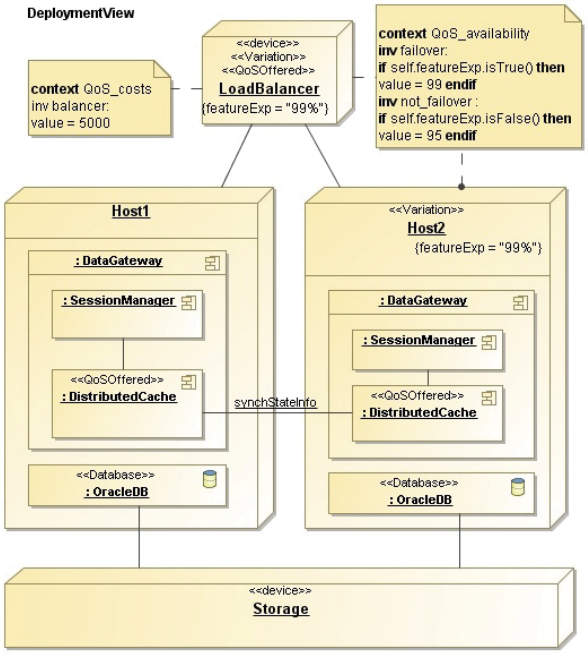


Figure 11. Quality annotated deployment model.

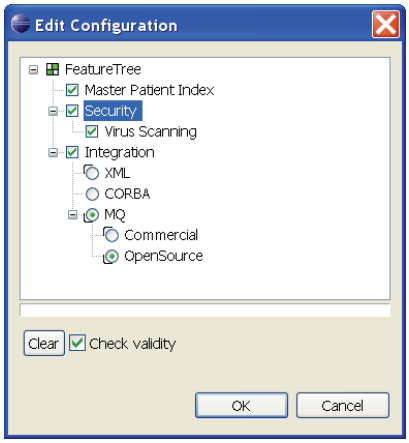


Figure 12. Selected product variant.

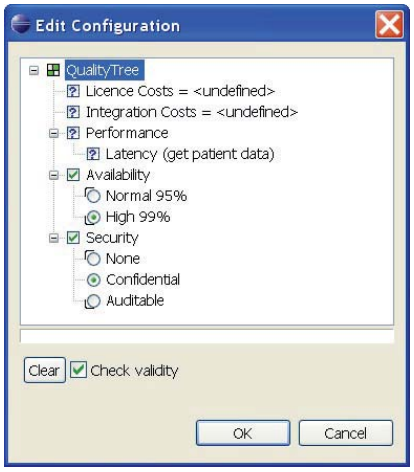


Figure 13. Initial quality tree defined by the customer.

6) Quality evaluation.

Due to the selection in the feature and quality tree, the aggregation functions can aggregate the overall quality attribute values of the product instance (see Figure 17 and 18 for the resulting product instance artifacts). E.g., the license cost (a quality attribute referred to by requirement M1) is determined by the sum of the existing artifacts on all diagrams (see formula in `QoS_costs: allCostValues(Order:asc)->sum()`):

Base	30,000
Load Balancer	5,000
Commercial session manager	5,000
Total	40,000

The evaluation of the quality attributes is shown in Figure 14.

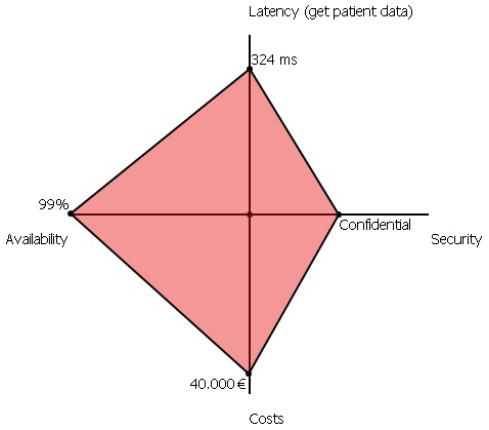


Figure 14. First quality spider chart.

7) Quality validation

The customer realizes that the latency of the use cases, represented by latency for "get patient data," is unacceptable. Modifying the messaging software provider from "open-source" to "commercial" results in the quality spider chart depicted in Figure 15, showing that the price increases to 45000€, but now the latency reaches an acceptable level.

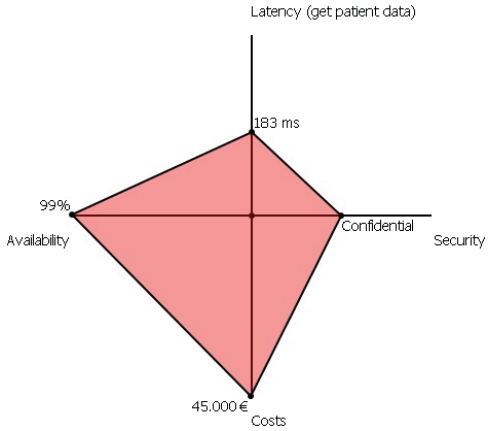


Figure 15. Final quality spider chart.

8) Product Instance Generation

The resulting artifacts are depicted in Figure 16, 17, and 18.

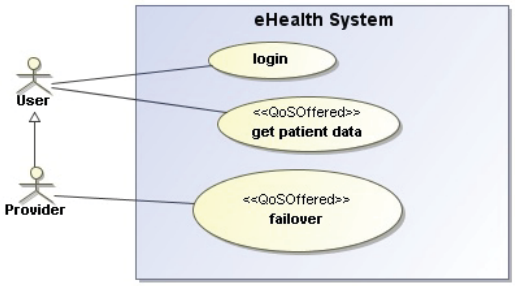


Figure 16. Tailored UseCase View.

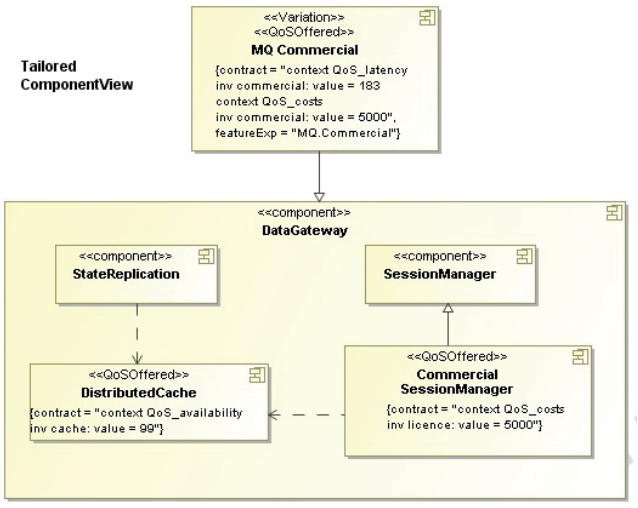


Figure 17. Tailored Component View.

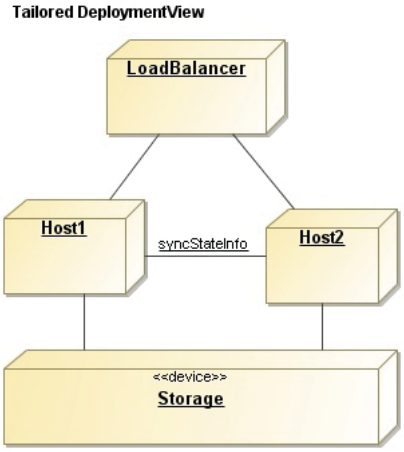


Figure 18. Tailored Deployment View

The use case Failover is included, thus the user manual will contain the section about administration and monitoring of the high availability solution.

The deployment diagram shows dual hosts with a load balancer required to fulfill the 99% availability. From the deployment diagram, the bill of materials can be derived and will now contain dual hosts and a load balancer, which has to be ordered from a 3rd party provider.

The component diagram includes only MQ as an integration provider and a distributed cache that is necessary for the multi-host deployment for sharing the state across multiple hosts.

The eHealth scenario exemplified how IQSPLE facilitates greater thought to and usage of quality in the domain and application engineering stages. First, customers can make decisions about required qualities based on facts instead of subjective estimations from the SPL engineer. Customers are also able to see how a decision affects particular qualities and if the consequences of a decision are acceptable. The SPL engineer benefits since thought to

system qualities are explicitly stipulated, which can help to improve the overall quality of the SPL architecture. In case a quality requirement is not fulfilled by the SPL, the engineer can track the different impacts on the quality and single out optimization opportunities. Additionally, it is possible to determine if a feature selection breaks any given quality requirements, which is done by filtering all possible variants based on existing quality requirements.

V. EVALUATION

Evaluation criteria considered beyond the M and S requirements were an initial assessment of scalability for current SPL development including the usage of OCL for on-the-fly quality evaluation.

The measurements were performed on an Dell Latitude E6500 (Core2Duo CPU @ 2.53GHz) PC with 3.5GB RAM running Microsoft Windows XP Pro SP3, Java JDK 1.6, Eclipse Galileo 3.5 (Modeling Distribution SR1), openArchitectureWare 5, CVM framework 0.6.0, Eclipse OCL2.0 Interpreter 1.3, and the Eclipse Modeling Framework 2.5. All measurements were performed 3 times and averaged.

For the first set of measurements, the tailoring process (as shown in Figure 3) for binding the variability of the Quality Annotated Model to derive a single variant (Quality Model Instance) was considered to determine the current practical scalability limitations of variation points, features, and resulting generation time. Table I and Figure 19 show a nearly linear correlation between a change in the number of variation points and the generation time when the number of features was held constant. An increase in the number of features also showed a nearly linear increase in the generation time. This result is explained by the iterations used in the generator code implementation for each variation point and for each feature. As to conditions, varying the number of Boolean conjunctions up to 20 for a variation point made no significant difference due to other inherent overheads.

TABLE I. TAILORING PROCESS TIME (MSEC) GIVEN A NUMBER OF FEATURES AND VARIATION POINTS

Number of Variation Points	Total Number of Features		
	300	*600*	*900*
1500	4302	6411	8709
3000	6771	11307	15526
4500	9453	16016	22563

The derivation and quality analyses of all variants of a SPL can be computationally expensive and make its usage impractical. Thus, measurements on the performance of the implementation for automated variant derivation were performed to determine its limits. The time to derive all variants of a quality-annotated model with 100 QoSConstraints (simple additions) was measured for a binary structured feature tree of or-relations, while the number of features was increased from 10 to 32. The results in Table II and Figure 20 show that, given a current PC, the quality could be evaluated with up to 20 features and 2000 resulting variants on-the-fly with results in less than a minute.

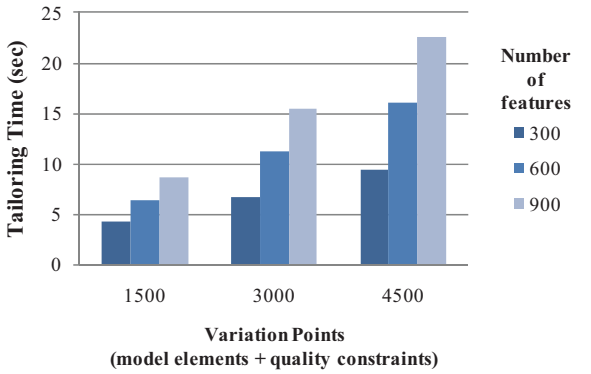


Figure 19. Tailoring process time (sec) vs. variation points and number of features.

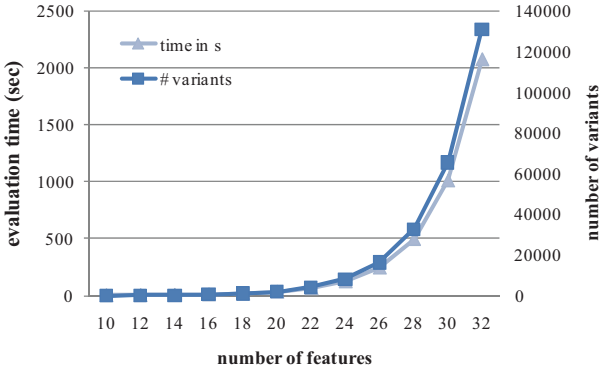


Figure 20. Quality evaluation time (sec) vs. number of features and variants.

TABLE II. QUALITY EVALUATION TIME GIVEN A NUMBER OF FEATURES AND VARIANTS

Features	Variants	Time (sec)
10	63	4
12	127	5
14	255	7
16	511	11
18	1023	18
20	2047	35
22	4098	65
24	8191	126
26	16383	247
28	32767	497
30	65535	1012
32	131071	2075

Based on these results with available tooling and systems, it is currently feasible to use and have the benefits of IQSPLE in industrial settings. The evaluation showed that performance for single variant quality evaluations was sufficient for usage today, but scalability issues were found

with handling large variation sets. When the quality evaluation of a large number of variants is desired, it is recommended that quality evaluations be executed in offline batch mode, and the results for all variants stored in a database for later access in order to enable tradeoff analysis to take place without encumbrances. Optimization possibilities include evaluating boundary constraints on the quality function properties to avoid further calculation overhead, e.g., aborting a calculation when a boundary value is exceeded.

VI. RELATED WORK

.Related work includes the Feature-Softgoal Interdependency Graph (F-SIG) approach [10], which supports quality modeling in the domain analysis phase. Its lack of support for quantitative values results in only imprecise quality assessments of a variant. The Extended Feature Model (Ext-FM) [11] applies a Constraint-Satisfaction-Problem approach and allows both quantitative and qualitative values to determine the set of matching variants. However, it requires a hierarchical modeling of quality attributes that restricts the possible set of quality dependencies that can be modeled. The Integrated Software Product Line Model (ISPLM) [12] integrates an implementation model that supports quantitative Q-attributes, yet it does not specify how these Q-attributes are to be utilized for a Q-assessment or set selection of variants. The Q-ImPrESS project [13] aims at modeling quality attributes at an architectural level. A reverse engineering process is used to derive component models which than are evaluated for quality prediction. It lacks support for modeling variation points in the problem space and in the solution space. Quality-driven Architecture Design and Analysis (QADA) [14] is a SPL architecture design method supporting traceable product quality and design-time quality assessment. Qualitative quality requirements are treated as architectural style(s) and patterns, and quantitative quality requirements as the properties of individual architectural elements. While addressing not only the conceptual architecture but also the concrete architecture, it does not produce implementation artifacts. It uses quality viewpoints [15] and conforms to OMG's Model-Driven Architecture (MDA) and IEEE 1471 [16] and uses UML profiles. [17] describes a tool chain that supports QADA including quality evaluation and test result imports. The protégé ontology tool is used for quality attribute definition, whereas IQSPLE encourages the use of feature tree tooling (e.g., CVM) for quality attributes due to its simplicity and prevalence. A comparison of these methodologies is shown in Table III with "Y" meaning yes and "N" meaning No.

COVAMOF [18] supports the modeling of dependencies between a set of variation points, however it does not explicitly address quality modeling.

While the Attribute Driven Design (ADD) method [19] supports the explicit articulation of the quality attribute goals for SPLs, it is narrowly focused on the definition of the conceptual architecture.

With regard to addressing SPL variability and quality annotation in UML models, the comparison matrix in Table IV shows an assessment of related SPLE approaches for a subset of requirements. 'Quality annotation' refers to the capability of annotating existing artifacts in SPL with quality information. 'Requirements analysis' refers to the support of the requirements process from elicitation to documentation, while 'feature model integration' means the usage of feature models as a basis for the approach. 'Negative variability' and 'structural variability' are defined in [2] and describe the means to define SPL artifacts and transfer them into product instance artifacts. 'UML compliant' refers to the restriction of using standardized modeling based on UML including OCL, stereotypes, tagged values, etc., an influential factor for industrial adoption of an approach. Modeling artifacts can contain 'modeling constraints' (i.e. constraints defined in the solution space) and 'configuration constraints' (i.e. interdependencies of features in feature trees). Constraints might become expensive to evaluate, thus a separation of constraints that can be evaluated on-the-fly and constraints that will only be checked during the generation process might become necessary, evaluated under 'checks during generation'. 'Product instantiation' evaluates the ability to create a definition of the derived product instance, the simplest form of which could be a list of selected features. An approach can explicitly include 'code generation' in its process and allow 'quality variability tracing across elements', meaning selections and bindings through the artifacts.

TABLE III. METHODOLOGY COMPARISON FOR QUALITY SUPPORT

Requirement	F-SIG	Ext-FM	ISPLM	Q-ImPrESS	QADA	IQSPLE
M1: qualitative values	Y	Y	N	Y	Y	Y
M2: quantitative values	N	Y	Y	Y	Y	Y
M3: algorithms for calculating the resulting attribute values	N	Y	N	Y	N	Y
M4: presentation as feature	N	N	N	N	N	Y
M5: artifact annotation	N	N	N	Y	Y	Y
S1: calculate the quality values of a given variant	N	Y	-	Y	Y	Y
S2: determine the set of possible variants	N	Y	-	N	N	Y
S3: constrain the selectable features	N	N	-	N	N	Y
S4: visualization of quality values	N	N	N	Y	N	Y
S5: quality modeling integration in solution space	N	N	N	Y	Y	Y
S6: reuse of artifacts	N	N	Y	Y	Y	Y
S7: traceability support	N	N	Y	Y	Y	Y

TABLE IV. METHODOLOGY COMPARISON FOR UML VARIABILITY SUPPORT

	SPLIT	PLUS	MDD-AO-PLE	UML ext. [22]	IQSPLE
quality annotation					✔
requirement analysis	+++	+++	++	+	++
feature model integration		✔	✔	✔	✔
negative variability				✔	✔
structural variability	✔	✔		✔	✔
UML compliant	✔	✔		✔	✔
modeling constraints	✔	✔	✔	✔	✔
configuration constraints			✔	✔	✔
checks during generation			✔	✔	✔
product instantiation	+++	+	+++	+	+
code generation			✔		✔
quality variability tracing across elements					✔

Approaches include the conceptual framework SPLIT [20], where additional UML stereotypes, e.g., <<variabilityMechanism>> and <<variationPoint>>, are used for specifying variable elements. However, SPLIT does not integrate an abstract feature view as does the IQSPLE, and the variation points and the corresponding variants require a separate class that may cause transparency issues in large SPLs. PLUS (Product Line UML-Based Software Engineering) [21] extends UML to model variability and commonality using stereotypes and primarily subclassing. [22] presents a generic modeling approach with additional variability stereotypes as extensions to UML. The variation points and variants can be assigned with tagged values to define certain properties, such as the binding time of variants, the multiplicity of associable variants, and the condition of binding. However, this approach does not address the derivation of product line instances. Crosscutting variability in SPLs is investigated in MDD-AO-PLE [23][24][25] and related aspect-oriented (AO) SPLE work. While this work has not specifically addressed the difficulties described in this paper for quality modeling integration, the combination of these AO techniques with IQSPLE could be synergistic and should be investigated in future work.

VII. CONCLUSION AND FUTURE WORK

By integrating quality modeling into SPLs with a holistic method and tool approach, the application of IQSPLE results in product as well as process benefits. At the product level, the resulting product instance has a higher potential value to the customer since it is more likely to conform to his or her expectations, satisfying not only requested features but also complying with quality expectations. At the process level, the use of an interactive, quality-driven selection process provides efficiencies by supporting on-the-fly evaluation and visualization for expeditious trade-off analysis while assisting with and automatically constraining valid variant selection against quality requirements. Efficiency is also furthered via reuse of quality modeling and annotations throughout the SPL lifecycle. The effectiveness of the process is enhanced by making qualities first-class requirement entities that are analyzed and formalized. The comprehensive approach promotes contemplation of (aggregated) quality impacts from the initial SPL concept since qualities can be (directly) annotated across the solution space artifacts (and are available directly to SPL engineers). SPL process effectiveness is also furthered by automatic evaluation and optimization as well as traceability support for the source of variability decisions. IQSPLE supports the flexible derivation of individual variants instead of a limited set of predefined variants via immediate feedback on the resulting quality attribute values of the current selection. Barriers to adoptability of quality modeling in industrial SPLs are lowered by IQSPLE due to its avoidance of unnecessary complexity (e.g., ontologies are avoided) and focus on integrating known and common tooling and standards (UML, OCL, stereotypes, tagged values, OMG's QoS profile, MDD (e.g., oAW), Eclipse, feature trees).

By fulfilling the M and S requirements, IQSPLE supports qualities in quantitative and qualitative ways. The application of constraints on features allows the explicit modeling of quality values inside a feature tree. It is not necessary to make any structural changes to feature trees or add any additional implementation details, so feature trees can still be used for customer communications. With quality functions, a mechanism is provided to transform different quality impacts into a single quality characteristic and thus make it possible to compute qualities of a variant. To make it easier to recognize how changes in feature selections affect qualities, quality values can be displayed in spider charts.

The eHealth scenario shows how IQSPLE supports the detection of optimization opportunities and trade-offs during product instantiation, making use of the traceability of the modeled dependencies.

Models are necessarily limited in their portrayal of reality, and holistic quality modeling of a SPL faces significant challenges due to the large set of variations. While IQSPLE may contribute towards improved quality modeling in SPLs, much work remains. Future work should address visualization of influences on qualities via quality interaction dependency graphs; model checking integration (e.g., UML dependency changes may affect OCL constraints); OCL validity and syntax-checking of, e.g., component names and quality attributes, perhaps eventually code-completion support; the support for and integration of developmental qualities (e.g., architectural metrics affected by a specific configuration); replacing the current brute-force variant evaluation algorithm by alternatives described in "Evaluation" Section V; decision support via analysis of potential variants; enabling tolerances and trade-offs in formulas for automatic optimization; and the application of the IQSPLE in other industrial domains beyond the medical domain.

REFERENCES

[1] J. Bartholdt, M. Medak, and R. Oberhauser, “Integrating Quality Modeling with Feature Modeling in Software Product Lines”, Proc. of the Fourth International Conference on Software Engineering Advances (ICSEA 2009), IEEE Computer Society, 2009.

[2] J. Bartholdt, R. Oberhauser, A. Rytina, „Addressing Data Model Variability and Data Integration within Software Product Lines”, International Journal On Advances in Software, ISSN 1942-2628, vol. 2, no. 1, 2009, pp. 86-102

[3] F.J. v.d. Linden, K. Schmid, and E. Rommes, “Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.” Springer, 2007, ISBN 3540714367.

[4] K. Pohl, G. Böckle, and F.J. v.d. Linden, “Software Product Line Engineering: Foundations, Principles and Techniques”. Springer, 2005, ISBN 3540243720.

[5] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute, Carnegie Mellon University, 1990.

[6] M. L. Griss, J. Favaro, and M. d. Alessandro, “Integrating feature modeling with the RSEB,” Proc. of the 5th International Conference on Software Reuse (1998), ICSR, IEEE Computer Society.

[7] J. Bayer, et al “PuLSE: a methodology to develop software product lines”, In Proceedings of the 1999 Symposium on Software Reusability (SSR '99), ACM, pp. 122-131, 1999.

[8] I. Ozkaya, L. Bass, R. S. Sangwan, and R. L. Nord, “Making Practical Use of Quality Attribute Information,” IEEE Softw. 25, 2 (Mar. 2008), pp. 25-33. DOI= http://dx.doi.org/10.1109/MS.2008.39.

[9] OMG, UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics & Mechanisms, v1.1, formal/08-04-05

[10] S. Jarzabek, B. Yang, and S. Yoeun, “Addressing quality attributes in domain analysis for product lines,” IEE Proceedings Software, vol. 153, no. 2, pp. 61–73, 2006.

[11] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, “Automated reasoning on feature models”, in LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, pp. 491–503, 2005.

[12] N. Siegmund, M. Kuhlemann, M. Rosenmüller, C. Kästner, and G. Saake, “Integrated product line model for semi-automated product derivation. Using non-functional properties,” Proc. of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS), pp. 25–31, 2008.

[13] S. Becker, M. Hauck, M. Trifu, K. Krogmann, J. Kofron, “Reverse Engineering Component Models for Quality Predictions”, CSMR 2010, IEEE, Mar 2010.

[14] M. Matinlassi, E. Niemelä, and L. Dobrica, "Quality-driven architecture design and quality analysis method, A revolutionary initiation approach to a product line architecture," VTT Technical Research Centre of Finland, Espoo, 2002.

[15] A. Purhonen, E. Niemelä, and M. Matinlassi, "Viewpoints of DSP Software and Service Architectures," Journal of Systems and Software, vol. 69, 2004, pp. 57 - 73.

[16] IEEE, "IEEE Recommended Practice for Architectural Descriptions of Software-Intensive Systems," Std-1471-2000. New York: Institute of Electrical and Electronics Engineers Inc., 2000.

[17] A. Evesti, E. Niemel, K. Henttonen, M. Palviainen, "A Tool Chain for Quality-Driven Software Architecting," splc, pp.360, 2008 12th International Software Product Line Conference, 2008.

[18] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, “Modeling dependencies in product families with COVAMOF”, Proc. of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006), March 2006.

[19] F. Bachmann and L. Bass, "Introduction to the Attribute Driven Design Method," icse, pp.0745, 23rd International Conference on Software Engineering (ICSE'01), 2001.

[20] M. Coriat, J. Jourdan, and F. Boisbourdin, “The SPLIT method: building product lines for software-intensive systems,” In Proceedings of the First Conference on Software Product Lines: Experience and Research Directions (Denver, Colorado, United States). P. Donohoe, Ed. Kluwer Academic Publishers, Norwell, MA, 2000, pp. 147-166.

[21] H. Gomaa, “Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures”, Addison-Wesley, 2005, ISBN 0201775956.

[22] M. Clauss, “Generic modeling using UML extensions for variability”, In Proceedings of the Workshop on Domain Specific Visual Languages, OOPSLA 2001, Jyväskylä University Printing House, Jyväskylä, Finland, 2001, ISBN 951-39-1056-3, pp. 11-18.

[23] M. Voelter and I. Groher, “Handling Variability in Model Transformations and Generators”, in Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM’07), Sprinkle, J., Gray, J., Rossi, M., Tolvanen, J.-P., (eds.), Computer Science and Information System Reports, Technical Reports, TR-38, University of Jyväskylä, Finland 2007, ISBN 978-951-39-2915-2.

[24] M. Voelter and I. Groher, “Product Line Implementation using Aspect-Oriented and Model-Driven Software Development,” In Proceedings of the 11th international Software Product Line Conference (September 10 - 14, 2007). International Conference on Software Product Line. IEEE Computer Society, Washington, DC, 2007, pp. 233-242.

[25] I. Groher, "Aspect-Oriented Feature Definitions in Model-Driven Product Line Engineering", Dissertation, Johannes Kepler Universität, Linz, April 2008.

Enabling Innovations in Mobile-Learning: A Context-aware and Service-based Middleware

Sergio Martin, Elio Sancristobal, Rosario Gil,
Gabriel Díaz, Manuel Castro and Juan Peire
Electrical and Computer Engineering Department
UNED (Spanish University for Distance Education)
Madrid, Spain
{smartin, rgil, elio, gdiaz, mcastro,
jpeire}@ieec.uned.es

Mihail Milev, Nevena Mileva
Electronics Department
University of Plovdiv
Plovdiv, Bulgaria
mmilev@dipseil.net, nmileva@uni-plovdiv.bg

Abstract— **Development of mobile learning projects involves addressing many challenges, from pedagogies (e.g., what must students learn? and how?) to technical issues (e.g., use of context-awareness, use of communication or collaboration methods, mash-up of information). This paper introduces a framework intended to provide different educational tools and information to mobile learning programmers and designers in order to simplify and reduce the project development. The paper offers an overview of the state of the art of mobile learning applications, focusing on some of its problems: lack of interoperability and difficulty of using advanced technologies, such as location-based systems. It also addresses why most of the m-Learning applications do not make use of the existing services and knowledge of the Learning Management Systems, which are in fact the real pillars of the e-learning methodology. The framework design is guided by a new paradigm for development of e-learning tools and platforms: Learning as a Service (LaaS). The main contribution of this framework is to provide contextual information from different sources and sensors (e.g., geographical, motion); and integration of many existing services from e-learning platforms.**

Keywords- context-awareness, e-learning, framework, location, LMS, mobile learning, ubiquity

I. INTRODUCTION

Nowadays mobile technologies are being applied on educational environments quite successfully. One of the main reasons of this success is the improvement of the technical features of the devices. New generations of mobile devices have wider and touchable screens; built-in digital cameras; and connectivity not only with GPRS but also with Wi-Fi or 3G. In some of them it is even possible to find GPS receivers, RFID, NFC readers or smartcards integrated.

All these new technologies inside a small and portable device are giving rise to a new generation of applications in all kind of environments. These kinds of applications are called mobile learning (M-Learning) inside the educational environment. Here, mobile devices are supporting collaborative and mobile work and enabling students to learn anytime anywhere, especially through games, applications or courses designed for these small devices.

Actually location-based and context-aware systems can give a very interesting added-value to M-Learning, because they allow the creation of mobile context-aware application that lets students interact with their environment in a totally new way. For instance, a student in a canteen will have different needs than in a laboratory or in a secretary; or a teacher in a classroom will need different information than in an office. It is possible to offer personalized learning through the mobile device depending not only on his/her profile but also on the moment and his/her location.

The other focus of the paper is over the integration and interoperability of the existing e-learning tools and applications with the mobile learning environment, giving rise to a complex mobile digital ecosystem of educational applications.

The paper might be of interest for those involved in the development of mobile learning applications, since the introduced framework could become a powerful and useful tool for them; for those also designing supportive tools to simplify the development of mobile learning applications; and in general for those involved in the development of any kind of e-learning or m-learning application, since the paper introduces a service-oriented methodology for the development of applications.

The paper is structured as follows: Chapter II offers an overview of the main goals of the system. Chapter III describes what the authors consider m-learning, and provides a mobile learning classification. This chapter is complemented with the learning theories and methodologies of Chapter IV. In chapter V, the necessity of the integration of the new technologies related to location systems into the m-learning environment is addressed. Chapter VI offers an overview of the Learning as a Service (LaaS) methodology used in the project design. Chapter VII shows the importance of the interconnection between the e-learning platforms and the m-learning applications, according with the LaaS paradigm. Finally, chapter VIII describes the design of the proposed middleware. Chapter IX offers an

overview of related works; and Chapter X some conclusions.

II. GOALS

The aim of the paper is the description of a framework developed recently by the authors that supports and simplify the creation of M-Learning applications: M2Learn [1].

The main contribution of this architecture is the combination of ubiquitous and context-aware technologies with the Learning as a Service (LaaS) paradigm, which will be introduced in the following chapters. The result is an easily-extended framework that offers useful services and information to high-level applications. Thus, developers using this framework will see their work considerably reduced.

Although there are many other frameworks to help in the creation of context-aware systems and mobile applications, this development supposes an improvement of the state of the art. The reason is because M2Learn is particularly focused on the learning environment, as its design is closely related with the existing e-learning platforms and tools. As a summary, the main features that the framework includes are:

- Context-awareness. Developed mobile applications will easily incorporate information about user's geo-location, which is useful for personalizing the services and activities offered to students; or for having a log of their movements in certain activities. Currently the location methods supported by the M2Learn framework are Global Positioning System (GPS), triangulation of Wi-Fi access points, and identification through Radio Frequency Identification (RFID). The framework also monitors student's hand movements through motion sensors. It offers a more natural way of interacting with technology, giving rise to a new kind of engaging applications. Other relevant piece of information to compile the student context is the academic profile (e.g., degree, subject, or preferences), since it can be used to personalize services and information according with their interests. Finally, the user's context contains the user's activities history (log), which can be useful to study user preferences (e.g., applications used, information accessed, etc), and movements in an environment (Figure 1).

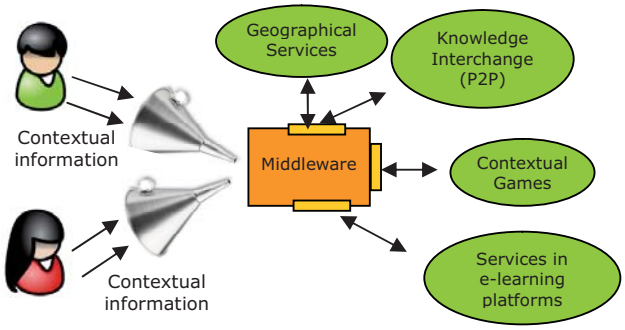


Figure 1. Offering personalized services and information depending on user's contextual information..

- Communication tools (e.g., forum, chats) from e-learning platforms (e.g., dotLRN or Moodle). Lately much effort has been undertaken to develop useful e-learning tools but most of the new mobile applications leave them aside, with almost no interoperability with other systems. For that reason, authors have developed interfaces between many e-learning services and the M2Learn framework. These interfaces let students access the e-learning contents and services not only anytime, but also anywhere, and with any kind of device. These interfaces also give the opportunity of integrating information from the mobile applications to the e-learning platforms.
- Collaboration. The framework supports the incorporation of Web-based tools for collaborative work such as Google Docs. This feature let the developer create mobile learning applications that include for example a text document that will be created collaboratively among the students.
- Knowledge. M2Learn framework manages different formats of knowledge representation, e.g., PDF, and DOC. But the framework also gives support to the integration of knowledge from search engines, web sites (e.g., Wikipedia) or Learning Management System's (LMS) FAQ.

In the development of this middleware, some issues have been considered, such as which is the best model of m-learning application: learning on the move or learning with mobile devices?; which is the kind of mobile applications that provides an added-value in the educational experience?; how the new features can be implemented inside these new kind of applications?; and why is better to include the existing e-learning resources into the mobile-learning paradigm instead of doing everything again from scratch?.

III. IS M-LEARNING LEARNING WITH MOBILE DEVICES OR LEARNING ON THE MOVE?

When we talk about mobile-learning, it is possible to assume that m-Learning is all kind of activity carried out in an educational environment with a mobile device. This trend basically moves the traditional learning, with a teacher giving a master class in a classroom, towards the mobile world. Many examples can be found, for example in [2] where an application on a mobile device reads a chapter of a book in a karaoke-style; or the MPSS project [3], where students must follow some courses and are evaluated through some test exercises using the mobile device.

On the other hand there are other m-learning projects oriented towards outdoor learning. This is a totally different point of view, due to the fact that in these projects the idea is not to try to apply the old methodology into the new technologies, but they try to develop new environments where mobile devices offer an added value to students' education. These environments do not try to supplant the master class. They just complement the traditional and more formal education, opening a new range of possibilities to

students. For example, Bouvin [4] worked in a prototype where learners should explore and document parts of a city for later presentation in plenum. Other project more oriented to the context-awareness is a prototype developed by Bomsdorf [5] allowing learning materials to be selected depending on a given situation – this takes into account learner profiles such as their location, time available for learning, concentration level and frequency of disruptions. Similarly, a context-aware mechanism has been developed by Bouzeghoub et al. [6], which takes into account time, place, user' knowledge, activity, and environment together with the device capacity for adaptation to the user.

After many years of researching in this field our point of view is that m-Learning must complement both e-learning and traditional learning, because in our opinion each methodology has its own place.

Thus, from our study [7], the most spread m-Learning applications can be classified into several kinds of applications depending on its functionality:

- Learning on the move: Intended to education out of class, not replacing the face-to-face class, but complementing it, and providing different ways of learning on the field. There, m-learning seems to cater for certain specialties more than others such as agronomy, biology, geology, archaeology, tourism, etc.
- Location-sensitive. Location-based systems provide a very interesting added-value to mobile learning applications. They allow the creation of context-aware applications that lets students interact with the environment in a totally new way. This kind of applications react specifically to their current location, time and other environment attributes and adapt their behavior according to the changing circumstances as context data may change rapidly.
- Mobile review: Mobile devices offer very good results for quick reviews of aspects that the students have previously learnt in class. This is typically useful for example in the train. This model is often presented for both review of contents with documents or notes; and auto-evaluation with quiz tests.
- Collaboration: Development of collaborative exercises together with other students, giving rise to applications such as mash-up systems (based on geographical information services), Wikis, or games.
- Services from e-learning tools. For instance, quick accesses to services of a LMS, for example, check the messages in the LMS forum of a course; or access a Virtual Lab.
- Podcast allows ubiquitous learning whereby students can access a variety of educational material anywhere, anytime on iPods, MP3 and MP4 players or mobile phones. Podcasts allow anywhere, anytime learning. They permit students to access educational materials at home, while travelling to university or work, or doing any activity they choose. They can play the recordings at any time, which is convenient to them rather than be confined to set class times. Podcasts in the educational setting allow students on-demand access to audio or video-recordings of lectures or other learning materials at their convenience. But although Podcast is a very in-fashion methodology, does not really suppose a revolution in education. It only applies the same methodology (a master class given by a lecturer) to a new format. It presents an important disadvantage that avoids its generalized use in education: it is one-way.
- Augmented Reality is a technology that basically merges information or images with video-streaming from a Web-cam. The result is similar to virtual reality but using real-world images in real-time. Some of the many potential revolutionary applications in education are related for example with the study of architecture, art, anatomy, decoration, or in general anything that a graphic, a simulation or a 3D model could improve the comprehension of the concepts.
- Games can be one of the most powerful tools for learning. Students are easily engaged with them, especially with those that use natural interaction, such as motion recognition (e.g., puzzles; or question-based contests using the mobile device as a pointer).
- Other useful m-learning applications are for example reminders or schedulers of the learning process. This applications helps students to remind some events at certain moments and places, e.g., "Remember take the Math's book from the Library" when the student is walking nearby.

The proposed framework has had in consideration the different possible applications that a mobile learning project includes, so it can offer useful information and services to them, simplifying considerably its development.

IV. LEARNING METHODOLOGIES AND THEORIES APPLIED TO MOBILE LEARNING

From the educational point of view, the different learning methodologies and theories have also been taken into account in the design of the M2Learn framework. The objective was to design a framework to give support to different kinds of applications, not only from the functionality point of view, but also from the educational.

According with the literacy, there is no academic theory or methodology specifically developed for mobile learning, addressing assessment, pedagogy and instructional design issues. In its place, the traditional learning theories are being applied with more or less success depending on the kind of application and environment. The following points describe the main theories, methodologies and paradigms.

A. Behaviorism

Learning theory mainly developed by Skinner. It is based upon the idea that all behaviors are acquired through conditioning [8]. The other important exponent of this theory

is Pavlov. According with Pavlov, behavior becomes a reflex of a given stimulus [9], as in the case of Pavlov's dogs. In this theory, learning is based on a stimulus (problem), its response (solution) and the reinforcement given (feedback). This reinforcement can be presented as a comment, an award or a punishment.

Mobile learning examples can be found in applications devoted to deliver content, get the learners' response and provide constant feedback:

- Classroom response systems [10].
- Mobile Performance Support System for Vocational Education and Training (MPSS Project) [11].
- Content delivery by SMS.

Roschelle describes as a benefit of the implantation of this theory in mobile learning the anonymity in the students' responses [12]. Thanks to technology students do not feel worried about consequences of bad answers. It helps the teacher in understanding the classroom's knowledge level [13].

Although this approach is widely used in e-learning and has some advantage in m-learning, it finds several disadvantages such as the use of limited displays and restricted input methods when it is used as a way to deliver content.

B. Constructivism

Constructivism views learning as a process in which the learner actively constructs or builds new ideas or concepts. It is often associated with pedagogic approaches that promote active learning, or learning by doing. In this theory, learning is viewed as a process where learners actively construct new concepts from their experiences (previous and current knowledge) [14] through two processes:

- **Accommodation**. The process of reframing the mental representation of the external world to fit new experiences. It also promotes that failure leads to learning. Along these lines, when we act on the expectation that the world operates in one way but it does not happen, we accommodate this new experience and reframe our model of the way the world works. So learning comes from the experience of failure, or others' failure.
- **Assimilation**. It is the incorporation of new experience into an already existing framework without changing that framework. This may occur when individuals' experiences are aligned with their internal representations of the world, but may also occur as a failure to change a faulty understanding. For example, they may not notice events, may misunderstand input from others, or may decide that an event is not important as information about the world. In contrast, when individuals' experiences contradict their internal representations, they may change their perceptions of the experiences to fit their internal representations.

In this theory, such as Bruner points out, instructors encourage students to discover principles by themselves, helping them to move from passive listeners to active constructors of knowledge by working to solve realistic problems [15].

Constructivism itself has many variations, such as Active learning, discovery learning, and knowledge building. Other branch is social constructivism that defends that knowledge is built when learners interact with others through talks and activities about shared problems or tasks. Here, learning is supported by more skilled members.

Regarding mobile learning, one of the most relevant applications within this learning theory is any kind of application based on role-games. To that effect, some educational games fit perfectly with this theory. Educational games provide an immersive experience that helps learners to act in the educational environment (virtual world) as an active. It provides a better knowledge acquisition, because they can manipulate the virtual world without the fear of a failure. Games also allow interacting with other learners collaborating to achieve a particular task.

Other kind of application that matches with this theory is the location-based systems aimed to facilitate learning out of the classroom, for example in an art or archaeological museum.

Finally, participatory simulations are also related to this theory [16]. This kind of applications proposes immersive recreation of real situations to become part of the process. Here learners do not watch a simulation, but they are part of it [13].

C. Situated learning

This paradigm promotes learning within an authentic context with social interaction [17] [18]. In this case, teachers propose problems to be solved (Problem and case-based learning) to students in real environments, where they feel immersed in an authentic environment that helps a better acquisition of knowledge. It fixes perfectly with mobile learning through the context aware and location-based systems [19] [20].

D. Observational Learning or Social learning theory

This kind of learning occurs when an observer's behavior changes after viewing the behavior of a model [21]. Thus, this paradigm is based on imitation of a model that the observer finds interesting, attractive or desirable in some way.

Although this approach does not necessarily imply interactivity, it is widely used in mobile learning through the use of Video-based systems (e.g., web-based video systems, mobile TV or Podcast), that transmit knowledge to learners through the use of videos. Other mobile applications that make use of this paradigm are those where authentic situations are described to show how to react in certain situations.

E. Collaborative learning

This paradigm promotes learning based on social interactions [22]. In this environment, there must be a

dynamic interaction among instructor, learners and the proposed activities to help them creating their own truth thanks to the interaction with the others.

This approach also fits perfectly with mobile learning, because handheld devices can offer synchronous and asynchronous communication methods. In addition, the Web 2.0 concept is arriving also to the mobile environment, supporting collaborative work such as edition of on-line documents, creation of wikis, blogs, virtual boards, etc.

V. NEW LEARNING NEEDS NEW TECHNOLOGIES

Once we determine the way to go inside m-Learning, other aspects must also be considered. Mobile devices are getting better everyday, incorporating new functionalities, such as GPS receivers, motion sensors, Wi-Fi connectivity, etc. These new features must be included inside the educational environment; especially those related with geo-location, due to the fact, knowing where students are in each moment can be used for offering them personalized services [23].

It is a fact that the main and most widely used location method in mobile devices is the Global Positioning System (GPS). This technology offers a quite reasonable accuracy at outdoor environments. Each GPS satellite transmits data that indicates its location and the current time. Signals, moving at the speed of light, arrive at a GPS receiver at slightly different times because some satellites are farther away than others. The distance to the GPS satellites can be determined by estimating the amount of time it takes for their signals to reach the receiver. When the receiver estimates the distance to at least four GPS satellites, it can calculate its position in three dimensions.

However, the use of this technology inside buildings is not possible (without the use of GPS repeaters, which are not very spread) because the receiver needs to have direct contact with the satellites. For that reason, technologies such as Wi-Fi, Cell towers or RFID are appearing in location systems for indoor environments.

In the Wi-Fi based location the mobile device recollects different levels of noise and power that the access points spread in an environment (at least 3 access points are required) every few milliseconds. This information is processed to obtain the user's coordinates [24].

This same philosophy is being applied with Cell Towers instead of using Wi-Fi. This is the case of the new iPod that is revolutionizing the mobile market thanks to its great interface features and location capabilities [25] [26].

On the other hand, Radio-frequency identification (RFID) is a technology that is being used in more environments, and is one of the pillars of the called "Internet of Things". It is an automatic identification method, consisting in storing and remotely retrieving data using devices called RFID tags or transponders [27]. This identification is usually used for identification [28], as the substitute of bar codes, tracking any kind of products or even animals or people. Some tags can be read from several meters away and beyond the line of sight of the reader. In addition this identification method can also be used to locate people in an area, but it only locates people when the RFID reader identifies the tag that the user carries.

VI. LEARNING AS A SERVICE (LAAS)

But innovation in mobile learning, besides hardware, also requires new methodologies and paradigms that guide the development of new projects.

In our case, during the last years we have been developing the Learning as a Service (LaaS) paradigm, which is an extension of the widely extended Software as a Service (SaaS). SaaS is based on the idea of having different services available on the Internet (on the Cloud) that can be used and integrated regardless of physical location. In the case of LaaS, these services will come mainly from e-learning applications, as we work in the education environment. As examples of services provided by e-learning applications and used in our middleware we can find: forum rooms and chat from LMS, Wikis, Blogs, collaboration tools, etc.

LaaS is based on the concept of modelling the educative services with the objective of providing better interoperability capabilities in different levels: service-service, platform-platform, and service-platform. Equally it is based on the encapsulation of learning objects services. The aim is to develop autonomous and self-contained services that can be easily integrated in different environments such as Learning Management Systems and mobile learning environments.

The main objective of this paradigm is the reuse of existing services, providing interoperability of services among platforms and applications. This methodology has been coined as a consequence of the important efforts already done in the e-learning field, and the fact that new mobile learning applications usually leave them aside.

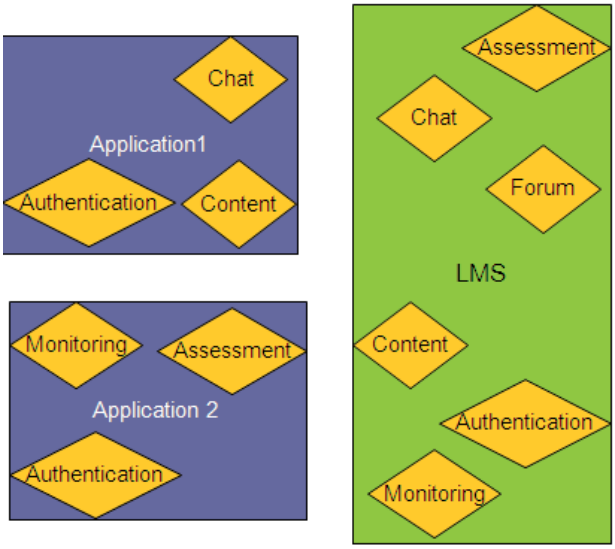


Figure 2. Different applications and LMS in an educational environment with replicated services.

New mobile developments tend to create everything from scratch, giving rise in one hand to a very spread student's e-portfolio. On the other hand, both economic and time project efforts are considerably higher than if services were designed for service reutilization and interoperability among platforms and applications (Figures 2 and 3).

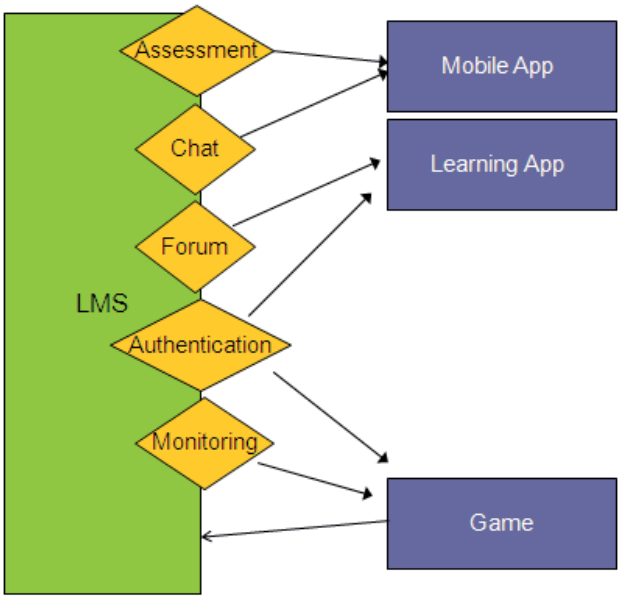


Figure 3. Reuse of service. Utilization of LMS's services in external applications, including games and mobile learning applications. Other applications such as games could include also services in the LMS.

As a consequence, LaaS methodology promulgates the creation of learning services that could be integrated in other applications and services. Thus, M2Learn framework includes interfaces to externalize many LMS's services, such as forum rooms, and FAQs to be easily integrated in mobile learning applications; games; or any other kind of applications. LaaS also remarks the importance of integration of external services into the LMSs, extending its functionality.

As a consequence, other result of the application of LaaS in our project is that the monitoring activities of mobile learning applications can be integrated into the LMS. Thus, teachers will have the opportunity to check the whole e-portfolio of the student, no matter if the information comes from the LMS, from a mobile application, or other service.

In addition, there is another surrounding concept to this methodology within educative services: Digital Educative Ecosystems [29]. This concept is based on the idea of creating environments made of different integrated systems (mobile clients, applications, services and other tools) devoted to improve the learning experience by supporting communication and collaboration. One of the bases of this concept is the use of SOA (Service-oriented architecture) technologies such as SOAP [30], Web Services [31], ESB (Enterprise Service Bus) [32] or REST [33]. Actually, most of the developed services in the project follow the REST methodology.

VII. CONNECTING WITH THE LEARNING REPOSITORIES: SERVICES IN LMS

Reading the m-Learning bibliography, most of applications do not have any relation with the already existing LMS. Some examples are in [34], where a prototype was developed in order to use Moodle through Smartphones. Basically they changed the three-frame template of Moodle for a one-frame template to allow users to visualize better the content. Finally it did not offer the whole functionality of Moodle because many features did not work properly in a smartphone.

Other example can be found in [35] where a web service-based architecture is proposed to move some of the Moodle functionalities into a mobile device. This also allow to re-use some of the existing services in LMS, such as authentication, monitoring, etc., so it is not necessary to create them again.

Thus, excepting some prototypes, most of m-Learning systems are isolated and autonomous applications that do not make any use of the knowledge stored in them. These platforms are in fact the real support of the e-learning methodology, but it seems like we wanted to let them out of the m-Learning paradigm.

That is why we also try to interconnect these e-learning platforms with the mobile devices, in order to take advantage of all the services and knowledge already existing in them.

Our middleware interacts with the university LMS also through a set of RESTful Web Services, providing an interface to implement some functionalities in a mobile device, such as, access to forum rooms, content or FAQs. In addition, as we have cited before, there are other advantages related to the re-use of services instead of creating them again: authentication, tracking activities, evaluation, content, etc.

In addition, the LMSs can provide very useful contextual information. For example, for many context-aware applications knowing the courses where a student is signed in can be very useful in order to personalize the offered services. For example, by using the credentials (login and password) it is possible to retrieve information about his/her educational environment, e.g., accessing to the university's e-learning platform. In this case the system can know the student's degree or subjects, and provide this information to higher level applications that will use it to personalize the services provided to the student.

VIII. MIDDLEWARE DESIGN

Once we have achieved all these conclusions we are prepared to design the middleware that will make easier the creation of context-aware m-learning applications, interconnecting them with the already existing LMS services.

Firstly, the architecture of the system must ensure it is possible to use several sensor networks to provide contextual information about the user [36] [37]. For that reason several sensors controllers have been developed in order to understand the information provided by physical sensors (level 1 at figure 4). At this stage we are working

with GPS, RFID and Wi-Fi, but the system is designed using open interfaces so that it is easy to add new location methods, such as cell towers, Bluetooth, infrared, etc.

The proposed architecture can be studied as a stack of services, protocols and applications. At the top level is the context-aware application (level 6 at figure 4) that uses the contextual information provided by sensors to access to particular services such as information in a LMS, e-mail, etc (level 5 at figure 4).

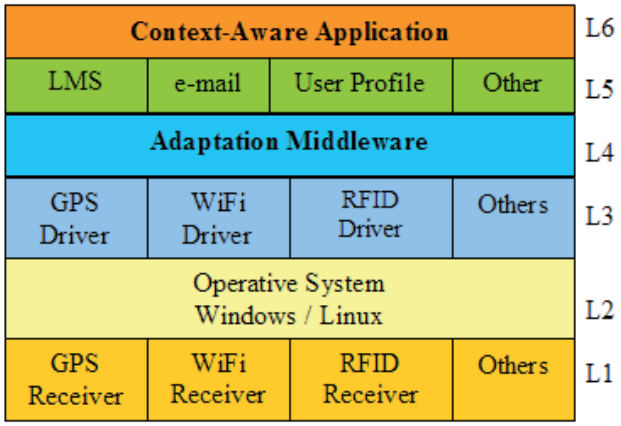


Figure 4. Architecture scheme

The most important level in the stack is the adaptation middleware (level 4 at figure 4) that is a homogenizer layer that offers an interface to the applications. The context-aware application will not have to interact with the sensors, and even it will not know from which sensor the information comes from at this time. This interface provides all the contextual information in a transparent way, ensuring the applications will not have to worry about the implementation of the lower layers. This is the same philosophy than the TCP/IP levels.

It is based on a set of wrappers (or controllers) that interact with the physical devices and offer homogeneous information about not only geo-location, but also about the user's context (Figure 5).

The main goal of this interface is to use the information retrieved from the location technologies to obtain more information about the user. For example, from a particular user is possible to know basic information like the geo-coordinates and the name. It is also possible to know information about the device and the operating system.

On the other hand, the framework uses a communication manager that is in charge of the management of the external communication services and tools. For example, one of the developed wrappers is an interface to a forum room in dotLRN, so the developer can easily create applications that read or write messages in a forum, so the teacher will be able to see the activity in the mobile application through the LMS. This particular interface has been developed creating a REST-like Web Service that wrap the complexity of the dotLRN forum management. The M2Learn framework connects with this service to offer this functionality to the mobile application developer.

Other example of service integrated in the framework is an intelligent question-answer system that retrieves information from different sources to answer students' questions. This service can be configured to access to Web Sites such as Wikipedia or the university site; institutional databases; or search engines to look up for answers.

Finally, the top level applications use all this information provided by the interface, but without knowing from which source it comes. The fact is that there is a lot of contextual information about the user and services that can be used to improve the learning experience.

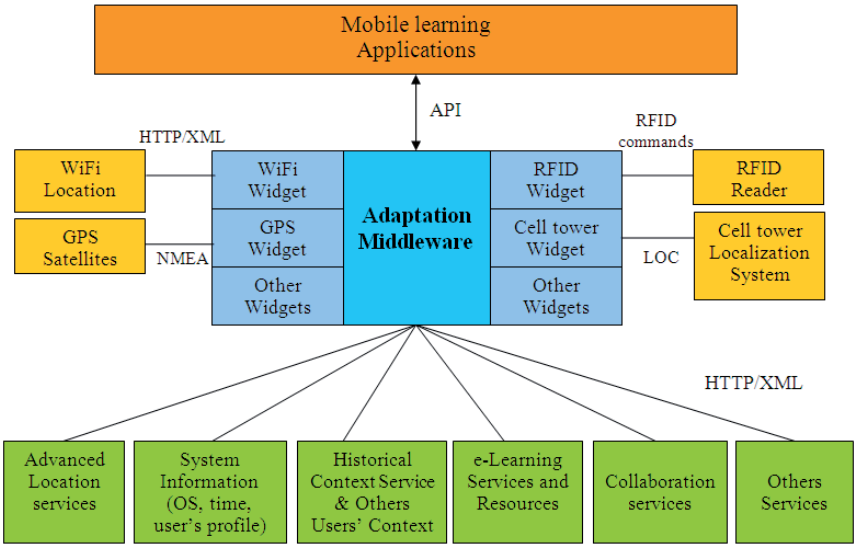


Figure 5. Architecture to provide interoperability between different location technologies and retrieve contextual information for M-Learning applications.

All these modules make up the M2Learn Unit (Figure 5). It is the API that higher level applications use to access the contextual information and personalized services.

This interaction is achieved through a set of Web Services. They provide an interface to implement mobile device functionality, such as for example, access to forum rooms, content, or FAQs. In addition, there are other advantages related to the re-use of services instead of creating them again: authentication, tracking activities, evaluation, content, etc. The LMS can also provide very useful contextual information.

However, a mobile learning environment is not made up of a single user, but is complex digital ecosystem with different kind of users (students, teachers, and staff), services, applications and platforms. That is why our scenario envisages the coexistence of several M2Learn Units communicating and collaborating simultaneously.

From a general overview, the mobile digital ecosystem built around the M2Learn framework contemplates the co-existence of (Figure 6): M2Learn units, e-learning applications, and LMS.

From a development point of view, there are two possible ways of building applications using the M2Learn framework:

- A mobile client built over the M2Learn framework, using its API to access the information and services available in the mobile digital ecosystem.
- A Mash-up system that retrieves the information of the mobile clients in the environment through the Context-Hub.

The interaction in this mobile digital ecosystem is possible thanks to two important modules:

- Context Hub. This module receives the user's contextual information and allows its distribution (Figure 7). Using the Context-Hub module users can discover information about other surrounding users. In addition, this module is also fundamental to support the development of mash-up systems. It provides contextual information of all the mash-up system users, which can merge location, preferences, history and other services.

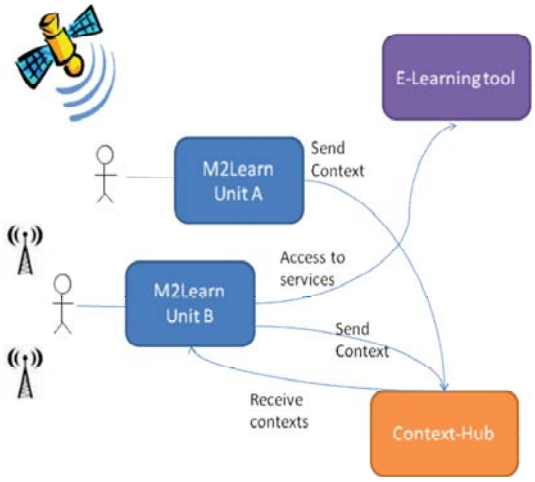


Figure 7. Several M2Learn Units sending their context to the Context-Hub. The Unit B retrieves all the context to allow high-level applications its use.

- Context Directory Service (CDS). This service is a central directory where all the external services must be registered in order to be available for the M2Learn Units. Later, applications built over the M2Learn framework access the CDS requesting the available services in this environment (Figure 8).

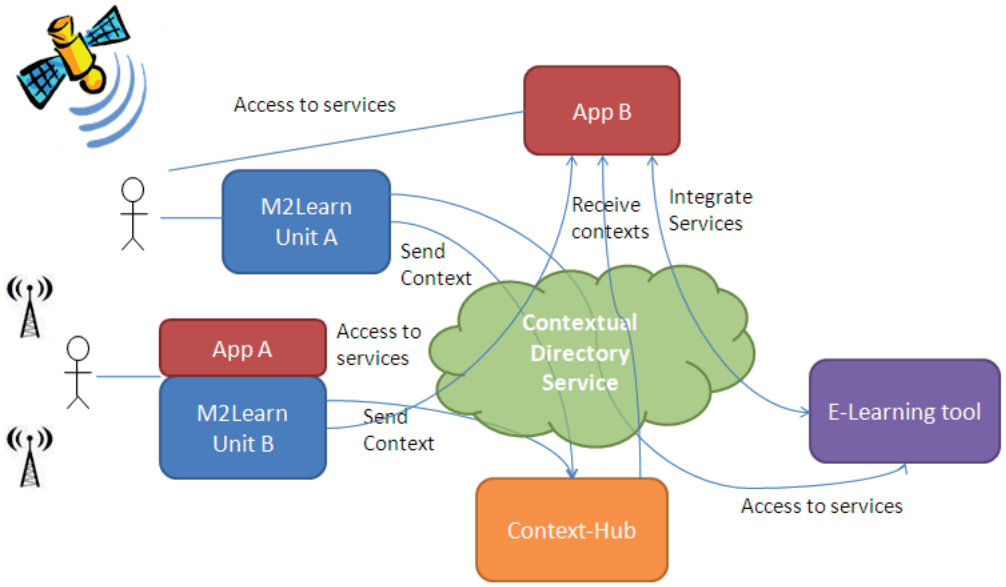


Figure 6. Interconnection of modules through the Contextual Directory Service. Some applications make use of the Context-Hub to get other users' contextual information. Application A is built in a mobile device over a M2Learn Unit, while the applications B in a Web-based Mash-up system that retrieves information of the mobile digital ecosystem and matches with a service from an e-learning tool.

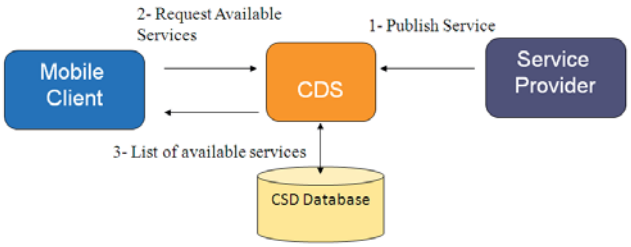


Figure 8. A service provider register a service in the CDS. Later a mobile application request the CDS the list of available services.

IX. EVALUATION

Evaluation has been conducted using students of a "Professional Expert Course on Mobile Programming". They have developed several applications using the framework. Later, students completed a questionnaire on user satisfaction and the simplification degree obtained through the framework's use.

Within this experience, students easily developed a mobile context-aware application that loaded different LOM-based resources depending on their location and profile. One of these resources was a simulation of a virtual laboratory of thermodynamics. The development of this application was effortless, and required very few lines of code thanks to the use of M2Learn.

X. RELATED WORK

In the following section other context-aware frameworks are described intended to create not only m-learning applications, but any kind of context-aware system. But any of them offer the same set of features than our framework, due to the fact that it offers information and services not only from the mobile device (contextual information) but also from the university LMS and from other external services.

A. *HyCon Framework*

The HyCon framework [38] was developed to provide a general platform suitable for experiments with hypermedia mechanisms in a context-aware and mobile environment. HyCon encompasses an infrastructure for implementing context-aware services and applications and a framework, which can be used by applications programmers to build such services and applications.

B. *Stick-e Note Architecture*

The Stick-e note software [39] infrastructure developed at the University of Kent and provides one of the first approaches to support the development of context-aware applications.

The aim of the infrastructure is to simplify the creation of context-aware applications using the electronic equivalent of a Post-it note and, as such, it focuses on information presentation and in particular discrete context-aware applications, i.e., those in which the information presented to the user does not change continuously.

In such applications, separate pieces of information are attached to specific contexts (location, states, temporal ranges, and adjacency) and are presented to the user when the appropriate context is entered.

The infrastructure is aimed at mobile users carrying small computing devices, such as PDAs, enhanced with environmental sensors, but is essentially an on-line system and does not explicitly address mobility issues.

C. The Context Toolkit

The Context Toolkit [40] is an architecture developed at the Georgia Institute of Technology that aims to provide reusable solutions to the problems of developing context-aware applications. The main aim of the toolkit is to free developers from having to deal with the low-level details of context acquisition and allow them to concentrate on the specification of higher-level application behaviours.

The toolkit is inspired by the success of toolkits for Graphical User Interface (GUI) development and is based on the GUI concept of a widget as a reusable component for abstracting away from and hiding the specifics of a physical device. Through the widget abstraction, the Context Toolkit aims to enable context data to be handled in the same way user input is currently handled.

The Context Toolkit provides useful domain-specific abstractions for the incorporation of context data garnered from sensors into applications, through the use of the widget abstraction, and this is its major strength. In addition, the interpreter abstraction of the toolkit provides a means to convert sensor data to higher-level context.

D. *MiLK: The Mobile Informal Learning Kit*

MiLK [41] is a support tool that allows teachers and students to develop event paths that consist of a series SMS question and answer messages that lead players through a series of checkpoints between point A and point B. These event paths can be designed to suit desired learning scenarios and can be used to explore a particular place or subject. They can also be designed to facilitate formal or informal learning experiences.

E. *The Context Fabric*

The Context Fabric project being carried out by the Group for User Interface Research at the University of California at Berkeley [42] proposes a novel approach to providing support for context-awareness in the form of a service infrastructure model. This model attempts to shift as much of the task of context-aware computing as possible to a network-accessible middleware.

This approach aims to aid the development of applications based on a diverse and constantly changing set of sensors and devices by providing uniform abstractions and reliable services for common operations. The Context

Fabric provides sensor abstraction and is one of the few projects providing an explicit treatment of proximity as a useful concept in context-aware computing. Sensor fusion as an approach to managing the uncertainty of sensor data is not dealt with in the project.

XI. CONCLUSSION

M2Learn project gives a solution to the development problems within the mobile learning field. At the same time, this project promotes the creation of innovative applications and provides the guide for the development of new applications based on the new key factors of mobility in learning: context-awareness, social interaction and integration of e-learning resources.

Mobile devices are a very familiar tool for learners. Their experiences are closer to the use of videogames; watch videos; communication via mobile devices; and use of collaborative technologies (e.g., blogs, wikis, mash-ups and social networks) than to be a mere listener in a master class given by a teacher. Therefore, thanks to the use of these devices, students will take an active role in the learning process in a more interactive and according way to what they are accustomed to use. Learners will feel more motivated and engagement with learning. This idea was remarked by Elliot Soloway, an expert in mobile learning from the University of Michigan: "*The kids these days are not digital kids. The digital kids were in the '90s. The kids today are mobile, and there's a difference. Digital is the old way of thinking, mobile is the new way*."

M2Learn project is intended to give a step further in the state of the art of design of mobile learning applications. It supports the development of innovative mobile learning applications that really complement and enrich the learning experience. This is a learner-centred paradigm that really encourage the "anywhere, anytime", improving the social interactions, providing a personalized educative experience to each learner, and reaching to places where traditional or on-line learning cannot reach. M2Learn Middleware is devoted to help mobile learning to find its place in education, as a complement to traditional and on-line learning instead of replacing them and promoting blended approaches.

ACKNOWLEDGMENT

Authors would like to acknowledge the Spanish Science and Education Ministry and the Spanish National Plan I+D+I the support for this paper with the project TIN2008-06083-C03/TSI "s-Labs – Integration of Open Services for Remote and Virtual Labs"; and the European Union for the Leonardo Project 142788-2008-BG-LEONARDO-LMP "mPSS – mobile Performance Support for Vocational Education and Training Project" and the ERASMUS Project 141944-LLP-1-2008-1-ES-ERASMUS-ECDSP "IPLECS Internet-based Performance-centered Learning Environment for Curriculum **S**upport". Finally authors want to acknowledge the support provided by e-Madrid Project, S2009/TIC-1650, "Investigación y Desarrollo de tecnologías para el e-learning en la Comunidad de Madrid".

REFERENCES

[1] Martin, S., Gil, R., Sancristobal, E., Díaz, G., Castro, M., Peire, J., Milev, M., and Mileva, N., "Middleware for the development of context-aware applications inside m-Learning: Connecting e-learning to the mobile world", Proceeding on The Fourth International Multi-Conference on Computing in the Global Information Technology. ICCGI *2009*. August 23-29, 2009 - Cannes/La Bocca.

[2] Almirall, M. and Rivera, J.M., "Automatic Mobile Learning Contents," Proceedings on the 2009 IADIS Mobile Learning Conference, Jan. 2009, Barcelona (Spain).

[3] Mobile Performance Support System (MPSS): http://mpss.ath.cx/

[4] Bouvin, N., Brodersen C., Hansen, F. and Iversen, O., and Nørregaard P., "Tools of Contextualization: Extending the Classroom to the Field," Proceedings of the 2005 conference on Interaction design and children. Boulder, Colorado, Pages: 24 – 31, 2005.

[5] Bomsdorf, B. (2005) "Adaptation of Learning Spaces: Supporting Ubiquitous Learning in Higher Distance Education", Dagstuhl Seminar Proceedings 05181 Mobile Computing and Ambient Intelligence: The Challenge of Multimedia.

[6] Bouzeghoub, A., Do, K., and Lecocq, C., "Contextual Adaptation of Learning Resources", IADIS International Conference Mobile Learning, pp. 41-48, 2007.

[7] Castro, M., Colmenar, A., and Martin, S. (2010) "Trends of Use of Technology in Engineering Education", Proceeding on the I IEEE Conference on Engineering Education. April 2010, Madrid (Spain)

[8] Skinner, B. (1968). The Technology of Teaching. New York: Appleton-Century- Crofts. 1968

[9] Bower, G. H. and Hilgard, E. R. (1981). Theories of Learning, Fifth Edition, Englewood Cliffs, NJ: Prentice Hall, Inc.

[10] Dufresne, R. J. (1996). Classtalk: A Classroom Communication System for Active Learning. Journal of Computing in Higher Education. 7 (2), pp. 3-47

[11] Martin, S. et al (2009), "Work in Progress: A Mobile Performance Support System for Vocational Education and Training," Proceeding on the 2009 Frontiers in Education Conference, San Antonio TX (USA). October 2009.

[12] Roschelle, J. (2003). Keynote paper: Unlocking the learning value of wireless mobile devices. Journal of Computer Assisted learning, 19(3), 260-272.

[13] Naismith, L., Lonsdale, P., Vavoula, G., and Sharples, M.. Literature review in mobile technologies and learning. NESTA, 2004. Futurelab series, report 11. Bristol: NESTA Futurelab

[14] Piaget, J. (1929). The Child's Conception of the World. New York: Harcourt, Brace Jovanovich

[15] Bruner, J. S. (1966). Toward a theory of instruction. Cambridge, Mass: Belknap Press of Harvard University

[16] Colella, V. (2000). Participatory simulations: building collaborative understanding through immersive dynamic modeling. Journal of the Learning Sciences, 9(4): pp. 471-500

[17] Lave, J., and Wenger, E. (1991). Situated Learning: Legitimate Peripheral Participation. Cambridge, England: Cambridge University Press.

[18] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware Applications: from the Laboratory to the Marketplace. IEEE Personal Communications, 4(5):58{64, October 1997.

[19] Lonsdale, P., Baber, C., Sharples, and M., Arvanitis, T. A context-awareness architecture for facilitating mobile learning. In Learning with mobile devices, research and development, 2004. Edited by Attawell, J. & Savill-Smith C., pp. 79-85.

[20] Rogers, Y., Price, S., Harris, E., Phelps, T., Underwood, M., Wilde, D., Smith, H., Muller, H., Randell, C., Stanton, D., Neale, H.,

Thompson, M., Weal, M., and Michaelides, D. (2002). Learning through digitally augmented physical experiences: reflections on the Ambient Wood project. Equator Technical Report, pp. 1-19 http://www.informatics.sussex.ac.uk/research/groups/interact/papers/pdfs/Rogers_Ambient_Wood2.pdf. Queried on July 2009.

[21] Bandura, A. (1986). Social foundations of thought and action: A social cognitive theory. Englewood Cliffs, NJ: Prentice Hall.

[22] Vygotsky, L. Mind in Society: The Development of Higher Psychological Processes. Edited Cambridge Mass, London: Harvard University Press. 1978.

[23] Martín, S., Bravo, J., Hervás, R., Sancristobal, E., Gil, R., Díaz, G., Losada, P., Castro, M., and Peire, J., "Location-based Services for Mobile Devices at University", IMCL, International Conference on Interactive Mobile and Computer Aided Learning, pp. 1-7, April 16 - 18 2008, Amman (Jordan).

[24] Borenovic, M.N., Simic, M.I., Neskovic, A.M., and Petrovic, M.M., (2005). Enhanced Cell-ID + TA GSM Positioning Technique. The International Conference on Computer as a Tool, 2005. EUROCON 2005. Volume 2, pp. 1176 – 1179.

[25] Youssef, M.A., Agrawala, A., and Udaya Shankar, A., 2003. WLAN location determination via clustering and probability distributions. Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), pp.143-150.

[26] Su, W., Lee, S.-J., and Gerla, M., (2000). Mobility prediction in wireless networks. 21st Century Military Communications Conference (MILCOM 2000). Volume 1, 22-25 Oct. 2000 pp. 491 – 495.

[27] Polito, S., Biondo, D., Iera, A., Mattei, M. and Molinaro, A., 2007. Performance Evaluation of Active RFID Location Systems based on RF Power Measures. IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, 2007 (PIMRC 2007). pp. 1-5.

[28] Bravo, J., Hervás, R., and Chavira, G., "Ubiquitous Computing in the Classroom: An Approach through Identification Process." Journal of Universal Computer Science 11(9): 1494-1504, 2005.

[29] Cheung. B., Stewart. B., and McGreal, R., "Going Mobile with Moodle: First steps," IADIS International Conference on Mobile Learning 2006.

[30] Conde, M., Casany, M.J., Alier, M., and García, F.J., "Back and forth: From the LMS to the mobile device. A SOA approach," 2009 IADIS International Conference Mobile Learning, Barcelona (Spain).

[31] Martín, S., Diaz, G., Sancristobal, E., Gil, R., Castro, M., and Peire, J., "Supporting M-learning: The location challenge", Proceedings on the 2009 IADIS Mobile Learning Conference, Jan. 2009, Barcelona (Spain).

[32] Martín, S., Sancristobal, E., Gil, R., Díaz, G., Castro, M., and Peire, J., "A context-Aware Application based on Ubiquitous Location", UBICOMM 2008. The second International conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, October 2008, Valencia, (Spain).

[33] Hansen, F., and Bouvin, N., "Mobile Learning in Context — Context-aware Hypermedia in the Wild", International Journal of Interactive Mobile Technologies (iJIM), Vol 3, No 1 (2009).

[34] Pascoe, J., "The Stick-e Note Architecture: Extending the Interface Beyond the User", 1997 International Conference on Intelligent User Interfaces, pages 261-264, ACM Order Department, PO Box 12114, Church Street Station, New York, NY 10257, January 1997. ACM, ACM.

[35] Salber D., Dey, A., and Abowd, G., "The Context Toolkit: Aiding the Development of Context-Enabled Applications", Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI), Pittsburgh, PA, pages 434-441, May 1999.

[36] Polson, D., and Morgan, C., "MiLK: The Mobile Informal Learning Kit.Collaborating to design succesful mobile learning applications", IADIS International Conference Mobile Learning 2007.

[37] Hong J. I., and Landay, J. A, "An Infrastructure Approach to Context-Aware", Computing. Human-Computer Interaction, 16(2-4):287{303, 2001.

[38] Hansen, F., and Bouvin, N., "Mobile Learning in Context — Context-aware Hypermedia in the Wild", International Journal of Interactive Mobile Technologies (iJIM), Vol 3, No 1 (2009).

[39] Pascoe, J., (1997). The Stick-e Note Architecture: Extending the Interface Beyond the User, 1997 International Conference on Intelligent User Interfaces, pages 261-264, ACM Order Department, PO Box 12114, Church Street Station, New York, NY 10257, January 1997. ACM, ACM.

[40] Salber D., Dey, A., and Abowd, G., (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI), Pittsburgh, PA, pages 434-441, May 1999.

[41] Polson, D., and Morgan, C., (2007). MiLK: The Mobile Informal Learning Kit.Collaborating to design succesful mobile learning applications. IADIS International Conference Mobile Learning 2007.

[42] Hong J. I., and Landay, J. A, (2001). An Infrastructure Approach to Context-Aware Computing. Human-Computer Interaction, 16(2-4):287{303, 2001.

Sources of Software Requirements Change from the Perspectives of Development and Maintenance

Sharon McGee[1] and Des Greer[2]
School of Electronics, Electrical Engineering and Computer Science
Queens University
Belfast, United Kingdom
{[1]smcgee08|[2]des.greer}@qub.ac.uk

Abstract— **Changes to software requirements occur during initial development and subsequent to delivery, posing a risk to cost and quality while at the same time providing an opportunity to add value. Provision of a generic change source taxonomy will support requirements change risk visibility, and also facilitate richer recording of both pre- and post-delivery change data. In this paper we present a collaborative study to investigate and classify sources of requirements change, drawing comparison between those pertaining to software development and maintenance. We begin by combining evolution, maintenance and software lifecycle research to derive a definition of software maintenance, which provides the foundation for empirical context and comparison. Previously published change 'causes' pertaining to development are elicited from the literature, consolidated using expert knowledge and classified using card sorting. A second study incorporating causes of requirements change during software maintenance results in a taxonomy which accounts for the entire evolutionary progress of applications software. We conclude that the distinction between the terms maintenance and development is imprecise, and that changes to requirements in both scenarios arise due to a combination of factors contributing to requirements *uncertainty* and events that *trigger* change. The change trigger taxonomy constructs were initially validated using a small set of requirements change data, and deemed sufficient and practical as a means to collect common requirements change statistics across multiple projects.**

Keywords- Requirements change; requirements management; project management; card sorting; software evolution; development; maintenance.

I. INTRODUCTION

To some, effective management of changes to software during its lifetime is the key to the effective software project management [43]. While accepting that requirements changes are inevitable during software development, the increased cost of changes later in the development lifecycle [53][2], combined with the threat that volatility poses to project schedule, cost [3][4], and defect rates [5][4], means that requirements volatility constitutes one of the top ten risks to successful project development [6]. Continuing post-delivery, constant adaptation and change is necessary if software is to retain value and remain useful [38]. Viewing software evolution as a continuum from conception to demise is a perspective purported by some researchers [45], though much empirical effort is bounded by a clear distinction between initial development and post-implementation [34][44][35]. Pfleeger's [7] recommendation that "We must find a way to understand and anticipate some of the inevitable change we see during software development" is complemented by Bennett and Rajlich's [35] encouragement to focus upon empirically founded predictive models of maintenance.

Working with an industrial partner, our shared objective is to design and conduct a series of studies that collectively address the challenge of requirements change anticipation. Our longer term aims are 1) To investigate the correlation between the source of change and requirement type, 2) To assess the impact of change source upon requirements volatility and 3) To examine the pattern of source-induced change during development and maintenance. The first step is an exploration of the causes of requirements change, both pre- and post delivery.

For the purpose of change management, it is generally recommended that change requests are held in a database with attributes such as 'origin' and 'change type' [8]. An obvious starting point would therefore be to analyse existing change control databases. However, it has been observed that reasons for change are insufficiently recorded *for the purpose of analysis* [9]. While this statement cannot be said to apply generally, it has also been the experience of the authors. Standardizing data collection across multiple projects regardless of life-cycle phase will not only inform explorative research, but also provide a means by which industrial software providers can take ownership of empirical opportunities. In this study we set out to build a taxonomy of requirements change based on the source of the change, including and comparing sources of change during software development and maintenance. This classification of requirements change sources should be useful as a pick-list (along with other pre-defined attributes) in change diaries across multiple projects within one organization, for the purpose of future analysis.

Thus, the following questions are addressed:-

1. What are the sources of requirements change during software development and maintenance?
2. Can they be similarly classified according to change source domain?

This paper combines a previous study [1] with new results and is organised as follows. Section 2 describes previous studies related to the classification and causes of requirements change. Section 3 outlines the research approach and methods used in this study. In Section 4 we establish the software project categorisation used in this study. Section 5 describes the research process, and illustrates the derived taxonomy. Section 6 discusses our findings with respect to previous work and outlines possible application limitations. Finally we end our paper with conclusions and plans for further work.

II. RELATED WORK

More abstract theories suggest that requirements change because our perceptions of reality differ from actual reality [32], or that the real world is unbounded yet our understanding of the world is both bounded and based upon assumptions which are often invalid [38].

Empiricists, seeking to complement these ideas with more practical support, explore the causes of requirements change by examining evidence during software development and maintenance. Studies designed to classify requirements changes fall into one of two camps. The first are those that advocate the need for a domain-specific taxonomy. Lam et al [10-12], who address the problem of managing volatility by process control, recommend that volatility classification should capture the domain-specific nature of change in order to facilitate change estimation and reuse. This is echoed by Stark [13] who analyses the impact of maintenance changes on release schedule. The following discussion focuses upon those studies such as Harker et al. [14] which propose a more generic re-usable requirements change classification.

A. Software Development Change Classifications.

Harker et al. [14] divide empirically gathered requirements changes into five categories depending upon the source of the change – i) fluctuations in the organization and market environment; ii) increased understanding of requirements; iii) consequences of system-usage; iv) changes necessary due to customer migratory issues or v) changes due to adaptation issues. Based on Harker et al.'s study, an appraisal by Sommerville [15] includes compatibility requirements relating to business process change in place of migratory and adaptation issues. Working from data held in a change control database within an industrial setting, Nurmuliani et al [16] catalogues volatility by type (addition, modification, deletion), origin, and reason for change. Noting that most change requests used in the study had little information about the reason for change, a further study was undertaken using card sorting to classify the recorded changes [9]. This resulted in a list of 'super-ordinate constructs' classified by reason for change – product strategy, hardware/software environment changes, scope reduction, design improvement, missing requirements, clarification changes, testability and functionality enhancement.

As can be seen there is little agreement in the terminology used for classifying requirements change, and it would seem at first sight that studies to date have little commonality. This may be due to the different contextual basis of the studies, or perhaps that classification was established at different levels. It is possible, for example, that Nurmuliani et al.'s change reason of 'missing requirement' is included within Harker el al.'s change source of 'increased understanding'.

A genre of studies related to requirements engineering risk and uncertainty is also of relevance. Mathiassen et al. [17] classify requirements engineering risks by reliability, complexity and availability, and relate these to appropriate techniques.

B. Software Maintenance Change Classifications

Much empirical and theoretical work re-uses or builds upon Swanson's classification [41] of maintenance changes [34][44][46], which includes corrective, adaptive and perfective changes. Chapin et al. [42] provide a thorough review of literature referring to maintenance change types, and propose a new classification which is an extension and clarification of previous work, and is based upon observed activities. These include changes to documentation, code, and business rules. Incorporating both errors and enhancements, this classification focuses not upon the reason, cause or source of the change, but instead upon the type of change being made. Both Kemerer & Slaughter [44] and Heales [37] take a different approach and classify changes according to what is being changed. From a theoretical view point, Perry [39] discusses the dimensions of change and concludes that software development imitates the 'real world' by the creation of a 'model' from which we abstract an 'understanding of system requirements'. These are subsequently implemented upon a foundation of sometimes weak 'technical theory'.

Due to the divergence of change sources compiled in these studies, none of the classifications exclusively met the needs of the subsequent stage of this research. However, their findings, together with requirements change causes derived from other studies are used to provide a collection of change constructs upon which to base our classification effort. A full list of change source constructs elicited from the literature can be found in the appendix.

III. RESEARCH APPROACH

This study is the first of a family of studies [18] employing a collaborative research approach, in that it seeks to contribute to the body of knowledge in this area, whilst answering to the need of our industrial partner to better understand, manage and measure requirements changes. Collaboration with industry is generally recommended to ensure relevance and better transfer of research results [19]. In this instance the industrial partner gave of their time to provide expert knowledge of software project management and product maintenance.

A. Preliminary Studies

To explore the scope and complexity of the problem, and decide upon appropriate and effective research methods, a number of initial investigations were undertaken. Three unstructured interviews, during which project managers in the main reflected upon their current project, demonstrated the need for a focus for 'memory-jogging'. A subsequent self-administered questionnaire exposed difficulties with change construct interpretation and understanding, and a review of a change database revealed that not all changes were recorded, particularly those relating to the technical solution. Therefore, methods were sought that would maximise the opportunity for consensus building, provide a visual basis for brainstorming, and maximise the potential for knowledge sharing and exchange.

The unit of analysis is our industrial partner organization. Participants were sampled from the company's Project Managers and Maintenance Engineers by convenience, within the stratum of those with at least 12 years experience in IT.

B. Organisational Context

Our industrial partner in this research employs 300 staff, 136 of whom are involved with software development. They have 6 offices around the UK and Ireland and deliver IT solutions to clients across both the public and private sectors. Most of their contracts involve a single customer and roughly 80% of these relate to governmental work. Nearly all project managers are Prince2 certified and work with a range of traditional and agile methodologies.

C. Workshops

In requirements engineering, group elicitation techniques such as workshops aim to foster stakeholder agreement and buy-in [20], and are a mechanism whereby individuals can make decisions through the consensus building leadership of a facilitator [21]. In view of this, they were used to familiarize all participants with the constructs, come to a common understanding of their meaning, and reach a consensus of opinion at the end of the study regarding the structure of the taxonomy to be used.

D. Card Sorting

Card sorting is a knowledge elicitation technique which involves categorizing a set of cards into distinct groups according to a single criterion. Each card represents a construct which can be expressed in words or pictures, and participants are invited to place them into related groups. The categorisation may be left to the participants (open sort) or pre-determined (closed sort). Maiden & Rugg [21] suggest that card sorting is one of the most suitable techniques for acquiring knowledge of data (in contrast to knowledge of behaviour or process). Further, Rugg and McGeorge [54] argue that card sorting overcomes one of the disadvantages of the repertory grid method of categorisation since this uses Likert-type measurements to capture participant responses and is not well suited to nominal scale data. However, the repertory grid approach does lend itself easily to statistical analysis, which is one of the challenges of card sorting [22]. Other more semantic disadvantages include the need for careful selection and naming of cards in order to ensure cross participant construct understanding, and the potential disparity of group labelling during open sorting. However, the use of extensions such the Delphi method (each participant iteratively improves a proposed hierarchy) [55] can overcome some of these difficulties. Most analogous to this approach is affinity diagramming which is similar to card sorting except that the focus is upon reaching a consensus, and therefore consists of a single card sorting exercise with a number of participants. However, by contrast to singular participant card sorting, taking this approach will mean that the differences in participant perspectives will be lost. Salient amongst the advantages of card sorting are its simplicity, focus on participants terminology, and ability to elicit semi-tacit knowledge [22]. A special edition of the journal 'Expert Systems' in 2005 [23] was dedicated to the subject and it has widespread use in psychology, knowledge engineering and requirements engineering. Accordingly, single participant card sorting with supporting aforementioned workshops for terminology understanding and analysis consensus was deemed an appropriate approach to the derivation of a taxonomy of change sources.

IV. SOFTWARE PROJECT CATEGORISATION

In order to accommodate and compare sources of change pertaining to all phases of the software lifecycle, it is first necessary to clearly define what we mean by development and maintenance. Noticing that there is some terminological disparity in the literature [35], we firstly derive a character based project categorization founded upon existing studies. It is from this basis that we establish understanding between academic and industrial research team members and consider the validity of the results of this study.

A. Software Evolution

Lehman's influential and continuingly relevant work on software change [38][45] brought the term evolution into common research usage. Defined as "the dynamic behaviour of programming systems as they are maintained and enhanced over their lifetimes" [47], Belady & Lehman are deliberately inclusive of all stages of the software lifecycle, including initial development [38][43]. Subsequent to this work, authors have applied the term to development [48], used it as a substitute for maintenance [34][44], and proposed that it refers to a period of time between initial development and servicing [35]. Noting that the term lacked a standard definition, Bennett & Rajlich [35] sought to clarify its meaning by asking the question "What is maintenance?" and proposing a staged model for the software lifecycle [35]. This theory derived model promotes the latter view that software enters a phase of evolution following initial delivery and stops evolving once it is no longer feasible to make requirements changes. Subsequently the software enters a period of servicing when only minor corrections are made. Bennett & Rajlich claim that from a research perspective each stage has "different technical

solutions, processes, staff needs and management activities". Therefore empirical research should firstly ensure context is specified, and secondly explore the best solution for each stage. Interestingly, in a retrospective examination, Lehman & Ramil observed that their empirical research supported the staged model [38].

B. Software Development and Maintenance

The term maintenance has been defined by the IEEE [49] as "The modification of a software product after delivery to correct faults, to improve performance or other attributes or to adapt the product to a modified environment". As argued by Godfry & German [36], this definition is not representative of all post-delivery activity, and the semantic inference of the term evolution more closely reflects the changing nature of software, and in particular accounts for requirements changes. Nonetheless, the term maintenance is still used widely, though not consistently. Kitchenham et al. [46] developed an ontology of maintenance in which two scenarios are outlined. The first scenario (A), more commonly understood as evolutionary development, is included since in this instance the incremental nature of software delivery necessarily implies that there is a portion of software in the post-delivery phase. The second scenario (B) represents the case where activity concerning software change is facilitated by a maintenance organisation distinct from that of development. The second of those is the more traditionally accepted view of maintenance and the context of much 'maintenance' research. As an interesting aside, Basilli [50] considers software re-use and surmises that from a re-use perspective all development can be considered maintenance due to the prevalence of components usage. Chapin et al. [42] assert that a classification of requirements change types, more traditionally ascribed to maintenance, can equally be applied to software development, and that this project nomenclature is relevant only in so far that it is prevalent in industry. Indeed, in that environment, deciding whether a project is 'maintenance' or 'development' is merely a question of project funding and contractual agreement. Supportive of this contention is the observation that the maintenance process ontology from Kitchenham et al. [46] is derived from and bears direct semblance to a development process ontology proposed by de Almeida et al. [52]. The activities involved in managing change (evaluation, impact analysis, approval, implementation, regression testing) and the supporting processes of configuration management, requirements traceability and release planning are beneficial elements of change management, irrespective of life-cycle phase. However, Chapin et al also assert that the level of effort consumed by these activities depends upon whether they occur in a development or maintenance environment, and that recognition of the differences between the two phases will lead to more realistic measurement and work evaluation [51]. Kemerer & Slaughter suggest that the types of changes seen during longitudinal post-delivery studies are not homogeneous. Further empirical research may reveal predictable patterns of evolutionary change which would contribute to knowledge regarding the software lifecycle [44].

It is apparent therefore that there is some commonality of change process and activity shared amongst projects in phases termed development, evolution and maintenance. However, the observations made by Bennett & Rajlich [35] Chapin [51] and Kemerer & Slaughter [44], who advocate the benefit of differentiating between life-cycle phases, are of sufficient significance to warrant empirical investigation.

While an exhaustive account of the comparison between development and maintenance is out of the scope of this study, the categorisation illustrated in Table 1 was derived for the purposes of this and future empirical studies. It combines the staged model proposed by Bennett & Rajlich [35], Kitchenham et al.'s maintenance scenarios [46] and Chapin's classification of change types [42]. The division between development and maintenance was drawn to reflect the importance of the factors relating to team knowledge, stability and responsibilities [46][51], coupled with the distinct contractual governance prevalent during 'product upkeep' and 'servicing'. From Table 1 we derive the following definition of software maintenance.

Maintenance projects are those that:-

1. Employ staff whose work assignment is distinct from that of pre-delivery development, and whose application domain knowledge is not assumed.
2. Operate under a clearly defined support contract.
3. Involve activities of product correction and enhancement to production software.

V. TAXONOMY DEVELOPMENT

This section describes the process of taxonomy development. Upon agreement of the proposed categorisation, a consideration of the sources of requirements changes observed during software development informs the organisation of an initial change source classification. This is followed by further study incorporating sources of change associated with maintenance projects.

A. Project Categorisation Clarification

With one project manager present, the proposed project categorisation was reviewed. Two post-delivery support contracts were examined and it was noticed that small changes termed 'enhancements' were permitted under the terms of both contracts provided that they did not exceed an agreed (contract-specific) cost ceiling. These would be undertaken by a member of the organisation's maintenance team and scheduled in accordance with maintenance priorities. Provision was made in both contracts for further enhancements, whose costs were estimated to be in excess of the ceiling, which would require the agreement of a further contract. This work would be undertaken by a dedicated software development project team.

TABLE 1 DEVELOPMENT AND MAINTENANCE PROJECT CATEGORISATION

	Development		Maintenance	
	Development	Iterative Delivery	Product Upkeep	Servicing
Naming Convention	Initial Development[1]	Evolution[1] Maintenance Scenario A_2	Evolution[1] Maintenance Scenario B_2	Servicing[1] Maintenance Scenario B_2
Staff Roles	Pre-delivery only	Pre and Post-delivery.	Post delivery only	Post-delivery only
Software Engineer Knowledge	Domain and project-specific technical knowledge inherent	Continuity of domain and project-specific technical knowledge.	Some Domain and project-specific technical knowledge required but not assumed.	Domain and project-specific technical knowledge not required or assumed.
User Support	N/A	Feedback through requirements analysis activities	Help/Support Desk Service Level Agreement	Help/Support Desk Service Level Agreement
Types of changes	All types	All types	All types	Corrective

[1] Bennett & Rajlich [35]

[2] Kitchenham et al. [46]

Under the proposed project categorisation, the enhancement work falling under the maintenance contract would be termed 'maintenance' while the work requiring further funding arrangements would fall under 'development'. Since both cases would involve requirements changes made to post-delivery software, this supports Chapin et al.'s comment that industrial naming convention is a matter of budget considerations [42], and highlights the potential for confusion when understanding the context of research studies. It was emphasised by the project manager that any development project emerging from a maintenance contract would necessitate more depth of requirements analysis processes than that required by the 'mini projects' undertaken under the terms of the maintenance contract. The project categorisation as proposed was used in the remainder of the study.

B. Development Change Source Constructs

Electronic keyword searches were performed to assemble candidate academic papers, industrial articles, and text books. Citations referring explicitly to requirements change/evolution sources/causes/uncertainty/creep/risk were followed in a forward direction in search of the initial source. This resulted in a total of 73 papers and text books which were reduced to a final 14 sources by the criteria 'software development' with 'discovered empirically' or 'seminal work/text book'. As 'seminal work' was subjectively assessed, this cannot be considered a systematic review. However, without this criterion, papers such as 'Issues in requirements elicitation' [24] would not have been included. The authors felt that this would be an oversight.

During the collation of change source constructs it became apparent that reasons for change such as 'diverse user community' and 'New tools/technology' were often gathered together under the umbrella term 'cause' [14, 15, 25]. Clearly there is a distinction between *uncertainty* giving rise to change and events that *trigger* a change. Whilst an event can lead to a change without preceding uncertainty, uncertainty can not result in a change unless an event resolves or intervenes to mitigate the risk of uncertainty. It could be argued that change is 'caused' by a combination of uncertainty and trigger, although in reality causation cannot be proved to arise from one, other or both due to the presence of confounding environmental factors. Accordingly, uncertainties and triggers, collectively referred to as sources of change, were separated. This separation was not difficult since in most cases the semantics of the constructs related to an *event* (trigger) or a *situation* (uncertainty).

C. Initial Workshop – development construct consolidation

The first workshop taking 2 ½ hours introduced the constructs to 3 project managers and each construct was clarified for meaning. In so doing, constructs sharing a similar meaning were amalgamated, and those represented by other constructs at a finer level of granularity were removed. Additionally, constructs such as 'New Functional Feature', which would necessarily arise as the consequence of resolved uncertainty or opportunity were also removed. The most debated of the constructs was 'changes following prototyping'. Though quoted as a cause of change, it was the opinion of the participants that this change source should be thought of as a technique, having no more causal

significance than other techniques such as 'requirements inspections'. Either all techniques should be included and constructs added accordingly, or constructs pertaining to increased understanding should represent the techniques. The final consensus favoured the latter argument, though the addition of technique constructs remains a question for further research. Four triggers were added and as a result of this process, the number of constructs was reduced from 73 to 46. Making the distinction between trigger and uncertainty was confirmed both to be viable and useful, since triggers could more easily be attributed to requirements changes. The constructs are listed in the Appendix under the headings 'Development Trigger Construct' and 'Development Uncertainty Construct'. What remained was to classify the triggers and assign uncertainty constructs accordingly, thus endorsing the classification and confirming that uncertainties had corresponding change events.

D. Participant Card Sorting (development)

Individual card-sorting ensured that the opinions and contribution of each project manager were represented. The process was first validated by a pilot card sort with 1 project manager and 1 researcher. Each card sorting session was audio-recorded and reviewed, and photographs were taken of card classifications. This process took between 45 minutes and 1 ½ hours.

Each of the 23 development trigger constructs (as they appear in the Appendix) was written on a card and assigned a random number which could be seen clearly in the photographs. Six participants were asked to classify the triggers according to their source.

All participants classified the triggers into between 3 and 5 categories, and there was homogeneity between the classifications, although in all cases they were named differently. For example, one project manager referred to 'ownership' of the categories; another used process labels such as 'customer interface' and 'Requirements engineering'. Naming convention aside, 14 of the 23 constructs were placed in the same pattern by all participants, that is, co-resided in 3 groups. Notably, differences of card placement related to degree of granularity of classification. For example, 4 participants grouped 'increased customer understanding' and 'first engagement of customer' alongside constructs relating to understanding the technical solution. The classifications of the remaining 2 project managers conveyed the importance of distinguishing between changes that arose due to increased understanding of the problem, and those relating to the technical answer to that problem. Only 1 classification, illustrated the distinction between market factors and those concerning the customer organisation, the remainder considering them similarly 'external' to the project.

E. Second Workshop- Consesus building

Four project managers attended a second workshop lasting 3 ½ hours. Stimulating and interesting discussion resulted in a unanimously agreed trigger taxonomy to which uncertainty constructs were attributed.

Beginning with 3 untitled groups containing a total of 14 trigger constructs, it remained to come to a consensus of opinion regarding the remaining 9. As observed by one of the participants, the granularity differences were a matter of perception. For example, as a project manager, constructs such as 'market stability' or 'customer organisation strategic change' were equally external to their control. However, from the perspective of a customer, this is perhaps not the case. Therefore, the final taxonomy was built according to the variance of classifications made during the card sorting procedure. Consequently, a taxonomy comprising 5 groups was derived and agreed.

These groups comprised the change domains illustrated in Table 2. Uncertainty constructs were attributed to their associated domain. At this stage several additional uncertainty constructs were added. Most notable amongst these were technical uncertainty, and technical complexity of solution. Though considered general project risks [26], they had not previously been recognised as a source of requirements change. This may be because they do not alter the vision of the problem, but rather the way in which the problem is addressed.

TABLE 2 CHANGE DOMAINS AND DESCRIPTIONS

Change Domain	Description
Market	Differing needs of many customers, government regulations, external to project.
Customer Organisation	Strategic direction of a single customer, customer organisation considerations, external to project.
Project Vision	Problem to be solved, product direction and priorities.
Requirements Specification	Specifying the requirements of the established problem.
Solution	Technical answer to problem.

Nonetheless, as discovered by Curtis et al. [25] 'creeping elegance' is a source of change and a risk to budget and schedule slippage. There was some debate about the positioning of 'project size'. Initially considered to be a risk to change in all domains, it was further reasoned that size has an effect, due to the increased difficulty of conceptualizing the problem. Therefore 'size' was placed in the domain of project vision.

F. Maintenance Change Source Constructs

Of the initial 73 papers, 11 contained references to post-delivery requirements change causes. Having established a project categorisation, there was difficulty applying it to other studies since none of them made reference to contract conditions or staffing arrangements. Only the criteria 'changes to production software' was used. Interestingly there were significantly fewer empirical studies examining sources of requirements changes post-delivery than during development, despite the high proportion (75% [35]) of

enhancement work carried out during that time. It was noted that many studies examining risks or uncertainty within the maintenance environment were exploring risks to maintenance *change productivity* rather than *change likelihood*. Perhaps this indicates support for the argument of Kitchenham et al. [46] that one of the major differences between development and maintenance is that development is requirement-driven and maintenance is event-driven. In their words, "This means that the stimuli (i.e., the inputs) that initiate a maintenance activity are unscheduled (random) events". Perhaps prohibitive to investigation is the limited value that an exploration of change causes would yield, should the contention be empirically proven that maintenance change is stimulated by random events. Once again, separation of trigger and uncertainty presented no difficulties.

G. Third Workshop - Maintenance construct consolidation

Consolidation of maintenance constructs during a workshop consisting of a researcher and 2 maintenance team members, taking 2 hours, followed the same process as development constructs, reducing an initial set of 36 constructs to 11 triggers and 12 uncertainties (see the appendix). This is in marked contrast to the number of constructs concerning development projects elicited from the literature. Many of these constructs ignited lengthy discussion. Of particular note was that many of the uncertainty constructs were likely to introduce error rather than requirements change. Those in that category included 'maintenance team instability' and 'maintenance team knowledge'. By contrast, these team-related constructs had been considered sources of requirements change during development. It was believed that the perceived more limited business knowledge required by maintenance engineers coupled with the reduced need for requirements analysis processes to implement 'mini changes' meant that these team attributes had no significant effect upon requirements changes. Also interesting was the observation that some uncertainties such as 'economic climate' altered a projects capacity to make change, rather than invoking change. The construct 'system usage' was removed since it was seen as an 'activity' during which an alternative change source may manifest (such as 'increased understanding'), rather than a cause of change itself. This bears comparison to the removal of techniques during the development construct consolidation. The 9 added constructs included Commercial Off-the-shelf Software (COTS) usage, which was felt to be a contributor to requirements change, due to the need to react to new COTS opportunities and release functionality. 'Number of interfaces' and 'Number of functions' were added to reflect system complexity, as it was thought that system complexity doesn't in itself lead to changes of requirements during maintenance, though it would during development when requirements are still being understood. 'Function Usage' was also added since system functions used more frequently are prone to higher levels of change.

H. Fouth Workshop - Card Sorting (Maintenance)

Since the intention was to discover if sources of change during maintenance and development projects could be similarly classified it was decided to perform one closed card sort [22] within a workshop setting. Provided with the change domains derived previously and described in Table 2, two maintenance engineers were asked to ascribe the maintenance change constructs to one, many, or none of the change domains.

The participants found the trigger constructs easy to attribute, though some of the uncertainty constructs resided in both *requirements specification* and *project vision*. A higher number of users, or a high level of function usage may uncover opportunities to improve the way in which the system requirements have been implemented, or reveal new desires and needs. Similarly, the discussion surrounding 'project size' during the consolidation of constructs pertaining to software development, 'system age' was initially thought to reside in all domains. However, further consideration led to the conclusion that, while an older system is more likely to require functional updating without changes to the surrounding market or customer environment, the system could retain value in its current state. By itself, the age of a system will only affect performance or data storage issues requiring *solution* maintenance. The term 'semantic relativism' described by Heales [37] as 'generation of language construction' was placed in the domain of project vision, although the participants felt that as a concept it had less relevance than the other uncertainties, and was difficult to evaluate. No constructs remained unplaced.

I. Fifth Workshop – development and maintenance taxonomy consolidation and comparison

During this workshop, taking 3 hours, both project managers and maintenance engineers were brought together to compare and consolidate the two previously derived taxonomies. The following agenda items were agreed:

1. Identify and consolidate corresponding maintenance and development constructs a) within the same domain and b) within alternative domains.
2. Review constructs to determine if those located in a single taxonomy related equally to both.

1) Maintenance and Development Construct Consolidation.

Seven of the 12 maintenance related triggers, and 6 of the 12 maintenance related uncertainties were semantically synonymous, though named differently to development related constructs. Those that resided within the same change

domain retained the naming convention used in the software development change source taxonomy. There was some discussion regarding the naming and placing of 'Number of interfaces' and 'number of functions' which had been placed in both the domains of *project vision* and *requirements specification* by the maintenance engineers. These represented factors contributing both to 'project size' and 'logical complexity of problem' residing in the domains of *project vision* and *requirements specification* respectively. The ensuing discussion led to the recognition that while these constructs embodied a similar concept, the difference lay in the effects of the uncertainty. For example, while 'project size' affected the capability of the development team to understand and model the problem, from the perspective of the maintenance team it increased the likelihood for change discovery during maintenance.

2) Development and Maintenance construct review

Only two sources of maintenance related requirements change - 'semantic relativism' and 'response to gap in market' were deemed applicable to initial software development. However, when taking iterative development into consideration, the constructs relating to system usage also became relevant. It was argued by the project managers that from their perspective 'alter performance' is a (non-functional) requirement change that would happen in response to a market or customer need and was therefore not a cause of requirements change. From the perspective of maintenance, 'alter performance' represented the pro-active changes made to deter system degradation or promote further usage. Therefore 'design improvement/solution elegance' was a more appropriate construct. Those remaining within the realm of software maintenance related only to system age.

However, many of the constructs pertaining only to development applied also to maintenance. Indeed, it was agreed that, aside from 'cost/schedule overrun', only those constructs relating to the ability to understand the problem related solely to software development. However, it was also noted that many of these sources, particularly those in the domains of *market* and *project vision* would result in the initiation of a new product release. So while the change may be incurred during maintenance, it will be realised by a software development team. Confirming the insight arising from the discussion regarding project size, a number of the uncertainties relating to software development were applicable also to maintenance, though with a distinct difference in effect. For example, during development high quality of communication with customers affected the clarity of the shared understanding of the problem, thereby *reducing the likelihood* of subsequent requirements changes. By contrast, during maintenance the quality of communication *increased the probability* of change recommendation, and hence had an effect upon system longevity.

The resulting taxonomy is shown in figure 1. The reader is referred to the appendix for full construct tracing from research origin to construct consolidation and comparison. The change domains relate to both triggers and uncertainties. There is a many to many relationship between the uncertainties and triggers within each change domain and in many cases a 'chain' of uncertainties may culminate with a trigger event. Those constructs marked '(D)' apply solely to development, while those marked '(M)' are relevant only to maintenance. Of interest was the observation by the project managers that the structure of the domain also reflects the amount of control they have of the uncertainties, with least control at the top - 'Market' and tighter control at the bottom – 'Solution'.

J. Validation of Change trigger Constructs

The capability of the change trigger constructs to describe the source of a change was initially validated by one of the participating project managers who used a small sample of changes (13) across two development projects to ensure that each had a corresponding trigger which accurately reflected the source of change. No changes were made at this stage. This taxonomy will firstly be used within the context of development, assessed for informative capability and internal validity, before considering the broader scope of applicability. Further validation of this taxonomy is the subject of an on-going study using a current project.

VI. DISCUSSION

This section evaluates the taxonomy thus derived with respect to previously published change classifications, explores the implications of the study with respect to the comparison between development and maintenance and outlines some possible limitations of this work.

A. Comparison with Previous Classificatons

The classification proposed in this study bears little synergy with change reasons derived by Nurmuliani et al. [16] as many of these reasons such as 'missing requirement' and 'new functional feature' were considered to be consequences of other events, rather than sources of change. By comparison, there is some resemblance to the classification of change sources defined by Harker et al. [14]. In particular, a combination of *market* and *customer organisation* domain sources equate to their 'mutable' class defined as "changes that arise in response to demands outside the system". By making the distinction between changes that occur in response to market demands, and those answering to customers' organisational considerations, the taxonomy developed here reflects the difference between customer-driven and market driven software development. Harker et al.'s 'emergent' requirements, "direct outcomes of the process of engagement in the development activities", correspond to constructs in both the *project vision* and *requirements specification* domain. In differentiating between *project vision* and *requirements specification* domains we are recognising the difference between variation in the product to be developed and change due to better understanding of the problem. This is an important distinction as it can support decisions regarding requirements elicitation techniques and rigour of documentation.

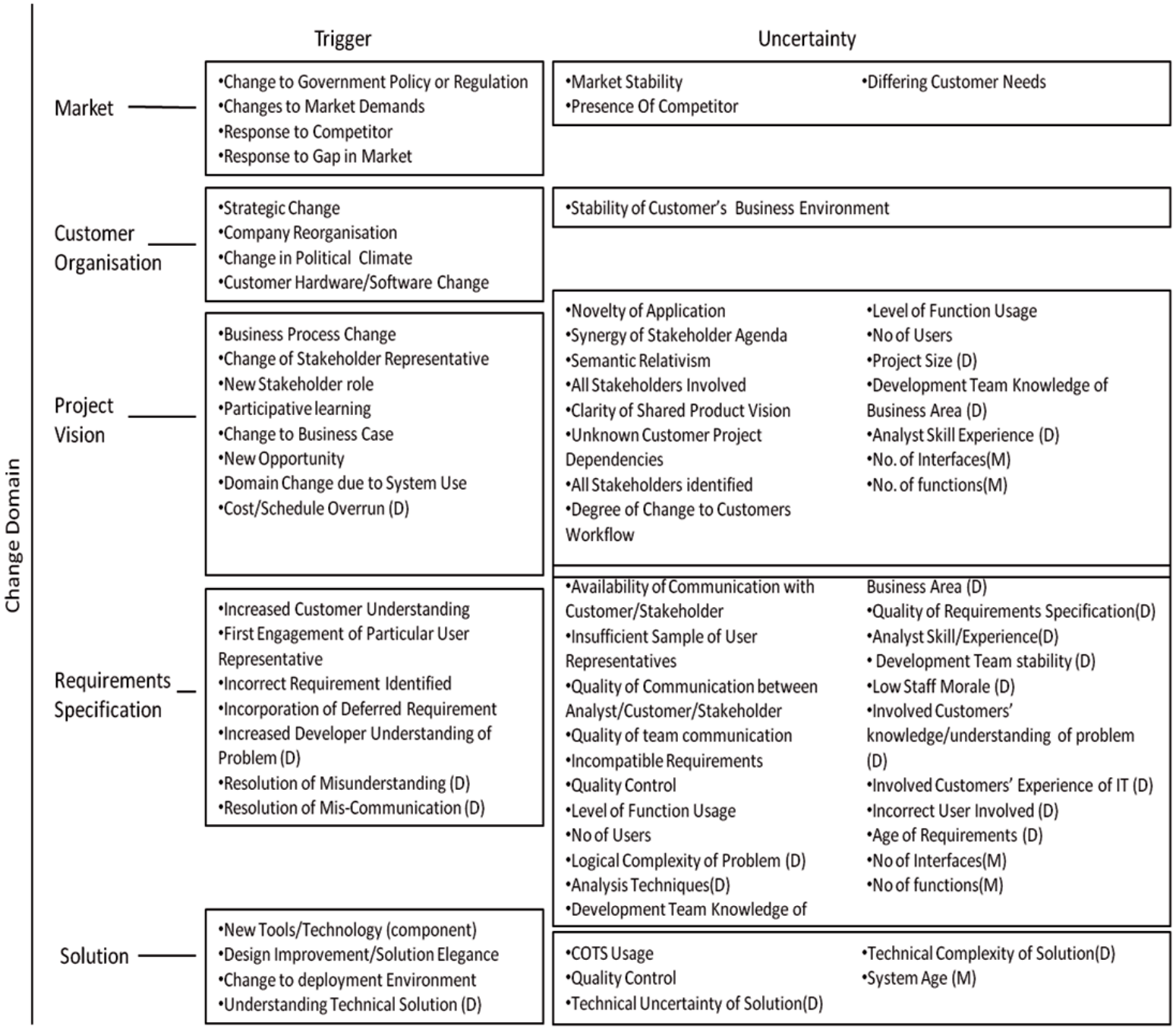


Figure 1 Requirements Change Source Taxonomy

There are no analogous domains within this taxonomy for the remainder of Harker et al.'s categories. These include prototyping or system usage, adaptive requirements and migration requirements, which were reasoned to be techniques, activities, or new requirements. Sommerville's classification [15], while including 'mutable', emergent' and 'consequential' change (system usage) also removes adaptive and migration requirements. Instead 'change to business process' form a category which is included here in the project vision domain, since these types of changes result in a change of product direction. The solution domain has no direct parallel in any classification but reflects the reality that changes to the technical solution, though perhaps less visible, pose a risk to timely development.

While there are some differences in contained constructs, requirements availability as defined by Mathiassen et al. [17] corresponds to *requirements specification* although constructs relating to requirements complexity and reliability

are included in both *customer organisation* and *project vision.* That said, further categorising these domains according to reliability and complexity would allow the findings of both studies to be combined, thus relating technique to change source domain.

A comparison can be drawn between this taxonomy and classifications of change types during maintenance [42]. Excluding error handling, the taxonomy derived here includes constructs in the *solution* domain relating to perfection and adaptation while enhancements are further classified according to the remaining change domains. There is an encouraging parallel with Perry's software development domains [39]. While the 'real world' is represented here in both the *Market* and *customer organisation* domains, Perry's 'model of the real world', 'derived specification' and 'underlying theory' correspond closely to *project vision*, *requirements specification* and *solution* respectively. Thus, to an extent this study corroborates Perry's theoretical model with empirical evidence, and furthers understanding of the nature of the domains.

Of particular significance for our on-going research is that by comparison to the descriptive or uncertainty based nature of previous work, the clearly defined constructs within each change source domain allow comparative source data to be attributed to change databases. Therefore it would be possible to assess the impact on the project of a particular change source such as 'new stakeholder' or 'first engagement of customer representative', giving software providers some empirical data with which perhaps to leverage customer involvement. Further, it would be possible to assess the level of change in each change source domain. Should, for example, a high proportion of changes come from the domain of *project vision*, this would indicate the vulnerability of the 'problem' to change, thereby empirically illustrating the need for more 'agile' creative processes.

B. Comparison between development and Maintenance

Having derived a project categorisation based upon the work of Kitchenham et al. [46], Bennett & Rajlich [35] and Chapin et al [42] (refer to Table 1), the taxonomy derived in this study verifies that many requirements change sources are similarly relevant to development and maintenance. Thus supporting Bennett & Rajlich's [35] observation that software evolves during both iterative development and maintenance, the differentiation presented by Kitchenham et al. [46] between the two scenarios is also reflected in this study. Sources of change arising due to continued understanding of the requirements are attributable to iterative delivery (scenario A), while those relating to system age are relevant only to product upkeep and servicing (scenario B). However, this observation relies upon a definition of maintenance that includes only minor enhancements, which are represented in Bennett & Rajlich's [35] model, not as a lifecycle stage, but as an iterative element of evolutionary product versioning. Refuting Kitchenham et al.'s contention that maintenance changes are event driven, while changes during software development are requirement driven [46], the separation of triggers and uncertainties and their pertinence to both development and maintenance, reveals that changes during software development can be equally reactionary to external events. The pro-active approach to maintenance described by one of the maintenance engineers in this study suggests that maintenance changes, like those during software development, aren't entirely event-driven, but transpire as a result of a combination of uncertainty, event and pro-active change discovery. Whilst the change sources illustrated in the taxonomy indicate the similarities between development and maintenance, further exploration of the consequences of the uncertainties may reveal differences to project risk.

C. Limitations

Generality of results are often sacrificed for richness and complexity, reflecting an inherent conflict between internal and external validity [19]. Given the disparity between both terminology and published change taxonomies combined with the debate among the participants of this study, it could be argued that change classification is by nature a subjective assessment. Motivated, however, by the potential for improvement to requirements change visibility and management, modelling change sources is a worthy initiative. The collaborative approach taken here has led to an internally usable model and reflects Sjoberg's et al.'s recommendation [19] to "formulate scope relatively narrowly to begin with and then extend it gradually". Therefore no claims can be made with regard to external validity beyond the boundaries of this study, and in particular to projects employing alternative delivery models such as service oriented and cloud computing. However given that the constructs were drawn from a variety of empirically based studies, it is plausible that the results apply to projects similarly adhering to a more traditional development lifecycle. The initial constructs are provided here, along with methods description such that it should be possible to replicate this study. Given the collaborative nature of this research, and its immediate applicability, it has a high level of relevance.

VII. CONCLUSION AND FURTHER WORK

This study set out to explore, classify and compare the causes of requirements change during software development and maintenance. A review of the terminology highlighted the fuzzy distinction between projects termed 'development' and those referred to as 'maintenance'. The disparity of terminology in the literature is complemented, and to some extent explained by the lack of distinction observed in industry. Project nomenclature is decreed dependent upon the size of the proposed change, and the supporting funding agreement. This carries the implication that research in the field of software development may apply to software maintenance and vice versa. Further, that establishing context in empirical research requires more than a reference

to development or maintenance in a research article. A somewhat narrower view of maintenance was defined in this study which excluded evolutionary development and reflected the naming convention used by our industrial partners.

Expert knowledge of experienced project managers and maintenance engineers was used to consolidate and classify change source constructs elicited from the literature. An initial study based on sources of change relevant to software development resulted in a classification which made the important distinction between uncertainty (situation) and trigger (event) giving rise to change. In itself, this taxonomy supports project risk visibility and facilitates the collection of clearly defined change source data. In differentiating between source domains pertaining to market, customer and project vision a software provider using this taxonomy can assess the level of changes that are arising due to a *change in the direction or vision* of the problem, by contrast to those pertaining to an *increased understanding* of the problem to be solved. In so doing, project managers can make use of internal empirical data to support process and technique selection, and risk management.

A second study incorporating significantly fewer sources of requirements change during software product maintenance classified the constructs according to the change source domains previously derived. The sources of maintenance requirements change could easily be attributed to the change domains defined in the initial study, and many of the constructs had been included within the original taxonomy. A comparative exploration revealed that most of the change constructs applied to both development and maintenance, though it was observed that the effects of the uncertainties differed, and that some of the changes incurred during maintenance would necessitate a new product release requiring software development. Those constructs relevant solely to development related to requirements and domain understanding, while those pertaining only to maintenance were concerned with system age. By contrast to the contention that software maintenance changes are event driven while development changes are requirement driven, an implication of this study is that changes to requirements are driven by a combination of event and uncertainty during both development and maintenance. Further, opportunities for requirements change may be sought pro-actively in both situations.

This study was founded upon previously published requirements change taxonomies, thus evaluating and building upon their efforts. Therefore it addresses the problems of divergent change source constructs, and reasons that some of the classifications previously described as 'causes' were either consequences of other changes, types of requirements, or more abstract concepts less easy to evaluate.

Having answered the questions posed in this study, it is now possible to further our research and begin exploring what kinds of requirements are more susceptible to change arising within the change domains defined in this taxonomy. This is currently on-going with our industrial partner. A further study is envisaged which will explore patterns of requirements change throughout the evolutionary progress of software development and usage. The theoretical aspect of the work presented here may contribute to ontological studies, and open more issues in relation to emerging paradigms such as dynamic updates in Service Oriented Architecture and alternative delivery models such as cloud computing. In the meantime the derived taxonomy can assist practically in the identification and analysis of requirements volatility and has particular relevance to customer driven software development especially those working within the government sector.

ACKNOWLEDGMENT

We would like to thank the project managers and maintenance engineers whose valuable time was devoted to the organization of this taxonomy.

REFERENCES

[1] S. McGee, D. Greer, "A Software Requirements Change Source Taxonomy", proc. 4th Intl. Conf. Software Engineering Advances, Porto, 2009.

[2] B. Williams, J. Carver and R. Vaughn, "Change Risk Assessment: Understanding Risks Involved in Changing Software Requirements", Proc. International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, 2006.

[3] D. Zowghi and N. Nurmuliani, "A study of the impact of requirements volatility on software project performance", Proc. Ninth Asia-Pacific Software Engineering Conference, 2002.

[4] S. Ferreira, F. Collofello, D. Shunk, G. Mackulac and P. Wolfe, "Utilization of Process Modeling and Simulation in Understanding the Effects of Requirements Volatility in Software Development", International Workshop on Software Process Simulation and Modeling, Portland, Oregon, 2003.

[5] T. Javed, M. Maqsood and Q. Durrani, "A study to investigate the impact of requirements instability on software defects", ACM Software Engineering Notes, 29, 3, 2004.

[6] B. Boehm," Industrial Software Metrics Top 10 List", IEEE Software, 4(5), 1987.

[7] S. L. Pfleeger, "Software Metrics: Progress after 25 Years? ", IEEE Software, 25(6), 2008.

[8] K. Wiegers, Software Requirements, Microsoft Press, 2003.

[9] N. Nurmuliani, D. Zowghi and S. P. Williams, "Using card sorting technique to classify requirements change", Proc. 12th IEEE International Conference on Requirements Engineering, Kyoto, Japan, 2004.

[10] W. Lam, M. Loomes and V. Shankararaman , "Managing requirements change using metrics and action planning", Proc. 3rd European conference on Software Maintenance and Reengineering, Amsterdam, Netherlands, 1999.

[11] W. Lam and V. Shankararaman, "Requirements change: a dissection of management issues", Proc 25th EUROMICRO Conference, Milan, Italy, 1999.

[12] W. Lam and V. Shankararaman, "Managing change in software development using a process improvement approach", Proc. 24th Euromicro Conference, vol 2, Vasteras Sweden, 1998.

[13] G. Stark, A. Skillicorn and R. Ameele, "An Examination of the Effects of Requirements Changes on Software Releases", CROSSTALK, The Journal of Defense Software Engineering, 1998.

[14] S. D. P. Harker, K. D. Eason and J. E. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering", Proc. IEEE International Symposium on Requirements Engineering, San Diego, CA, USA, 1993.
[15] I. Sommerville, Software Engineering, Personal Education Ltd, 2007.
[16] N. Nurmuliani, D. Zowghi and S. Powell, "Analysis of requirements volatility during software development lifecycle", Proc. Australian Software Engineering Conference, Melbourne, 2004.
[17] L. Mathiassen, T. Saarinen, T. Tuunanen and M. Rossi , "Managing Requirements Engineering Risks: Analysis and Synthesis of the Literature", Helsinki School of Economics Working Papers W-379, 2004.
[18] D. Perry, A. Porter and L. Votta, "Empirical Studies of Software Engineering: A Roadmap", Proc. 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.
[19] D. I. K. Sjoberg, T. Dyba and M. Jorgensen, "The future of Empirical Methods in Software Engineering Research", Proc. Future of Software Engineering, Minneapolis, MN, 2007.
[20] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap", Proc 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.
[21]N. A. M. Maiden and G. Rugg. "ACRE: selecting methods for requirements acquisition", Software Engineering Journal, 11(3), 1996.
[22] S. Fincher and J. Tenenberg, "Making sense of card sorting data", Expert Systems, 22(3), 2005.
[23] Anonymous, Expert Systems Special Edition on Card Sorting, 22, 3, 2005.
[24] M. Christel and K. Kang, "Issues in Requirements Elicitation", Technical Report No. CMU/SEI-92-TR-012 Software Engineering Institute, 1992.
[25] B. Curtis, H. Krasner and N. Iscoe, "A field study of the software design process for large systems", Communications of the ACM, 31(11), 1988.
[26] H. Barki, S. Rivard and J. Talbot, "Toward an assessment of software development risk", Journal of Managment Information Systems, 10(2), 1993.
[27] B. Boehm, "Requirements that handle IKIWISI, COTS, and rapid change", Computer, 33(7), 2000.
[28] A. Lamsweerde, Requirements Engineering : From System goals to UML models to software specifications, John Wiley & Sons Ltd, 2009.
[29] R. Pressman, Software Engineering. A Practitioner's Approach, McGraw Hill, 2005.
[30] T. Moynihan, "'Requirements-uncertainty': should it be a latent, aggregate or profile construct?" Proc. Australian Software Engineering Conference, Canberra, ACT, Australia, 2000.
[31]T. Moynihan. "How experienced project managers assess risk", IEEE Software, 14(3), 1997.
[32] A. M. Davis and K. V. Nori, "Requirements, Plato's Cave, and Perceptions of Reality", Proc. International Conference on Computer Software and Applications, Beijing, China, 2007.
[33] C. Ebert and J. De Man, "Requirements uncertainty: influencing factors and concrete improvements", Proc. 27th International Conference on Software Engineering, ST Louis, Missouri, USA, 2005.
[34] E. Barry, "Software evolution, volatility and lifecycle maintenance patterns: a longitudinal analysis synopsis", Proc International Conference on Software Maintenance, 2002
[35] K. Bennett, V. Rajlich, "Software Maintenance and Evolution: a roadmap", Proc 22nd International Conference on Software Engineering, ACM Press, New York, 2000.
[36] M. Godfry, D. German, "The past, present, future of software evolution", Frontiers of Software Maintenance, 2008, FoSM 2008.
[37] J. Heales, "Factors Affecting Information System Volatility", Proc. 21st Intl. Conf. Information Systems, Brisbane, Australia, 2000.
[38] M. Lehman, J. Ramil, "Software Evolution", Information Processing Letters, 88(1-2), 2003
[39] D. Perry, "Dimensions of Software Evolution",n Proc. Intl. Conf. Software Maintenance, Victoria, Canada, 1994.
[40] M. Lehman, J. Ramil, "Towards a Theory of Software Evolution – And it's Practical Impact"
[41] E. Swanson, "The dimensions of Maintenance", Proc. 2nd Intl Conf Software Engineering, CA USA, 1976.
[42] N. Chapin, J. Hale, K. Khan, J. Ramil, W. Tan, "Types of Software Evolution and Software Maintenance" Journal of Software Maintenance and Evolution: Research and Practice, 13(1), 2001.
[43] M. Lehman, "Software's future: managing evolution, IEEE Software, 15(1), 1998.
[44] C. Kemerer, S. Slaughter, "An empirical approach to studying software evolution" IEEE Transaction on Software Engineering, 25(4), 1999.
[45] M. Lehman, J. Ramil, P. Wernick, D. Perry, W. Turski, "Metrics and Laws of Software Evolution - The Nineties View", Fourth International Software Metrics Symposium (METRICS'97), 1997.
[46] . Kitchenham, G. Travassos, A. von Mayrhauser, F. Niessink , N. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, H. Yang, "Towards an ontology of software maintenance", Journal of Software Maintenance: Research and Practice 11(6), 1999.
[47] L. Belady, M. Lehman, "A model of large program development", IBM Systems Journal 15(3), 1976.
[48] K. Villela, J. Doerr, A. Gross, "Proactively managing the Evolution of Embedded System Requirements", Proc. 16th International IEEE Conf. Requirements Engineering, Caltunya, 2008.
[49] IEEE std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE, New York, 1991
[50] V. Basili, "Viewing Maintenance as Reuse-Oriented Software Development", IEEE Software 7(1), 1990.
[51] N. Chapin, "Productivity in Software Maintenance", Proc. National Computer Conference, Illinois, 1981.
[52] F. de Almeida, S. de Menezes, A. da Rocha, "Using ontologies to improve knowledge integration in software ebgineering environments", Proc. 2nd World Multiconference on Systemics, Cybernetics and informatics, 1998.
[53] N. Nurmuliani, D. Zowghi and S. P. Williams, "Requirements Volatility and Its Impact on Change Effort: Evidence-based Research in Software Development Projects", Australian Workshop on Requirements Engineering, Adelaide, 2006.
[54]G. Rugg, P. McGeorge, "The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts", Expert Systems, 22(3), 2005.
[55] C. Paul, "A modified Delphi approach to a new card sorting methodology", Journal of Usability Studies, 4(1), 2008.

Appendix

ID	Development Trigger Construct	Source	Removed	Applicable to Maintenance
45	Use of Prototype	[27] , [14]	Technique, covered by 21, 44	
68	New Stakeholder (role)	[27]		Y
96	Customer Company Reorganization	[28], [16], [14], [27], [29]		Y
51	New solution Tools/technology	[27], [28], [25], [14], [8]		Y
54	Change to government policy or regulation	[28], [14],[8]		Y
20	Participatory Learning	[14],		Y
28	Local Customization	{14]	New Stakeholder	
50	Customer migration to new solution	[14]	Type of requirement	
39	Customer need change	[24], [25], [16], [29], [29]	Too woolly, covered by 47, 96, 54, 82, 21, 45, 42, 90	
44	Developers Increased Understanding of problem	[25], [16],[17]		Y
56	Scope Reduction	[9][16]	By-product of 88, 66	
34	Changes to packaging/licensing/branding	[9]	Covered by 61	
65	Solution Elegance (Design Improvement)	[25], [9]		Y
67	Resolution of mis-communication	[9]		
49	Testability	[9]	Type of Requirement	
82	Business Process change (continuous improvement)	[15], [8]		Y
42	Response to competitor	[25], [8]		Y
16	Functionality Enhancement	[16]	Covered by 78, 65	
11	Defect Fixing	[16]	Doesn't result in requirement change	
69	Redundant Functionality	[16]	Covered by 66, 44, 20, 90, 67, 23, 51, 65, 21	
8	Missing Requirement Identified	[16]	Not a reason/cause/source	
86	Clarification of Requirement	[16]	Covered by 67, 23	
21	Increased customer understanding	[28], [15], [8], [17]		
72	New Class of User	[28]	Result of other changes, covered by 82, 68	
74	New Usage Condition	[28]	Covered by 78, 85	
15	New way of doing things	[28]	Covered by 82, 96	
77	Correction to Requirements specification	[28]	Covered by 23, 67	
78	New Opportunity	[28]		Y
1	Change in the use of the information	[17]	Covered by 82	
88	Cost or schedule overrun	[29], [28]		
49	Testability	[16]	Type of requirement	
85	Change to Customer's hardware/software	[9]		Y
58	System Usage (after installation, not prototype)	[27], [15], [14]	Out of scope of project development	
90	Changes to Market Demands	[8], [14], [16], [29]		Y
62	Resolution of Conflicting Requirement	[16]	Covered by 83	
55	New Functional Feature	[28]	New Requirement	
3	Improved Quality Feature	[28]	Change to requirement for another reason	
14	Result of Change in political climate (needs of particular group emphasized)	[24], [8], [25]		Y
93	Change to customer's environment	[15]	Covered by 96, 68, 85, 47, 66	
18	Changes in Underlying technologies	[25]	Covered by 85, 51	
83	Incorrect Requirement Identified	[16]		Y
23	Resolution of Misunderstanding	[25]		Y
92	First or re-engagement of user representative	Added		
66	Change to business Case (Return on Investment, Total cost of Ownership	Added		Y
61	Customer Organization Strategic Change(New Marketing/Sales direction, change to organization goals)	Added		Y
12	Change of Stakeholder Representative	Added		Y
4	Understanding Technical Solution	Added		

	Development Uncertainty Construct	Source(s)	Removed	Applicable to Maintenance
31	Analyst skill or experience	[4], [24], [30], [8]		
95	Development team knowledge of business area	[24], [31]		
79	Quality of Analysis techniques employed (workshops, interviews, modeling etc)	[4], [8], [15], [29], [28]		
59	Project Size	[24], [17], [15]		Y
30	Novelty of product (business novelty)	[31], [8]		Y
41	Logical complexity of problem	[17], [31], [24]		
19	Availability of communication with customer	[17], [8]		Y – added word 'stakeholder'
22	Involved customer's knowledge/understanding/clarity of requirements	[24], [32], [31], [8]		
64	Quality of Communication between analyst and customer	[24], [17], [32], [8]		
33	Involved customers experience with working alongside IT to produce solutions	[31]		
9	Diverse User Community	[31], [15], [8]	Summary of 29, 27, 40, 41, 2, 60	
32	Incompatibility between requirements	[28]		Y
24	Lack of well understood model of utilizing system	[17]	unclear	
55	Lack of well-understood model of the utilizing system	[17]	unclear	
6	Lack of structure for activity or decision being supported	[17]	Covered by 22	
52	Stability of Customers Business Environment	[24]		Y
76	COTS usage	[28], [27]		Y
2	All stakeholders identified	[27], [33]		Y
29	All Stakeholders involved	[24], [33], [27], [8]		Y
40	Clarity/unity of shared product vision	[30], [14], [33], [8]		Y
27	Synergy of stakeholder agenda	[28], [31], [14]		Y
43	Unknown Customer Project Dependencies	[33]		Y
46	Market Stability	[25], [32], [14]		Y
13	Differing Customer Needs	[25]		Y
38	Type of user doing specification (incorrect user involved)	[8][30] [17]		
89	Change in the utilizing system	[17]	unclear	
80	Low Staff morale	[4]		
10	Large number of users	[17]	Not a risk if correct user involved - 38	Y
87	Level of participation of users in specification	[17]	Covered by 19, 64	
81	Lack of user experience of utilizing system	[17]	Covered by 22	
63	Degree of Change to customers workflow	[31], [8]		Y
48	Quality of Requirements specification	Added		
71	Technical Uncertainty of Solution	Added		
84	Technical Complexity of Solution	Added		
53	Quality of Development team Communication	Added		Y – removed word 'development'
73	Age of Requirements(elapsed time since completion of requirements documentation)	Added		
60	Insufficient Sample of User Representatives	Added		
35	Development team (PM and analyst) stability	Added		

ID	Maintenance Trigger Construct	Source	Removed	Equivalent Development Construct
161	Use Of Case Tools	[34]	Technique	
141	Maintenance activities	[34]	technique	
146	Changes to deployment Environment	[36]		
153	New Opportunity	[36], [38] , [40]	Covered by 137	
128	Domain changes due to system Use	[36], [38]		
125	Evolution of surrounding environment	[36]	Unclear	
113	Deferred Requirements during development	[37]		
109	System Usage	[38] , [39] , [14]	Activity	
142	Changing Customer Needs	[38]	Unclear	Variation of development constructs
104	Changing Technology (for solution)	[38] , [40], [42]		New Tools/Technology
116	Increased customer understanding	[39]		Increased Customer Understanding
151	New technological Methods	[39]	Covered by 104	
112	New Tools	[39]	Covered by 104	
119	Organisation Changes	[39]		Company re-organisation
138	Change to Operational Domain	[40]	Covered by 146,104,151	
132	Increased User Sophistication	[40]	Covered by 116	
136	Response to competition	[40]		Response to Competitor
165	Ambition	[40]	Covered by 153, 137	
162	Business Process Improvement	[40]		Business Process Change
101	Migration to other technology Environments	[14]	Covered by 146	
144	Function added, replaced, deleted	[42]	Not a cause	
103	Adapt to new technological environment	[42]	Covered by 146	
123	Alter system performance	[42]		Design improvement/.solution elegance or response to competitor/new opportunity
147	Alter Maintainability	[42]		Design Improvement/solution elegance
137	Response to Gap in Market	Added		Response to gap in market (added)

ID	Maintenance Uncertainty Construct	Source	Removed	Equivalent Development Construct
166	Business Size	[34]	Represented by 157,163	
121	Maintenance team Instability	[34]	Not a cause of req change	
135	System Complexity	[34]	Increase defects but not req change. Represented by 157, 163	
143	In-house software	[34]	Increase change capability but not cause	
158	Maintenance Team Knowledge	[35]	Not causing req change	
117	Cohesive Architecture	[35]	Effects defects but not req change	
148	Presence of Competitor	[36]		Presence of Competitor (added)
118	Market Environment	[36]		Market Stability
154	Semantic Relativism	[37]		Semantic Relativism (added)
115	System Age	[37]		
102	Period	[37]	Unknown	
167	Economic Climate	[38]	Effects ability for change but not cause	
164	COTS usage	Added		COTS Usage
157	No of Interfaces	Added		Project Size
139	Diversity of User Needs	Added		Differing Customer Needs
156	Stakeholder Agreement	Added		Synergy of Stakeholder agenda
163	No of Functions	Added		Project Size
221	Quality Control during development	Added		Quality Control during development (added)
102	Function Usage	Added		Function Usage (added)
126	No. Of Users	Added		No Of Users (added)

Equipping Software Engineering Apprentices with a Repertoire of Practices

Vincent Ribaud, Philippe Saliou
Université de Bretagne Occidentale, LISyC - EA 3883
Université Européenne de Bretagne
Brest, France
{Vincent.Ribaud, Philippe.Saliou}@univ-brest.fr

Abstract—**Argyris and Schön distinguish espoused theories - those which people speak about – from theory-in-use - those which can be inferred from action. In small software teams, developing reflective thinking about action is a vital necessity in coping with change. We address these issues in a Masters of Software Engineering, performed with an alternation between university and industry. University periods are dedicated to a long-term project performed in a reflective practicum. It aims to develop a repertoire of practices which helps young engineers deal with the 'messiness' of situations. Such a practicum provides students, working in groups, with the possibility of reflecting on action. We propose using the Course-of-Action framework to record observable aspects of the actor's activity into semantic wikis. Two hypotheses are discussed (1) self-analysis and self-assessment help to reveal theories-in-use; (2) the Course-of-Action observatory helps maintain awareness of the repertoire. A case study of a 6-apprentice team illustrates the observatory use and the reconstruction of apprentices' activity. Primary conclusions are that self-observation and self-analysis of a software engineer's activity help raise awareness of the initial structure of the repertoire. We are however unable to conclude that it helps reveal their theory-in-use (what governs an engineer's behaviour) - usually tacit structures.**

Keywords-component; reflective practitioner, software engineering processes, Course-of-Action, semantic wiki.

I. INTRODUCTION

This paper is an extended and enhanced version of a paper presented at the ICCGI 2009 conference [1].

Small organizations – and small software teams especially – need to constantly adapt their task force to the products or services to be delivered. The software process community shares a tacit axiom that improving software processes automatically improves software products and contributes to the project success. Many efforts have been made to extensively define a set of processes and build assessment methods intended to verify to what extent defined processes are performed.

Yet D. Schön [2] argued that experienced professionals deal with the 'messiness' of practice not by consulting the research knowledge base, but by engaging in 'reflection-in-action': experiencing surprise in a new situation and responding to surprise through a kind of improvisation. To educate the reflective practitioner, Schön recommended looking at traditions of education for artistry – in art studios, or in conservatories of music and dance. Schön qualified these students as learning by doing in a reflective practicum. This analogy was used to provide a suitable educational environment for software design at CMU [3] or at MIT [4].

The notion of repertoire is very important in Schön's approach. Practitioners build up a collection of ideas, examples, situations and actions. "*A practitioner's repertoire includes the whole of his experience insofar as it is accessible to him for understanding and action*" [5, p.138].

This hypothesis – coupled with the observation that students (and young engineers) are experiential, tending toward learning by doing rather than listening – led us to focus on the goal of providing software engineering graduates with a non-empty repertoire of practices, together with an operational knowledge of software processes, activities and tasks. In 2002, we built an education system called 'Software Engineering by Immersion' entirely based on performing complete development cycles of a software project, and accomplished in three iterations. This 3-iterations system can be summed up with the sentence: 'A first turn to learn by doing, a second turn to do autonomously what has been learned, and a last turn to work effectively in a business. A Process Reference Model - greatly simplified from the ISO/IEC 12207:1995 standard and its amendments [6] - was used as the initial structure of the repertoire. As realistic working situations were experienced, students were provided with progressive filling of their repertoires.

In 2007, local employers in Brest requested employees in 'sandwich' (or work placement) conditions, and we adapted the 'Software Engineering by Immersion' programme to run as a work placement course. In such a programme, some of the educational objectives and related assessments are devoted to periods in industry. The second and third iterations were good candidates to assign to industrial periods, and first iteration (at the university) and second iteration (in the industry) were organised into alternating 2-week periods. We do however face the problems of relating the university-based and industry-based elements of the student's experience and avoiding a situation in which learners are required to climb two ladders simultaneously.

We decided to redesign the repertoire (including its construction and 'filling') in order to - as far as possible – meet the twin challenges of learning and producing within a small software project. Our current proposition is to use two theories of action, the former from Argyris and Schön [7] about theory-in-use and espoused theory and the latter - the Course-of-Action framework pioneered by Theureau and Pinsky [8]. The main idea is to provide young engineers and

small projects with an observatory of their individual and collective activity (by observable, we mean what is presentable, accountable and commentable) and to instigate the reconstruction of a high-level view of the global Course-of-Action from small and individual units of action.

The nature of collected data is very different and subject to change, as is the semantic of the relationships between data. This addresses technical challenges related to content management. We choose to use semantic wikis as a lightweight authoring platform. As Maxwell [9, p.199] outlines "*Our experience with and reflection on using wiki as a platform suggests that there is much to be gained from an approach which builds up from simple foundations rather than attempting to customize already-complex architecture*".

Section 2 overviews theories of this introduction, their application to software engineering and some related work. In Section 3, we present courses-of-action for software apprentices. Observing the course of apprentices' projects is discussed in Section 4. In Section 5, we present some data excerpts of a case study. We conclude the paper with a discussion and perspectives.

II. RESEARCH ISSUES AND RELATED WORK

We will present Argyris and Schön's theories of action as well as certain elements of the Course-of-Action framework. We will present two research hypotheses, and related work.

A. Espoused theories and theories-in-use

A starting point of Argyris and Schön's [7] theory (see Figure 1) is that people design action to achieve intended consequences, monitoring themselves in order to learn whether their actions are effective. They made a distinction between two contrasting theories of action: *theories-in-use* and *espoused theories*. "*When someone is asked how he would behave under certain circumstances, the answer he usually gives is his espoused theory of action for that situation. This is the theory of action to which he gives allegiance, and which, upon request, he communicates to others. However, the theory that actually governs his actions is his theory-in-use*" [7, pp.6-7].

Our first observation is that, in the software engineering field, lifecycle processes standards such as the 12207 [10] and process assessment standards such as 15504 [11] or CMMI may constitute the espoused theory, since it is what engineers claim to follow. But what engineers do (and this action is designed - it does not 'just happen') may reveal a different theory-in-use. A young engineer is rarely aware of either their theory-in-use or of any inconsistency - although an experienced engineer may be.

Theories-in-use can be made explicit by reflecting on action [7]. According to Schön, reflective thought takes place in a reflective practicum. Schön advocated traditions of education for artistry as exemplar through their reflective practicum. "*[…] its main features are these. It's a situation in which people learn by doing, […] where they learn by doing in a practicum which is really a virtual world. A virtual world in the sense that it represents the world of practice, but is not the world of practice […] in that world, students can run experiments cheaply and without great danger […] in interaction with someone who is in the role of coach*" [12].

A reflective practicum is intended to run experiments and develop reflection-on-action. In our practicum, we use organized processes to drive project and competencies building. Parallel to the engineering activities required by the project, apprentices are regularly required to self-analyze and self-assess their engineering practices. Our first hypothesis (H1) is that self-analysis and self-assessment helps an apprentice to reveal their theory-in-use.

Previous and related work. The studio is the central training method in architecture schools and this analogy was used to provide a suitable educational environment for software [3] [4]. Our system is very close to Tomayko's work, and most of his observations apply to our system: "*The use of a well-established development process, a matrix organization, and one-to-one mentoring give the highest return on investment*" [3, p.119].

Hazzan and Tomayko present in [13] a course intended to develop reflective thinking about the education of software engineers - but theories of action are not evoked.

Halloran [14] investigates the relationship between a software process assessment and improvement model and organizational learning. The paper points out the difference between 'engineer's espoused theory' and their 'theory in use' but does not develop this idea, focusing instead on the use of organizational learning to promote a proactive approach to continuous improvement and learning procedures.

Models of theory-in-use

Argyris and Schön argued that, even though espoused theories vary widely, theories-in-use do not. They labelled the most prevalent theory-in-use *Model I* and argued that this model inhibits learning. *Model II* favours it. This model looks to three elements. *Governing variables* are values that actors seek to satisfy [1]. Each governing variable can be thought of as a continuum with a preferred range (e.g. not too anxious, yet not too indifferent) that people are trying to keep in these acceptable limits. *Actions strategies* are sequences of moves used by actors in particular situations to satisfy governing variables [1], there are the moves and plans used by people to keep the governing variables in the preferred range (e.g. to use physical exercise to eliminate stress). Consequences happen as results of action. Consequences can be intended – those that the actor believes will result from the action and will satisfy governing variables (e.g. feeling better after sporting effort). Consequences can be unintended but they are designed because they depend on the theories-in-use of recipients as well as those of actors.

Single and double-loop learning

When the consequences of an action strategy are as the actor wanted, then the theory-in-use of that person is confirmed. If there is a mismatch between intention and outcomes, consequences are unintended. Argyris defines learning as the detection and correction of error. The first response to error is to search another action strategy (Model I). "*Single-loop learning occurs when errors are corrected without altering the underlying governing variables*" [2, p. 206]. An alternative is to question governing variables themselves (Model II), to subject them to critical scrutiny (e.g. to emphasize open inquiry into the anxiety rather than trying to suppress it). "*Double-loop learning occurs when errors are corrected by changing the governing variables and then the actions*" [2, p. 206]. Argyris and Schön argued that many people espouse double-loop learning, but are unable to produce it, and are unaware of it.

References

[1] C. Argyris, R. Putnam, and D. McLain Smith, "Action Science, Concepts, methods, and skills for research and intervention", San Francisco: Jossey-Bass, 1985.

[2] C. Argyris, "Double-Loop Learning, Teaching and Research", Learning & Education, Vol. 1 (2), Dec. 2002, pp. 206-219.

Figure 1. Theory of Action by Chris Argyris and Donald Schön.

B. Building their own repertoire

The Course-of-Action theory, pioneered by Theureau and Pinsky [8], provides a framework for analysis of the collective organization of the multiple courses of action in a complex, autonomous and open system. A 'Course of Action' is: "*what, in the observable activity of an agent in a defined state, actively engaged in a physically and socially defined environment and belonging to a defined culture, is pre-reflexive or again significant to this agent, i.e. presentable, accountable and commentable by them at any time during its happening to an observer-interlocutor in favourable conditions*" [15, p.7]. The course of action can be described from two complementary perspectives: from the point of view of its global dynamics - characterizing the units of the course of action and the relations of sequencing and embedding between these units, or from the point of view of its local dynamics, characterizing the underlying structure of the elementary units [15]. Given that we seek to establish a fairly high-level model of actions, we focus on the global point of view because it emphasizes the articulation of work situations and their co-ordination, and is better suited to process-level analysis.

Argyris and Schön suggested that each member of an organization constructs his or her own representation or image of the theory-in-use of the whole [16, p.16]. What is intended is to connect the individual world of the practitioner up with the collective world of an organization. But, prior to this discussion, we need to understand how we perceive our internal structure. The notion of repertoire is a key aspect of Schön's reflection in and on action. Practitioners build up a collection of ideas, examples, situations and actions. "*When a practitioner makes sense of a situation he perceives to be unique, he [she] sees it as something already present in his [her] repertoire. […] It is, rather, to see the unfamiliar, unique situation as both similar to and different from the familiar one, without at first being able to say similar or different with respect to what. The familiar situation functions as a precedent, or a metaphor, or an exemplar for the unfamiliar one*" [5, p.138].

A coach may help both discover the existence of this repertoire, and fill it, with the assistance of reflective thought. Coaches often answer questions with questions, in most cases, simply rephrasing the question. Our proposal is that small projects should be provided with a device which will act as a mirror for their observable activity, and that privileged moments of self-observation in front of the mirror (without adding too much extra work) should be seamlessly integrated in the course of the project. Our second hypothesis (H2) is that the Course-of-Action observatory helps maintain awareness of the repertoire, facilitating self-assessment and self-analysis.

Previous and related work. Hazzan debates the reflective practitioner perspective in software engineering education and the studio as a teaching method [17], but does not address the subject of the practitioner's repertoire.

The 'Course-of-Action' research framework consists of several empirical and technological research programs [15] in various domains such as work analysis [18] or traffic control [19]. We are not aware of any uses of the Course-of-Action framework in the software field.

III. SOFTWARE APPRENTICES' COURSES-OF-ACTION

We will be monitoring the 'Software Engineering by Immersion' Masters programme, and we will present the Course-of-Action observatory and its application to a software project.

A. The 'Software Engineering by Immersion' Masters Programme

1) Structural aspects of our programme

Our Masters Programme in Information Technology and Software Engineering is a 2-year programme, accessible to Bachelor graduates in Computing or 'back to school' software practitioners. For students enrolled in the Software Engineering by Immersion specialization, securing a 'professionalization contract' is a compulsory requirement. During this 12-month contract, the work placement student is a full-time employee, although also attending university for certain periods. Strictly-speaking in France, 'apprenticeship learning' and 'apprentice' are terms reserved for a longer work placement system, but the sake of clarity, we use the term 'apprentice' in this paper.

Competition for this type of contract is performed during the first 7-month intensive courses. The following 4-months are dedicated to an internship period. For the last year, periods at university have to fit into alternating 2-week periods. The year is divided into two periods, the former (from September to mid-May) with movement between university and company, and the latter (from mid-May to August) with a full-time period at the company.

2) Pedagogical objectives and organization

Of the 43 processes of ISO/IEC 12207:2008 [10], we concentrate on the 19 that are related to the software development cycle, which we have reorganized into 3 groups:

- in the *Software Project Management* Process Group: 6.3.1 Project Planning - 6.3.2 Project Assessment and Control, 7.2.2 Software Configuration Management, 7.2.3 Software Quality Assurance;
- in the *Software Development Engineering* Process Group: 6.4.1 Stakeholder Requirements Definition, 6.4.3 System Architectural Design, 6.4.4 Implementation Process replaced by 7.1.1 Software Implementation Process (and its 6 sub-processes: Requirement Analysis, Architectural Design, Detailed Design, Construction, Integration, Qualification), 7.2.4 and 7.2.5 Software Verification & Validation;
- in the *Software Development Support* Process Group: 6.2.1 Life Cycle Model Management, 6.2.2 Infrastructure Management, 6.4.7 Software Installation – 6.4.8 Software Acceptance Support, 7.2.1 Software Documentation Management.

These 19 processes are renamed (and some are also merged) to give a breakdown of apprenticeships into 3 software engineering process groups subdivided into 13 software engineering processes, together with a set of apprenticeship scenes (roughly associated with software

engineering activities) which provide the learning environment and define tasks. This hierarchical group process/process/scenes model, adapted from the ISO/IEC 12207, is given in Tables I and III and is used as a reference framework for the learning objectives.

From the university point of view, this division is the reference framework, in a diploma-awarding perspective. Group processes are course categories, processes are courses and scenes are sessions.

TABLE I. PROCESS BREAKDOWN

Process Group	Process	12207:2008 Related Processes
Software Project Management	Project management	6.3.1, 6.3.2
	Quality insurance	7.2.3
	Software configuration management	7.2.2
Software Development Engineering	Requirements capture	6.4.1
	Software analysis	7.1.2
	Technical architecture	6.4.3
	Software design	7.1.3, 7.1.4
	Software construction	7.1.5, 7.1.6
	Integration and validation	7.1.7, 7.2.4, 7.2.5
Software Development Support	Technical support	6.2.2
	Methods and tools support	6.2.1
	Documentation	7.2.1
	Installation and deployment	6.4.7, 6.4.8

The main feature of the university periods is to learn software engineering by doing, without any computing course but with a long-term project as the foundation of all apprenticeships. Alternating employees are attending university over 9 periods of 2 consecutive weeks, and work in teams of 6 apprentices to build a complete information system.

The rhythm is based on the lifecycle of a project, organized into stages. Each stage was arbitrarily sized to 2 weeks, due to the constraints of alternation. The cycle is: Stage 0: Warm-up; Stage 1: Project set-up; Stage 2: Requirement capture; Stage 3: Requirement analysis; Stage 4: Design; Stage 5: Software construction; Stage 6: Software construction; Stage 7: Integration and Verification; Stage 8: Qualification and Deployment.

3) Competency Reference Model

While apprentices are currently learning by doing software processes, process assessment will not measure a capability level but (in the best case scenario) a learning capability level. Because apprentices are building competencies, and because some reflective learning is required, we choose to promote self-assessment of personal abilities. In 2006, we set down the abilities (or competencies: "*the ability of a person to act in a pertinent way in a given situation in order to achieve specific purposes*" [20]) that scenes are intended to develop. We tried to answer the questions 'What is the student able to do, once the scene is performed? What are the related knowledge topics?' This analysis gave us a set of abilities for each process (see examples in Table II).

So we kept the 2-level breakdown of our reference framework - the first level being called competency areas (corresponding to process groups) and the second level competency families (corresponding to processes), and we positioned abilities and transversal competencies within these areas. Table II shows abilities and knowledge related to certain representative processes for each process group.

TABLE II. EXAMPLES OF COMPETENCY FAMILIES

Abilities and Skills	Knowledge Topics
Project Management	
• To use an ISO 9001 development baseline • To apply a Project Plan, updating it if necessary • Planning and project progress	* Software life-cycle model * Estimation and follow-up of development of components * Traceability and conformity * Project Plan
Software Requirements Capture	
To mobilize specification methods and tools in a real project: • within an ISO 9001-style baseline • in relation to requirements traceability • to produce a Software Requirement Specification	* Software Requirements Fundamentals: definition, functional and non-functional, quantification. * Requirements capture techniques: interviews, client meeting, statement of work, response to solicitation * Procedures, methods and tools for requirements specification. * Use cases.
Software Design	
• To use design methods and tools (in relation with requirements) to produce design documents: system and software architecture and detailed design • To implement methods and modelling tools of various aspects of a system (architecture and decomposition software, data structure) • To implement J2EE development and technology of associated framework • To implement DBMS concepts, techniques and tools	* Software Design Fundamentals: concepts and principles, design role in a development cycle, top-level and detailed design * Software decomposition configuration item, software component, software unit * Software architecture through different views: conceptual, dynamic, physical, data. * UML diagrams to describe static and dynamic views * Object-oriented design
Methods and tools support	
• To know Software Engineering methods and techniques for the software life cycle • To install, adapt, integrate and maintain software tools • To assist engineers in software deployment • To perform a consulting mission, alone or in a group	* Use case models and formats. * Analysis: patterns and model transformation *Design:architectural prototype, generic design *Configuration management: tools and guides * CASE tools

The complete breakdown (3 areas, 13 families, 48 abilities and 11 transversal competencies) is called the competency reference model for our immersion system [21].

B. The Observatory of Apprentices' Courses-of-action

1) The Course-of-Action observatory

A short definition of a Course-of-Action is "*the activity of one (or several) specific actor(s), engaged in a specific situation, belonging to a specific culture, which is significant for the latter, in other words, that can be related or commented by (or them) at any moment*" [18, p. 2].

The Course-of-Action analysis is based on an observatory which includes continuous observations of the behaviour of action and communication in a work situation as well as different traces of other elements such as interpretations, feelings, and judgments [18]. Although the data produced by these observations only gives access to the surface of interactions, it suffices for understanding the structural coupling between an agent and his or her situation.

The Course-of-Action framework proposes data collection methods which include: observing and recording actors' behaviour methods; methods to keep track of behavioural patterns; and methods of provoked and situated verbalization from actors.

Self-confrontation is a prominent activity in terms of documenting the Course-of-Action. This takes the form of collecting verbal data whilst the activity is actually being carried out and/or in a self-confrontation situation (e.g. in the case of driving, the driver watches a film of their journey (which is systematically recorded) and comments on it to clarify their own actions and events [15]).

Other kinds of verbalization, made by agents during activity analysis (called second degree self-confrontation verbalizations - to emphasize the fact that they are situated in the continuity of self-confrontation itself) are also implemented. Here the agents are placed in the position of observers and analysts, and their verbalizations, whilst not data, do nonetheless constitute their contributions to the analysis of their activity [15].

2) Link with field studies

From the 'data collection techniques' point of view, Lethbridge, Sim, and Singer [22] provide a useful taxonomy. They classify techniques according to the degree of human contact required. We use several 'observational first degree techniques' [22] (with direct access to young engineers), including diaries, think-aloud protocols, observation, and participant observation. Because we focus on self-interpretation by the actors, most recording is done manually by the actors themselves. From the taxonomy point of view, it may appear as a 'second degree technique' [22] (with indirect access to young engineers) because it does not require direct contact between participant and researcher.

Representative artefacts of the job are the outputs of software activities and tasks. The analysis of these artefacts falls in a 'third degree technique' [22] (without access to young engineers).

3) What can be collected in the course of projects ?

Recall the definition of the Course-of-Action in §III.B.1: what, in the observable activity of an agent […] is pre-reflexive or significant to this agent, i.e. (i) presentable, (ii) accountable and (iii) commentable by them at any time whilst it is happening […].

We have three types of observation: (i) presentation, (ii) accounting, (iii) comment. Software workers do not achieve complex technical gestures, or do not have to progress through a detailed procedure. So (i) presentations to an observer are quite difficult to reproduce, and the presentable artefacts that are most notable and representative of the job are the outputs of software activities and tasks.

Verbalization is widely used by the coach within the learning process to scaffold the apprentice's activity: when students ask or when tutors consider it to be necessary, a dialogue between apprentice and tutor about what, why and how the apprentice is doing helps them to carry out the activity. Recording this dialogue would be too complicated; furthermore, it would probably compromise - and possibly even destroy - this learning process.

We therefore focus on accounting and comments. Accounting will replace recordings of engineer behaviour. Products and documentary resources are the main objects of presentation, since they describe the activity's inputs and outputs. The 'historical' context of use of (i) resources and product production must also be recorded. This can be described in terms of events and processes, involving occurrences of agents (people) and artefacts (products and resources) meeting in space (in case of distributed collaboration) and time. In the first instance, we consider the individual courses of action of the various participants. At the next stage, we look at collective action involving parts of several individual courses of action taking place synchronically or sequentially. We need to divide individual Course-of-Actions into smaller units, which we call Performed Activity. Each event of interest must be (ii) individually accounted for in an instance of Performed Activity in relation with the apprentice and artefacts involved. This provides a kind of project diary or journal, and is performed in a wiki by each apprentice as the project goes along.

Provoked verbalizations are replaced with self-confrontation interviews as a way of documenting the constraints and effects of the segment of the actor's activity that is personally experienced. Even the smallest unit of collective Course-of-Action is called a Course-of-Action Unit, which organizes several individual Performed Activities. At the end of each 2-week university period, apprentices have to write a short report (individually, but also collectively if they worked on a group task) about what happened during the period (this is called a 'work diary' in the taxonomy of [22]). Apprentices may complete Performed Activity instances previously created, and must create Course-of-Action Unit instances for activities involving several individual Performed Activities.

For the industrial periods, accounting is performed in a different way. As detailed in Section IV.B.5, young engineers must perform a complete self-assessment, 4 times per year, regarding the competency reference model described in §II.A.3 - which is acting as an ability model for their job as an engineer. In support of these periodic assessments, they have to record events of interest in a portfolio, associating events with significant artefacts they may have used or produced.

With very few exceptions, we observed that information about academic and industrial periods are written in a descriptive style (what they do with linked artefacts, when and where) but gave little or no indication why and for what reason they did it. So, we can conclude that these reports are (ii) accounts and not (iii) comments.

4) What can be commented ?

We need a second-level of reporting intended to encourage comments and reflection-on-action. Final reports with in-class presentation could have been used, but we chose not do this in order to avoid introducing bias, since they are assessed with a mark. We found it more useful to trigger intermediary reports without any assessment. Of course, students will re-use analysis, writings and oral presentation in their final reports, as well as for required self-assessments (and this may provide extra motivation to perform sound intermediary reports) but we minimize assessment bias.

Among the self-confrontation methods used in the Course-of-Action framework, one is the so-called second level self-confrontation interview. It is performed after the self-confrontation interview proper. Its procedure is radically different, because its aim is not to collect empirical data about the actor's experience at instant t, but to develop co-operation in the analysis of the activity between the researcher and the actor [19]. We borrow this practice for reporting on industrial and academic periods.

Every two months (corresponding to two industry periods of 2 weeks each), the academic period begins with a half-day during which each apprentice (12 in all) presents an intermediary report of their activities at work. Writing up a meeting report is assigned to two students, based on the individual reports provided by each apprentice.

During academic periods where there is no industrial reporting, apprentices must perform a first-level analysis on the state of the software processes of their academic project. Each apprentice has to work on two or three processes (13 are used in all), building an intermediary process element called Step-of-Action based on historical Course-of-Action Units related to a given process. This analysis is intended to produce a reconstruction of the global dynamic in terms of smaller units and the sequencing and embedding relationships between these units.

IV. OBSERVING THE COURSE OF APPRENTICES' PROJECTS

We will survey the models used in the 'Software Engineering by Immersion' programme. We will present an enacted project that will be used as a case study.

A. Process models

1) Prescribed work

Leplat [23] has identified a difference between prescribed task and effective task. The prescribed task is a task that a designer or an organizer wishes an operator to perform. What he/she really achieves is the effective task.

It is a hard task to define things to do in a software projects, hence to provide a structured description of the prescribed tasks. Several standards were written with this goal. In our opinion, the process dimension of the ISO/IEC standard 15504:2004 [11] provides a complete view of the prescribed work to be done in a software project.

ISO 15504 separates process and capability levels in two dimensions. In the process dimension, individual processes are described in terms of Process Title, Process Purpose, and Process Outcomes as defined in ISO/IEC 12207 (where each life cycle process is also divided into a set of activities; each activity is further divided into a set of tasks [10]). In addition, the process dimension provides: a) a set of base practices for the process, providing a definition of the tasks and activities needed to accomplish the process purpose and fulfil the process outcomes; b) a number of input and output work products related to one or more of its outcomes; and c) characteristics associated with each work product [11]. The capability dimension consists of six capability levels (Level 0 reflects an incomplete process) and the process capability indicators for nine process attributes for levels 1 to 5. A process attribute is "*a measurable characteristic of process capability applicable to any process*" [11, p. 4]. Figure 2 represents the two dimensions and a performance of process assessment.

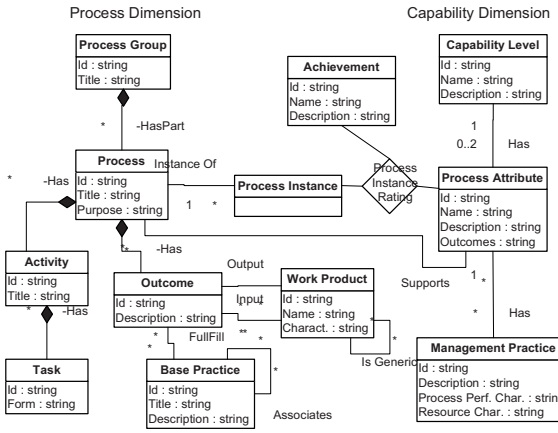


Figure 2. 12207 and 15504 Reference Models. Performing a process assessment yields a rating for each process attribute. A rating is a judgement of the degree of achievment (None, Partially, Largely, Fully) of the process attribute for the instance of the assessed process.

2) Work scenes

The 15504-5 standard provides software engineers with an exemplar model of a software project. Unfortunately, such an exemplar model is necessary but not sufficient for learning purposes. In our system, we have to organize the apprentices' activity into small units of work called an apprenticeship/production scene.

The apprenticeship/production scene is the reference context in which a part of the play happens: the scene aims for a unity of place, time and action; the scene is at once a situation in which people learn and do; a scenario of actions; a role distribution, and an area mobilizing resources and means. The different components of a scene, along with their articulation are depicted on a card. The card structure is standardized (see an example in Figure 3).

The main elements of a card are the process group / process (here development / design) tied up with the work; the role to play (here, designer or architect) with team-mates' assignment; the work description (here, the detailed design); the products (deliverables) to deliver (here, a Software Design Document, SDD); the supplied pedagogical resources (here, a writing guide, real SDD samples and an analysis and design course); workload and lead-time information.

No. 24	**Date**:	**Origin**:	**Roles assignment**	
Project:	**SCENE CARD**		Designer Architect	Solenn Arnaldi Aude Genoud
Process group: SW development engineering **Process**: Software design				**Name**: Detailed design

WORK DESCRIPTION

The goal of the software design activity is to establish a software design that effectively accommodates the software requirements. During card No. 20 'Preliminary design', software requirements have been transformed into software architecture and a top-level design for the external and internal interfaces. Now, the purpose of the 'Detailed design' is to:

- Transform the top-level design into a detailed design for each software component. Components are refined into lower software units that can be coded, compiled, and tested.
- Establish traceability between the software requirements and the software designs.

…

The expected result will be materialized with a Software Design Document (SDD) in accordance with the project baseline. This document describes the position of each software unit in the software architecture and the functional, performance, and quality characteristics which each must address. The sections related to the DBMS will be on the responsibility of the card No. 25 'Database Server analysis, design and generation'.

…

Teaching resources can be helpful in writing the SDD:

- Simplified writing guide for the software design document (TEMPO-IGQ348).
- SDD Examples: Techsas and Techdis.
- Object-Oriented Analysis and Design Using UML - Ch. 10 '

…

Products	**V.**	**Milestone**
Software Design Document (SDD)	A	8-3-2009

Start date	**End date**	**Workload**	
7-30-2009	8-3-2009	5	5

Figure 3. Example of a scene card.

3) Exemplar scenes

Our fundamental problem is to prescribe the content of apprenticeship scenes, their objectives and their outcomes and to link scenes with SE processes (and their outcomes). Our modest proposition is to build scenes from previous scenes description, called exemplar scenes. Rather than being provided with an abstract definition of the prescribed situation and its prescribed tasks, the tutor has to design scenes from previous exemplar scenes and his/her own previous scenes (from past projects). Table III shows the complete breakdown for some processes.

Although an intermediate level between process and task may exist (12207 activity), the hypothesis is made that it complicates the model - and that hypothetical activities are only presented so as to facilitate the link with the 12207.

4) Competency Assessment Model

Regarding the understanding of software processes that students are building, we were faced with a crucial issue. In the previous system (without work placement), students learned software processes by doing during the first iteration and reproduced these processes during the second one. Thus, links were easy to establish and a practical understanding of software processes occurred. Now, first iteration (focused on learning activities) and second iteration (focused on productive activities) are performed on different projects. The former is an apprenticeship project driven by the university and the latter is an industrial project driven by the companies with which students are placed. We need an assessment framework that is common to both projects and which allows apprentices to relate and cumulate experiences.

TABLE III. PROCESS BREAKDOWN AND EXEMPLAR SCENES

12207	**Process**	**Hypothetical activity**	**Scene**
Group Process 'Software Project Management'			
6.3.1, 6.3.2	Project Manage-ment	• Project tailoring • Project planning • Project progress	Response to solicitation Project plan Project plan review Weekly progress meeting Project monitoring and control
…			
Group Process 'Software Development Engineering'			
6.4.1	Require-ments Capture	• Functional requirement capture • Technical requirement capture • Document requirements • Requirements review	Retro-capture of requirements Functional requirements Technical feasibility study Document requirements Non-functional requirements Architectural feasibility study Requirements review
7.1.3 7.1.4	Software design	• Design tailoring • Architectural design • Detailed design • Database design • Design review	Maintenance tasks Retro-engineering Architectural design Database analysis and design Detailed design Design review
…			
Group Process 'Software Development Support'			
6.2.1	Methods and tools support	• Process establishment • Process improvement • Tools support	Life cycle process modelling Project process modelling Process monitoring implementation Tool usage guide
…			

Software companies use assessment of software processes for capability determination and process improvement [24]. Although we think that process assessment as defined in ISO/IEC 15504 or CMMI is beyond the reach of young engineers, we believe that a simplified Process Reference Model and a personal Process Assessment Model are required to provide a basis for the practice of software engineering. Furthermore, we think that these models may provide an initial structure of the repertoire.

We observed that our apprenticeship scenes and work placement periods mobilized a similar set of apprentices' competency. As mentioned in Section §III.A.3, each process is associated with a family of competencies constituted with a list of knowledge topics and a set of abilities or skills required to perform the process. We believe that a first step in competency assessment should be made by the engineer him/herself through a self-assessment of abilities at a maturity level. The assessment scale grows from 0 to 5; - 0 - Don' know anything; - 1 - Smog: vague idea; - 2 - Notion: has notion, a general idea but insufficient to an operational undertaken; - 3 - User: is able to perform the ability with the

help of an experienced colleague and has a first experience of its achievement; - 4 - Autonomous: is able to work autonomously; - 5 - Expert: is able to act as an expert to modify, enrich or develop the ability.

Four times a year, each young engineer is required to auto-assess his/her maturity level for each ability of the 13 competency families (as well as for each transversal competency). This periodic self-confrontation with the competency reference model is called a competency inventory and is performed while auto-analyzing the tasks performed and his/her achievement level with the abilities defined in the family.

Periodic competency inventories are stored in a Content Management System (CMS). The CMS is hierarchically structured according to the group process / process decomposition. This structure is also intended to store artefacts that may be of interest in illustrating the ability determination. When the apprentice needs to relate a task performed with a process's ability, he/she has to write an entry associated with the process and may link this entry with artefacts stored. It constitutes a rudimentary portfolio, but is sufficient for our purposes.

Our system reference models are presented in Figure 4.

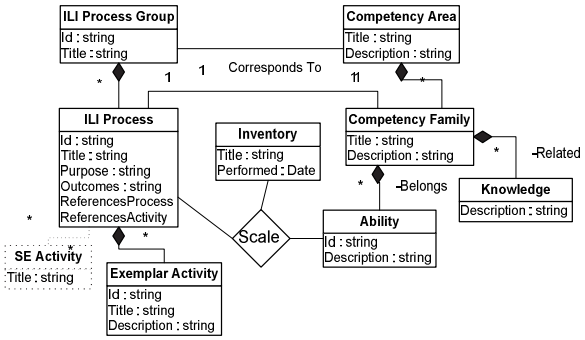


Figure 4. A model of Process Reference Model -PRM- (on the left) and Competency Reference Model (on the right). Periodic inventories hold abilities self-assessments.

5) Technical issues

As the project moves forward, information is constantly updated - in content, and in structure too. Moreover, metadata management is required. In order to support these purposes, we propose a very simple architecture based on the use of several inter-linked semantic wikis. Semantic wiki is the most flexible tool in order to record and shape a structured content.

The structural elements of these reference models do not change as projects go along and theirs events are recorded. In order to facilitate links between the project journal and these models PRM, information is stored into two semantic wikis and a Content Management System (CMS):

• http://oysterz.univ-brest.fr/12207, the 12207 wiki is a hypertext reference of the ISO/IEC 12207:2008.

• http://oysterz.univ-brest.fr/company, the upper-level of the company wiki contains decompositions Processes group / Processes / Exemplar Activities and Stages / Scenes.

• a CMS contains the periodic inventories together with related artefacts used as witness of the maturity level.

B. Process enactment

1) Questioning our hypotheses

A case study, discussed below, will provide observations and information belonging to the different types discussed in §III.B (presentations, accounts and comments) and structured with the models of §IV.A (hierarchical group process/process/scenes model and competency reference model) enhanced of events' modelling of the project-in-action presented above.

Self-recording of activity is materialized by adding new items either in the portfolio or in the wiki, which is acting as a journal. A measurement of each apprentice's recording (from low to high) gives an indication that the apprentice is aware of the structure of his/her repertoire and able to use it to classify their experiences. It will provide an empirical verification of hypothesis H2: the Course-of-Action observatory may help to be aware of his/her repertoire.

Once experiences are self-recorded, apprentices are periodically performing self-assessment and self-analysis. Comparing self-assessments of previous cohorts with those of our Study Team may provide an indication that the use of an observatory is influencing the maturity level reached by the team at study.

The self-analysis of the Course-of-Action is providing a view of the enacted processes as they are reconstructed and perceived by the apprentices themselves. On the other hand, the project should follow processes as they are prescribed by the company's tutor. A qualitative evaluation of the process reconstruction gives an indication of the gap between prescribed and enacted processes. It will provide an empirical verification of hypothesis H1: self-analysis and self-assessment helps an apprentice to reveal theory-in-use.

2) An empirical case study

This case study is based on the activity of a team of 6 young software engineering apprentices, the former author as a participant-to-observe having a direct contact of the team members, sharing their environment and taking part in the activities of the team, the latter conducting formal assessments as they happen. This case study observes the whole course of the project. As pointed out by Singer and Vinson [25], apprentices' consent is required. At the beginning of the project they were informed on the field study and its objectives, and they agreed to participate.

The project is a semantic annotation tool. The main goal of the project is to provide a semantic annotation tool able to annotate Web resources, search in different modes, browse hierarchically or with facets, and manage RDF vocabularies. The project uses Jena (http://jena.sourceforge.net) an open-source Semantic Web programmers' toolkit as RDF API.

3) Planning and monitoring the project

The project enactment is based on the process models of the previous section, a Y-shaped life cycle that separates resolution of technical issues from resolution of feature issues [26] and a typical WBS (Work Breakdown Structure: "*a deliverable-oriented hierarchical decomposition of the work to be executed by the project team to accomplish the project objectives and create the required deliverables. It organizes and defines the total scope of the project*" [27]).

The WBS has a structural and a temporal breakdown. Each process is achieved through scenes defined from exemplar activities. The WBS is temporally organized in 9 stages of 2 weeks. The planning of each stage orchestrates several work scenes. Scenes will be performed by team members, and should produce artefacts.

Information is recorded at mid-level of the company wiki (http://oysterz.univ-brest.fr/company). This mid-level structure acts as a simple but realistic model of a project: breakdown of the project stages into work scenes; allocation of persons to scenes; expected inputs and outputs. This mid-level is filled with instances (wiki pages) corresponding to the project WBS and updated regularly. The structure of this mid-level is given in the right half of Figure 5.

4) Recording the project progress

We state in §III.B.3 that software artefacts produced by the team will serve as (i) presentations. As the project progresses, events of interest are recorded in a journal associated with significant artefacts they may have used or produced. As described in III.B.3, each individual Course-of-Action is accounted for, on a 2-3 days basis, in an instance of the smaller unit, called a Performed Activity. Apprentices create a wiki page for each individual activity performed during the stage, fill this page with a short description of activities performed, link this page with related other pages (scene, person, artefact), and upload artefacts. At the end of each 2-week period apprentices account for individual and collective work in the finest grain of collective Course-of-Action, called a Course-of-Action Unit, which organizes several individual Performed Activities. This (ii) accounting provides a first-level of self-confrontation, as required by the Course-of-Action observatory.

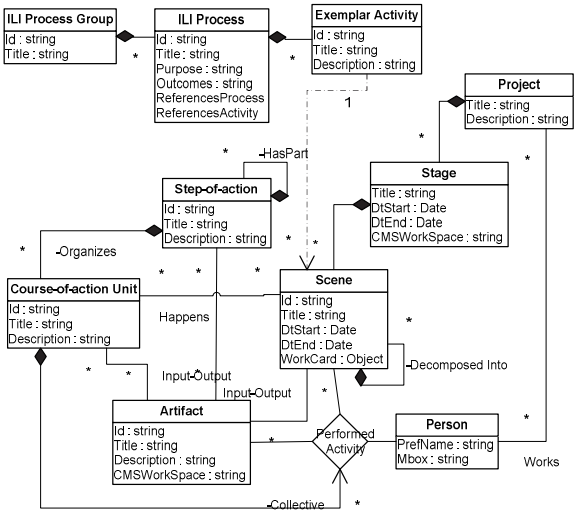


Figure 5. Representation of an enacted project. The lifecycle of a project is organized into stages, composed of scenes. During a scene, actors perform an SE activity inspired by an exemplar activity, yet contextual to the project. Input and output work products (artefacts) are linked to scene, activity and process. Self-observing the action leads to a rebuilding of project processes into steps of Course-of-Action units.

Every two months, apprentices perform a first-level Course-of-Action analysis on the state of the project's software processes. Each apprentice works on few processes and builds intermediary process elements called Steps-of-Action, based on historical Course-of-Action Units related to this process.

All information is recorded in the lower-level of the company wiki (http://oysterz.univ-brest.fr/company). The structure of this lower-level is shown on the left side of Figure 5.

5) Self-assessment

An attempt must be made to relate the university and industrial phases of the student's experience to one another. Fortunately, the competency assessment model of our system (which could be considered to be the learning objectives) is based on a simplified model of professional activities. So it may help apprentices to link up their competency building, thus avoiding their having to 'climb two ladders simultaneously' [28].

As stated at §IV.A.4, each apprentice is asked to self-analyze the activities they carried out (during both university and industrial periods) four times in the course of the year, in line with the immersion system's competency assessment model. Students assess their own maturity, on a scale of 0 to 5, for each ability or transverse competency.

In order to prepare the periodic competency assessments, apprentices use the CMS as a portfolio which hosts significant work and interesting artefacts. At any moment of the year, either in industry or at university, the apprentice may encounter a work situation, or perform a task which they perceived to be a significant experience. Within the competency reference model, they must identify one (or several) skills related to this experience, and then associate a new entry with a description of the experience, uploading artefacts that testify to this experience.

6) Building their own repertoire

The process models presented in Section IV.A are used throughout the year to structure the apprentices learning process in the reflective practicum at university (and partially in industry). The process models are also providing structure for the self-recording of apprentices' Course-of-Action and periodic self-assessments of competencies. We believe that these models provide an initial structure for the repertoire, acting as knowledge paths towards recording and retrieval practices within the repertoire.

V. EXCERPTS OF RECORDINGS AND ANALYSIS

We give some quantitative facts about the case study and empirically question the research hypotheses of Section 2.

A. Wiki accounting

The project is now complete, and Table IV gives the number of instances (wiki pages) in each category:

- 69 work scenes occurred,
- students carried out 118 Performed Activities,
- roughly 100 artefacts were produced.

For each process, we have the quantity of: Work Scenes (SCE); Performed Activities (PAY: individual); Course-of-Action Units (CAU: collective), and Steps (STE: higher-level construct). The 6th column gives an indication of the quantity of Software Engineering Activities that may be envisaged in the process. The 7th and 8th columns report the 12207 breakdown of related processes: number of activities (Act) in (the) process (es) and number of corresponding tasks (Tsk). The 9th and final column gives the number of Base Practices (BP) in the corresponding process from the 15504 standard.

TABLE IV. QUANTITATIVE FACTS FROM THE CASE STUDY.

Process	*SCE*	*PAY*	*CAU*	*STE*	*SE Act.*	*Act*	*Tsk*	*BP*
Project management	13	22	13	5	5	7	14	15
Quality insurance	2	2	1	2	2	4	16	8
Configuration management	2	2	2	3	3	6	6	10
Requirements capture	10	18	10	3	5	5	12	6
Software analysis	2	2	2	2	2	1	3	6
Technical architecture	7	10	5	3	4	2	2	-
Software design	7	9	5	4	4	2	15	12
Software construction	8	16	4	3	5	1	5	4
Integration - validation	8	12	5	4	5	6	20	20
Technical support	8	16	2	2	2	3	4	6
Methods and tools support	3	3	3	3	2	-	-	6
Documen - tation	2	4	2	2	2	4	7	8
Installation - deployment	1	2	2	1	2	2	5	6

Students report on their activity at the end of each 2-week stage. Where an activity has extended beyond a single stage (e.g. technical support or coding), students adopt a simple strategy: creating one single mid-level structure (Course-of-Action Unit), and linking it to individual low-level units (Performed Activity) belonging to different stages.

B. Self-Recording

In order to evaluate the impact of the Course-of-Action observatory, we measure the use of the portfolio associated with the competency assessment model. We compare the two teams of the 2008-2009 cohort: a Control Team – which does not record its activity in an observatory – and Study Team. Each team comprises 6 apprentices.

For each process of the Process Reference Model, and for each apprentice, we measured the number of associated entries in the portfolio. Table V shows, for each team, the minimum, maximum and average number of entries.

For the Study Team, the average for each process is significantly higher than that of the other team. Remember that each apprentice of the Study Team has to report their activity in the observatory after each period at the university. Comparison of Repertoire Use Between a Control Team and the Study Team .

TABLE V. COMPARISON OF REPERTOIRE USE BETWEEN A CONTROL TEAM AND THE STUDY TEAM .

	08-09 Control Team			**08-09 Study Team**		
Process	***Min.***	***Max.***	***Avg.***	***Min.***	***Max.***	***Avg.***
Project Management	0	4	2.33	1	5	2.5
Quality Insurance	0	2	1	0	3	1.16
Configuration Management	2	3	2.5	2	5	3.5
Requirements Capture	0	7	3.83	2	8	4.66
Software Analysis	1	4	3	1	8	3.83
Technical Architecture	1	5	2.83	1	7	3.83
Software Design	3	6	4.5	3	8	5.83
Software Construction	1	7	4.16	3	8	5.33
Integration - Validation	1	4	2	1	5	3
Technical Support	2	4	2.66	2	10	3.33
Methods and Tools Support	1	4	2.16	2	8	3.83
Documentation	1	6	2.83	1	8	4
Installation - deployment	2	6	3.16	2	7	4

It is plausible to think that this periodic self-confrontation helps them to be aware of the Process Reference Model that structures the repertoire and facilitates filling the repertoire.

We may reasonably argue that our hypothesis H2 is well-grounded: the Course-of-Action observatory helps an apprentice to be more aware of the repertoire.

C. Self-assessment

A comparison of the different systems can be drawn from personal competencies follow-up. For the 13 competency families, Table V presents three self-assessment averages (in September, February and May) for the 2006-2007 cohort (previous system: no work placement), the 2007-2008 cohort (new system: with work placement), and the 2008-2009 case study team (current system: with work placement and observatory). Each cohort comprises 2 teams of 6 students.

All families make steady, and roughly equivalent, progress - with or without work placements (and with or without observatory). Due to the small number of students in cohorts, and the paucity of our statistical knowledge, no statistical comparison was performed. However, there is no evidence to indicate that the observatory helps understand software processes and reveal theories-in-use.

Some small differences can be pointed out: the 08-09 team-members assess themselves at a lower level than previous cohorts, except in terms of Project Management.

This may indicate that building the observatory, and reconstructing processes, have enforced this competency.

TABLE VI. TECHNICAL COMPETENCIES: PERSONAL FOLLOW-UP FOR THE 2006-2007, 2007-2008 COHORTS, AND 2008-2009 STUDY TEAM

	06-07 Cohort			07-08 Cohort			08-09 Team				
Competency family	**9/6**	**2/7**	**5/7**	**9/7**	**2/8**	**5/8**	**9/8**	**2/9**	***S***	**5/9**	***S***
Project Management	1.5	2.8	3.4	1.3	2.7	2.9	1.2	2.3	*1*	3.5	*3*
Quality Insurance	1.1	2.4	2.8	1.4	2.3	2.4	1	1.1	*2*	1.5	*2*
Configuration Management	1.2	1.8	2.9	1.6	2.9	3.0	1.2	1.6	*2*	2.7	*3*
Requirements Capture	2.1	3.2	3.6	1.8	2.8	3.0	2	2.3	*2*	3.2	*2*
Software Analysis	3.6	3.7	3.9	2.4	3.0	3.3	2	2.1	*2*	2.7	*3*
Technical Architecture	1.4	2.4	3.0	2.0	2.8	2.9	1.4	2.1	*2*	2.7	*3*
Software Design	2.8	3.2	3.5	2.3	3.1	3.6	1.8	2.1	*2*	3	*4*
Software construction	2.7	2.7	3.1	2.5	2.9	3.4	2.2	2.5	*2*	3	*2*
Integration - Validation	1.2	1.3	2.7	1.3	2.0	3.2	1.2	1.8	*2*	2.3	*3*
Technical support	2.3	3.0	3.4	2.4	3.1	3.5	1.4	1.9	*2*	2.3	*2*
Methods and tools support	1.7	2.6	3.2	2.0	2.5	2.9	1.2	1.8	*1*	2.5	*3*
Documen-tation	2.8	3.3	3.5	3.1	3.3	3.7	2.4	2.5	*2*	2.8	*2*
Installation - Deployment	2.4	3.3	3.5	2.9	3.3	3.7	1.6	2.6	*3*	3.2	*3*

Even though we were unable to confirm our hypothesis H1, we believe that relating the project observatory to the personal follow-up of competencies may improve apprentices' overall understanding of processes. A brief example: the complete Software Requirements Capture Process was performed in 10 scenes, spread over several sequences. Looking at the individual progress of an apprentice regarding this process, we note that her self-assessment stayed at a low maturity level of 2 - Notions - despite the fact that she had participated in several requirements-related scenes and observed her team-mates performing other related scenes. It is only after her participation in the Software Specification Requirements Document update that she assessed herself at level 4 - Autonomous – and finally perceived that the different Course-of-Action units related to requirements were related to the same field.

D. Process reconstruction

We concentrate on reconstruction by the students of higher-level Course-of-Action from the smaller units.

In Table VI, column S represents the average of the student carrying out reconstruction for this process (between February and May) - but there is no evidence that this work improved their understanding of the reconstructed process. Analysis should be correlated with the participation (and commitment) of students into scenes that are tied to the process. Further work is required.

In Table IV, the number of 12207 tasks (and 15504 Base Practices as well) give an indication as to the density of the process. The higher these numbers are, the greater the complexity - it should therefore lead to a process reconstruction involving a higher number of Steps-of-Action related to a roughly equivalent number of Software Engineering Activities. A difference between the 5th column (STE) and 6th column (SE Act.) - e.g. Requirements capture - may indicate that the reconstruction failed.

From the tutor's point of view, steps creation was haphazard. Simple processes, such as Design, have been correctly reconstructed. But, since a large number of BP (Base Practices) in Table IV indicate a complex process which may be oversimplified in the practicum (e.g. Configuration Management or V&V), the reconstruction was correct regarding the simplified process but it is partly inaccurate. For complex processes involving many scenes, reconstruction may fail - probably because apprentices are unable to perceive an abstract view of the process. This is what happened during the Software Requirement Process, where students were not able to create the Steps that would establish significant links with smaller units, nor inter-wikis links with the corresponding 12207.

VI. CONCLUSION AND FUTURE WORK

Argyris and Schön make a distinction between the two contrasting theories of action: *theories-in-use* and *espoused theories*. We proposed to adapt the Course-of-Action framework to observe software engineering apprentices' activity in the course of their final year. Two hypotheses are discussed: (1) that self-analysis and self-assessment help reveal theories-in-use, and (2) that the Course-of-Action observatory helps raise awareness of the repertoire. As a case study, the activity of a team of 6 young software engineers accompanied with two participants-to-observe is currently recorded in the observatory.

Observations are presentations (software artefacts), accounting (events in the project diary or a portfolio) and comments (steps reconstruction and activity reports). This self-observation builds a hierarchy of SE processes used as a structure for young engineers' repertoires. Four times a year, apprentices self-confront the work they did, self-assessing against a personal ability model.

Current progress with this work suggests that the process models (a personal Process Assessment Model and a simplified Process Reference Model) may form an initial structure of the repertoire, and that the observatory helps apprentices to be aware of their own experiences.

Further work is required to consider how the Course-of-Action analysis fits in with Reflection-in-Action and how it impacts the software engineering apprentices' ability to cope with innovation and change.

ACKNOWLEDGMENT

The authors wish to thank François-Xavier Bru, Gaëlle Frappin, Ludovic Legrand, Estéban Merrer, Sylvain Piteau, and Guillaume Salou for their participation in this work.

REFERENCES

[1] V. Ribaud, and P.Saliou, "Revealing Software Engineering Theory-in-Use through the Observation of Software Engineering Apprentices' Course-of-Action", in Proceedings of 2009 Fourth International Multi-Conference on Computing in the Global Information Technology, New York: IEEE Press, pp. 202-210, 2009.

[2] D. Schön, D., "Educating the Reflective Practioner: Toward a New Design for Teaching and Learning In the Professions", San Fransisco: Jossey-Bass, 1987.

[3] J. E. Tomayko, "Carnegie Mellon's software development studio: a five year retrospective" in Proceedings of the 9th Conference on Software Engineering Education, New York: IEEE Computer Society Press, pp. 119-129, 1996.

[4] S. Kuhn, "The software design studio: an exploration", IEEE Software, Volume 15 (2), March-April 1998, pp. 65-71.

[5] D. Schön, "The Reflective Practitioner", New York: Basic Books, 1983.

[6] ISO/IEC 12207:1995, AMD 1:2002, AMD 2:2004, "Information technology -- Software life cycle processes", Geneva: International Organization for Standardization (ISO), 1995, 2002, 2004.

[7] C. Argyris, and D. Schön, "Theory in practice: Increasing professional effectiveness", San Fransisco: Jossey-Bass, 1974.

[8] L. Pinsky, and J. Theureau, " Activite cognitive et action dans le travail, Tome 1: les mots, l'ordinateur, l'operatrice", Collection de Physiologie du Travail et Ergonomie, vol. 73, Paris: CNAM., 1982.

[9] J. W. Maxwell, "Using Wiki as a Multi-Mode Publishing Platform", in Proceedings of the 25th annual ACM international conference on Design of communication, New York: ACM, pp.196-200, 2001

[10] ISO/IEC 12207:2008, "Information technology -- Software life cycle processes". Geneva: International Organization for Standardization (ISO), 2008.

[11] ISO/IEC 15504:2004, "Information technology -- Process assessment". Geneva: International Organization for Standardization (ISO), 2004.

[12] D. Schön, "Educating the Reflective Practitioner" in Meeting of the American Educational Research Association, 1987.

[13] O. Hazzan, and J.E. Tomayko, "Reflection processes in the teaching and learning of human aspects of software engineering", in Proceedings of 17th Conference on Software Engineering Education and Training, New York: IEEE Press, pp. 32- 38, 2004, doi:10.1109/CSEE.2004.1276507

[14] P. Halloran, "Organisational Learning from the Perspective of a Software Process Assessment & Improvement Program" in: 32nd Hawaii International Conference on System Sciences. New York: IEEE Press, 1999.

[15] J. Theureau, "Course-of-Action analysis & Course-of-Action centered design" in: Hollnagel E. (ed.), Handbook of Cognitive Task Design, New Haven: Lawrence Erlbaum Ass., 2003

[16] C. Argyris, and D. Schön, "Organizational learning: A theory of action perspective", Reading: Addison Wesley, 1978

[17] O. Hazzan, "The reflective practitioner perspective in software engineering education", Journal of Systems and Software, Vol. 63 (3), September 2002, pp. 161 – 171, ISSN:0164-1212

[18] J. Theureau, G. Filippi, and I. Gaillard, "From semio-logical analysis to design: the case of traffic control" in Colloquium "Work activity in the perspective of organization and design", Paris: M.S.H., 1992

[19] J. Theureau, and G. Filippi, "Analysing cooperative work in an urban traffic control room for the design of a coordination support system, chapter 4" in: Luff, P., Hindmarsh, J., Heath, C. (eds.) Workplace studies, Cambridge Univ. Press, 2000, pp. 68-91.

[20] P. Meirieu, "Si la compétence n'existait pas, il faudrait l'inventer" in IUFM de Paris Collège des CPE, 2005, (accessed April 2009) http://cpe.paris.iufm.fr/spip.php?article1150

[21] V. Ribaud, and P. Saliou, "Towards an ability model for software engineering apprenticeship". Italics, Innovation in Teaching And Learning in Information and Computer Sciences, Vol.6 (3), July 2007, pp. 97-107.

[22] T. C. Lethbridge, S. E. Sim, and J. Singer. "Studying Software Engineers: Data Collection Techniques for Software Field Studies", Empirical Software Engineering , vol. 10 (3), July 2005, pp. 311 – 341, doi:10.1007/s10664-005-1290-x

[23] J. Leplat, "Regards sur l'activité en situation de travail - Contribution à la psychologie ergonomique", Paris: Presses Universitaires de France, 1997.

[24] Software Process Improvement and Capability dEtermination (SPICE), Software Process Assessment - Version 1.00, http://www.sqi.gu.edu.au/spice/docs/baseline, 1995

[25] J. Singer, and N. G. Vinson, "Ethical issues in empirical studies of software engineering", IEEE Transactions on Software Engineering, Vol. 28 (12), Dec 2002, pp. 1171- 1180, doi:10.1109/TSE.2002.1158289

[26] P. Roques, and F. Vallée, "UML en action", Paris: Eyrolles, 2002.

[27] ISO/IEC FCD 24765, "Systems and software engineering – Vocabulary". Geneva: International Organization for Standardization (ISO), 2009.

[28] J. Topping, Sandwich courses, Phys. Educ. Vol. 141 (10), 1975, pp. 141-143, doi:http://iopscience.iop.org/0031-9120/10/3/003

Modernization of a Legacy Application: Does it Have to be Hard?

A Practical Industry Cooperation Case Study

Arne Koschel, Carsten Kleiner
Faculty IV, Dept. for Computer Science
Applied University of Sciences and Arts
Hannover, Germany
{akoschel | ckleiner}@acm.org

Irina Astrova
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia
irina@cs.ioc.ee

***Abstract*—Modernization of a legacy application is not very hard any more. Whereas this may have been true a couple of years ago, this paper describes a case study, which shows that the modernization is significantly easier if modern integration tools, a service-oriented architecture and Web services are used. This is by contrast to a common belief that the modernization is always hard, regardless of the technologies used. The case study, where bachelor students succeeded to carry out the modernization of a legacy application, shakes that belief. The students neither had previous experience with the technologies used in the legacy application nor with the ones used for the modernization. As major contributions this paper provides an overview of approaches to modernization, a full case study for the modernization (including details on business process analysis, architecture, and tools), and comprehensive 'lessons learned' to help for 'the practice'.**

***Keywords*—service-oriented architecture; mainframe; legacy integration; experience report; Web service**

I. INTRODUCTION & MOTIVATION

Declared 'dead' for quite a while now, many legacy mainframe applications are still happily productive and will continue to be. Indeed, until today legacy applications that are based on mainframe database management systems (DBMSs) like Adabas and associated fourth generation programming languages (4GL) such as Natural, are still often in practical use. However, those applications are often only badly, if at all, integrated with newer enterprise applications. The integration of legacy application assets is required, e.g., due to joint use of functionality or data. It is an important task, which occurs frequently in industrial practice.

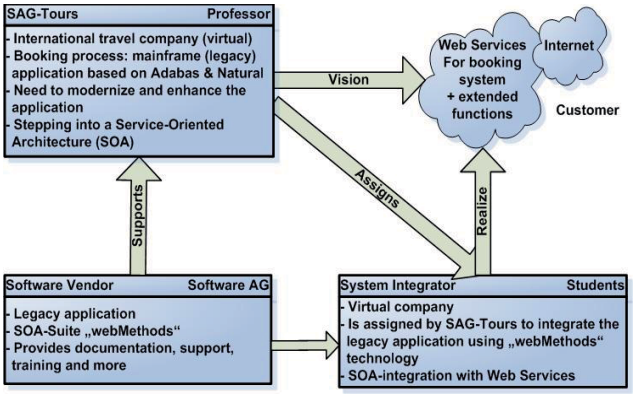


Figure 1. Overview of the SAG-Tours project

Initially, a punctual integration of the legacy assets was achieved by means of 'traditional' enterprise application integration (EAI) technology (cf. [2][6]). Nowadays a service-oriented architecture (SOA) [7][13] proposes a promising solution to this task.

This paper describes a case study for legacy modernization based on integration technology and Web services. Used in conjunction, they served as the base for integration of an existing legacy mainframe application (SAG-Tours) into an up-to-date distributed SOA.

Figure 2. User interface of SAG-Tours application after modernization

The case study was done in scope of the SAG-Tours project (see Figure 1). This project involved research and industry cooperation with a German software company, Software AG (SAG). The goal of the SAG-Tours project was

to integrate the SAG-Tours mainframe application into a modern Web environment – a requirement being driven by Software AG customers. The customers wanted the SAG-Tours application to become Internet-ready quickly (within a year), thus giving end users the possibility to access the application via a Web browser (see Figure 2). Previously, 'green screens' were the only way to access the application (see Figure 3). The SAG-Tours project team consisted of 10 final-year bachelor students supervised by 2 professors. The students had an average working effort of 1 day per week per person. The team was given some 'getting started' and 'configuration hotline' help from Software AG.

This paper provides two major contributions. First, it shows how such a technically complex integration task (where both old existing systems and several integration technologies are involved) can be undertaken. Second, it shows that this task could be carried out even with relatively inexperienced students under only moderate supervision of professors. This means the integration of at least functional-wise not too complex legacy applications into a SOA should not be a too difficult work any longer.

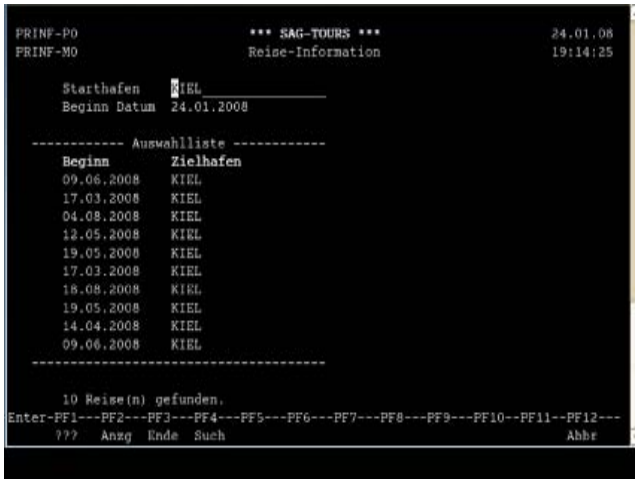


Figure 3. User interface of SAG-Tours application before modernization

This paper is an in-depth extension of our previous work [1]. The related work has been extended by providing several new references to related academic and industrial publications. We examine different possible approaches to modernization of a legacy application. Also, we explain details on our approach to modernization of the SAG-Tours application and our architecture that supports integration of the application into a SOA. Finally, we bring together all the lessons learned during the SAG-Tours project that can be useful for application in other similar legacy modernization projects.

The rest of the paper is organized as follows. Section II describes the related work. Section III describes the SAG-Tours application and its background technologies (viz., Natural and Adabas). Section IV analyzes existing approaches to modernization of a legacy application (viz. packaged applications, code conversion, re-hosting, re-architecting, and SOA enablement). Section V gives details on our approach to modernization of the SAG-Tours application (viz. SOA enablement). Section VI gives details on our integration architecture. Section VII describes lessons we learned during the SAG-Tours project; it is followed by conclusion and outlook to possible future work.

II. STATE OF THE ART

Related work especially regarding integration tools can be found in industry as well (see, e.g., SAP [19], IBM [20], Oracle [21][22], Software AG [23] and Microsoft [24]). However, because Software AG's integration technology stack was given us as a pre-requisite, we focus here on academic related work only.

Although Canfora et al. [17] use a wrapper approach as we do, they focus mainly on interactive functionality.

Englet [5] proposes a bottom-up integration approach, which is not restricted to interactive components. It is, however, suboptimal with respect to process modeling because it might not take process optimization into account.

Smith [12] discusses several ways to introduce a SOA into an enterprise, including legacy assets, but on a general level. Erradi et al. [4] discuss similar strategies. Instead of a general discussion, we focus mainly on concrete technical integration aspects.

Lewis et al. [9][10][15] develop a service-oriented migration and reuse technique (SMART) to assist organizations in analyzing legacy components in order to determine if they can be reasonably exposed as services in a SOA. SMART provides a preliminary analysis of viability of different migration strategies and the associated costs, risks and confidence ranges for each strategy. In particular, SMART gathers information about legacy components and produces the best migration strategy for a given organization. Thus, SMART helps organizations to select the right migration strategy. SMART can be used to analyze what legacy functionality can be re-used in a SOA. However, we do not need this analysis, because it was pre-defined for us, which legacy functionality had to be re-used. We consider this to be an everyday situation in practice.

Erl [14] introduces a pattern-oriented background for a SOA. While being helpful in general, more detailed work is required for a concrete integration task.

Sneed [18] proposes a salvaging and wrapping approach (SWA). This is a three-step procedure for creating Web services from a legacy application code. These steps are: (1) salvaging the legacy code; (2) wrapping the legacy code; and (3) exposing the legacy code as a Web service. SWA is effective in process and service integration. But it provides limited support for content integration by wrapping second-level Web services. This is similar to the second step of our approach.

Ziemann et al. [25] describe a business-driven legacy-to-SOA migration approach called enterprise model-driven migration approach (EMDMA). This approach is based on enterprise modeling, by introducing an elementary process model between the business function tree and the tree related to the legacy application, which is then aligned to the function tree of the legacy application. Finally, it applies a transformation from the legacy business process model to the

SOA process model. EMDMA draws attention to the fact that aspects such as functional granularity, security, reliability, scalability, etc. are not taken into account sufficiently during the migration. Thus, there is a need for investigating how these aspects of the legacy application can be mapped to a SOA.

III. SAG-TOURS APPLICATION

SAG-Tours [11] is a legacy mainframe application written in Software AG's Natural. It is based on a 1-tier terminal mainframe architecture (see Figure 4). Functional wise one has the possibility to order fictitious cruises.

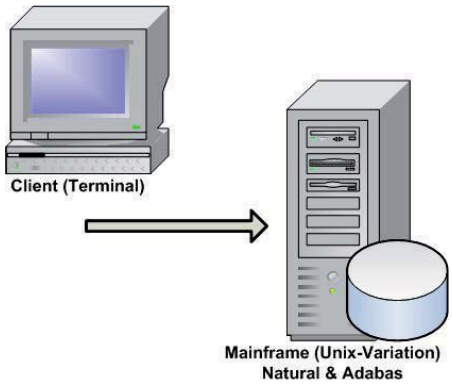


Figure 4. 1-tier system architecture of SAG-Tours application

Technically terminal emulations are used, which communicate with a Unix variation of Natural via telnet protocol. The SAG-Tours application itself has a connection to an Adabas database [3]. Adabas is a high performance mainframe DBMS, which is internally based on so-called inverted lists. An example for a Natural query would be as follows:

```
FIND CRUISE
    WITH START HARBOR= 'CURACAO'
```

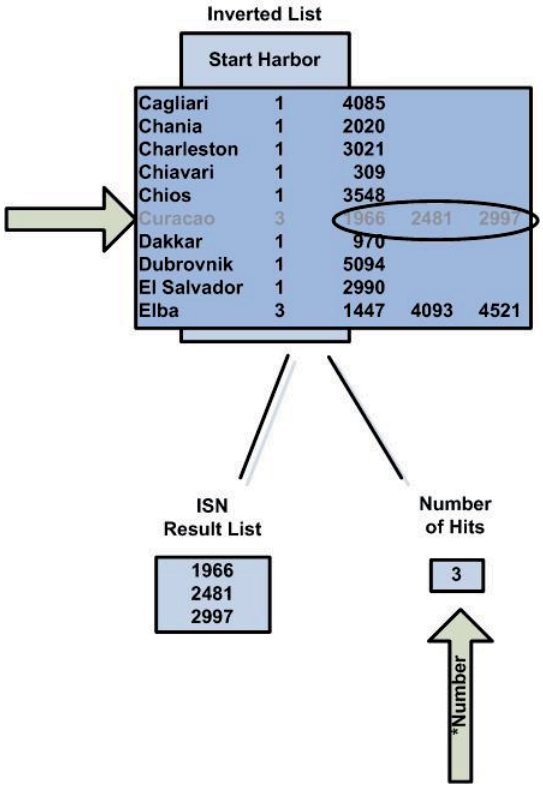


Figure 5. Internal structure of an Adabas example query using inverted lists

This query delivers all journeys with the starting harbor 'CURACAO'. Figure 5 shows '3' as the total hits number of query results. It also shows the resulting internal sequence numbers (ISNs). These ISNs can be interpreted as logical pointers to the relevant resulting tuples.

IV. APPROACHES TO MODERNIZATION OF LEGACY APLLICATION

Legacy modernization is the process to supplement or replace an organization's legacy applications and technologies using newer applications and technologies that are based on open standards, while retaining business logic [28].

There are five basic approaches to legacy modernization (that can be used alone or combined):

- Replacing legacy applications with packaged applications.
- Re-architecting legacy applications.
- Legacy application code conversion (also called automated migration of legacy applications [28]).
- Re-hosting legacy applications.
- Enabling SOA (also called enabling Web [29], re-interfacing [29] or business logic wrapping [26]).

There are advantages and disadvantages with all these approaches. An advantage of one approach is usually a disadvantage of another and vice versa. But "organizations that are SOA-enabling their legacy applications on the legacy platforms are outperforming those that are using any other

approach. They report better productivity, higher agility, and lower costs for legacy modernization projects." [29].

Since legacy applications are typically not architected with a SOA and Web services in mind, careful selection of an approach is required before modernizing a legacy application. Depending on the approach selected, the legacy application may require big, small or no change at all. Therefore, it was important to choose the right approach for the SAG-Tours application.

A. Packaged Applications

This approach [28] consists of replacing the legacy application with a packaged (commercial off-the-shelf) one made up of SOA components. These components can then be combined with re-architected components, re-hosted components and automatically migrated components using a SOA orchestration engine.

Because packaged applications are seen as sets of reusable components, the biggest advantage of this approach is the increased agility. However, this approach does not work in a situation where legacy applications have unique functionality that cannot be replicated by packaged applications. In this case, the business logic can be retained from the legacy application using one of other approaches such as re-architecting or re-hosting. Therefore, we rejected this approach.

B. Code Conversion

This approach [27][28] consists of converting the legacy application code into a new programming language (e.g., Java). It is often used in combination with replacing the legacy application with a packaged one.

The biggest advantage of code conversion is that the process of migrating, e.g., from Natural to Java can be automated; i.e., it can be carried out by a machine and require no human intelligence during the migration process.

Because a machine will carry out the migration process, it can be done quickly and consistently. But it only works if the gap between the legacy architecture and the new one is relatively small. E.g., it is not possible to convert the procedural design of Natural into the object-oriented design of Java. The fundamental design concepts of Java – e.g., a class and its behavior – are architectural concepts that require human intelligence to design. The designer of a truly object-oriented application will use these concepts in new ways that cannot be recovered from a legacy application designed using procedural techniques. Therefore, although the automatic migration of Natural code into Java can be done, but the resulting Java code will not be the same Java code that would be designed for a truly object-oriented application.

The biggest disadvantage of code conversion is that it is much more invasive to the legacy application than other approaches to legacy modernization such as re-hosting and SOA enablement. Therefore, we rejected this approach also.

C. Re-hosting

This approach [28] consists of migrating the legacy application to a lower cost platform. It can be used in combination with code conversion. E.g., during the re-hosting process, the legacy database calls to a mainframe database such as Adabas can be eliminated.

The biggest advantage of re-hosting is that it is non-invasive to the legacy application because the application is left "as-is".

Since re-hosting does not change the legacy application, one disadvantage of this approach is that it forces a continued reliance on legacy skills. Another disadvantage is that re-hosting retains much of the legacy architecture. This means that the implementation of Web services could be cumbersome. Therefore, we rejected this approach also.

D. Re-architecting

This approach [28] consists of extracting business logic from the legacy application, building a new application, integrating this new application with the legacy one, and finally, shutting down the legacy application.

The biggest advantage of re-architecting is that it maximizes the benefits of a SOA and new technologies. But it is the most expensive approach to legacy modernization – a legacy modernization project can span many years. Therefore, we rejected this approach also.

E. SOA Enablement

This approach [26][28] consists of wrapping business processes and presenting them as Web services to an enterprise service bus (ESB). This is the approach we selected.

The biggest advantage of SOA enablement is that it provides immediate integration of the legacy application into a SOA. In addition, this approach is relatively non-invasive to the legacy application. Therefore, legacy components can be used as part of a SOA with no or little risk of destabilizing the legacy application.

However, like re-hosting, SOA enablement forces a continued reliance on legacy skills. Another disadvantage of this approach is the need for communicating among disparate environments because the legacy components continue to reside on the legacy platform. However, using SOA enablement combined with re-hosting can eliminate the need for such communication because all the components – the re-hosted components that have been integrated into a SOA, the new components, the packaged application components, and the SOA orchestration engine that brings them all together – reside on the same new platform. This also makes it easier to convert the legacy components into a new programming language such as Java.

V. Conceptual Integration Approach and Business Process Model

In this section we will show how the domain specific business process model for the case study has been developed and present some conceptually important aspects of the result. We will also describe the conceptual integration approach we have chosen and the advantages of this approach.

A. Conceptual Integration Approach

As explained above, for integrating the SAG-Tours application into a SOA, we decided to follow SOA enablement [26], which continues to use the application itself. We selected SOA enablement primarily because rather quickly (within a year) this approach can bring the application into a modern Web environment, where the application can be accessed via a Web browser. This is one of the biggest advantages SOA enablement has over other approaches to legacy modernization.

Our cooperation partner, Software AG, also offers tools that use screen scraping to extract complete work processes from a legacy application and make them available as Web services (see Figure 6). We did not use these tools either. On the one hand, we did that to stay technology-neutral as far as possible. On the other hand, these tools typically yield only a few rather coarse-grained services in the business process layer (cf. [7]) and no services in the underlying layers of business services and basic services. By doing so, increased speed in an integration of the legacy application can be reached; but the increased flexibility of process definitions by variable combination of services of the underlying layers, which is one of the main goals of introducing a SOA, may not be achieved. We will show below how the services of the business service layer have been variably composed into business processes in the case study.

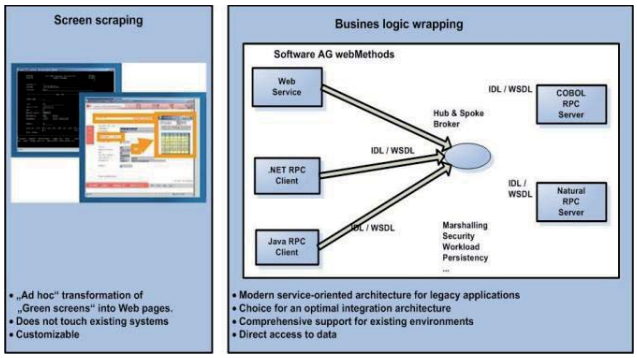


Figure 6. SOA enablement via screen scraping vs. Comprehensive mainframe integration based on services via business logic wrapping

Encapsulation of functionality from the SAG-Tours application in business services is shown in Figure 7. It shows the detailed flow of the business process for deleting a cruise, which is the result of the three-step integration approach (see Section V). It is very easy to identify the elementary tasks (such as finding and deleting of single business objects), which had been implemented as Natural programs in the SAG-Tours application. These programs will now be wrapped as business services so that the whole flow of the business process may be described as a composition of such business services.

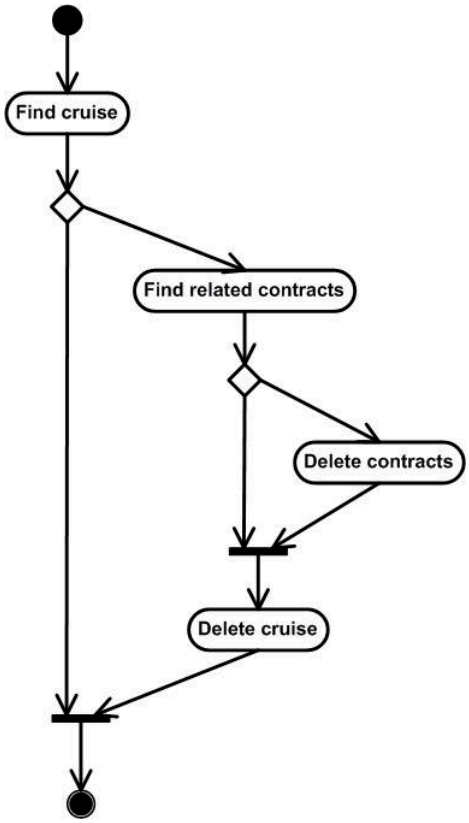


Figure 7. Sample business process 'Delete journey' modeled as activity diagram showing the integration of several existing business services

In the case study, the modeled business processes were translated manually into executable code on the ESB since there were only rather few processes. However, in the real-world scenarios (even in medium-size legacy modernization projects) an automated generation tool fitting for a particular technology required by the ESB should be used.

B. Business Process Model

In the case study, we started to set up a business process model of the domain. Since we chose a combination of top-down and bottom-up approaches to integrate the SAG-Tours application into a SOA, at first the optimal target processes had to be identified (top-down part). Since there have been quite a large number of processes, in order to check that the approach would also hold for larger legacy modernization projects, we grouped the identified processes into packages. The packages have to be formed based on domain-specific criteria; in the case study packages could be formed based on the domain entities, on which the processes primarily operated. Figure 8 shows the package model of processes. After identifying all processes and assigning them to packages, it was also possible to define dependencies between packages based on the underlying dependencies of the processes. This yields another helpful structuring of the whole set of processes and is also shown in Figure 8.

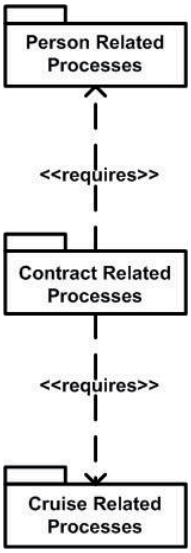


Figure 8. Packages for business process services and business services for structuring of the process model domain

After identifying the target processes, we identified the functionality of the SAG-Tours application that would be required in the processes and thus had to be modeled as business services (bottom-up part) in order to compose the processes in the final step. Identifying the legacy functionality and assigning it to business services had been a rather easy task. Most of the functional components could be directly derived from the existing Natural programs of the SAG-Tours application. However, some of the functional components had to be implemented anew in Natural (based on the Adabas database) in order to achieve a technologically homogeneous implementation of the foundation. For example, there was a Natural program DRINF-N0, which computed and returned all available cruises for a given start date and start harbor. This was encapsulated within the business service **BS_FindCruise** and could subsequently be used in different business processes.

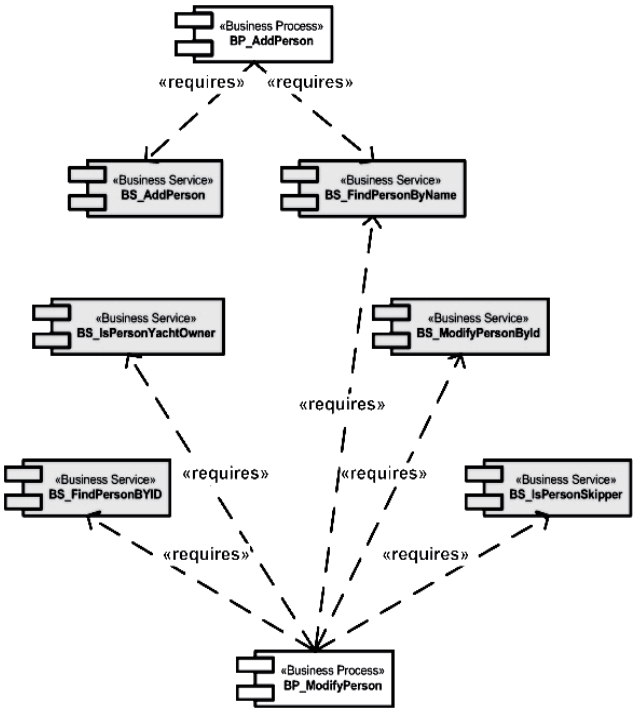


Figure 9. Business process model for person related processes and services

In the third and final step, the business services obtained in the second step could be composed to detail the business processes defined in the first step. For example, Figure 9 shows a part of the obtained business process model, which contains the processes and services related to person entities. The business process **BP_AddPerson** to add a new person in any role to the SAG-Tours application is, e.g., composed of the services **BS_AddPerson** and **BS_FindPersonByName**, which directly correspond to the functional components of the application.

As expected, most of the identified business services could be used in several different business processes. This can already be concluded from Figures 9 and 10, even though only part of the corresponding package models are shown there. For example, the business process **BP_ModifyPerson** to change information about an existing person in the Adabas database is composed of the services to find and modify a person by his or her ID or name as well as the services to obtain further information about the roles of the person.

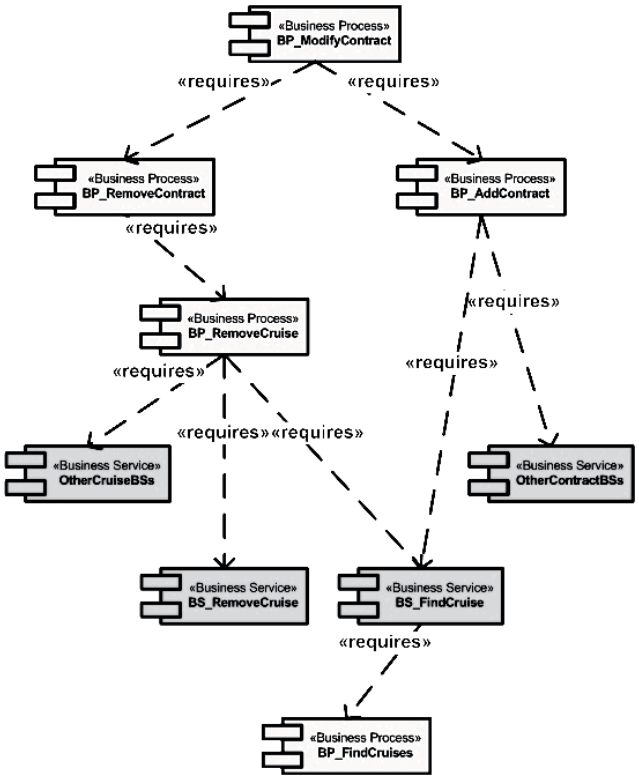


Figure 10. (Part of) the business process model for cruise and contract related processes and services

It should be noted that reuse is possible not only on the level of business services. As shown in Figure 10, there is also potential for reusing complete business processes within other business processes. For example, the business process BP_ModifyContract, which is executed to change any aspect of a booking by a person for a specific cruise, can be composed of the business processes to add and remove contracts, which in turn are composed of the business process to remove a cruise among others. Consequently, we can see that reuse on both levels is greatly simplified by the integration approach. Of course, business services may not be composed of business processes internally due to the definition of the according layers.

VI. INTEGRATION ARCHITECTURE AND STEPS

In this section, we will describe technical steps we have taken to bring up an integration architecture for the SAG-Tours application. At first, we will describe a general integration architecture, which is then mapped to a concrete integration tool suite. This is followed by an overview and more detailed technical steps one has to follow to use this integration architecture.

A. Integration Architecture

Since in the case study we followed the comprehensive mainframe integration approach (see Figure 6) that fits into a SOA as well, an N-tier integration architecture [2][7][13] was the appropriate means of choice. In this general integration architecture (see Figure 11), an end user uses a Web browser, which acts as a client front-end to a Web server. The Web server hosts presentation preparation logic, which (in a 'traditional' Web technology environment) prepares HTML pages for the end user's GUI and uses HTTP to interact with the end user's Web browser. On the other side, this logic accepts a service access protocol (e.g., SOAP over HTTP in the case of Web services) to access integrated services. In this case, an encapsulated DBMS is accessed, again, by means of some service access protocol, say, an ordinary remote procedure call (RPC).

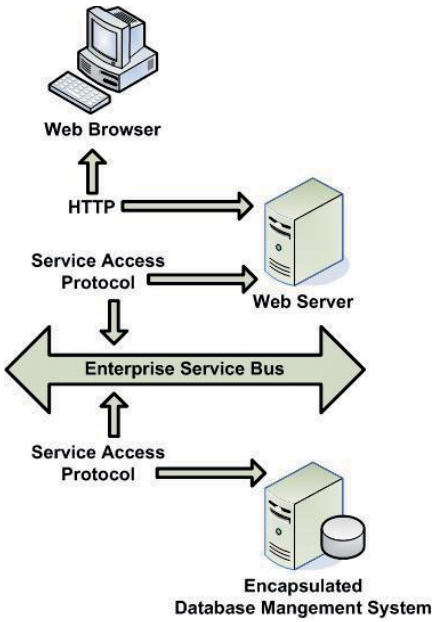


Figure 11. General N-tier system architecture using Web technology and an enterprise service bus

Since the integration technology stack was pre-defined for the SAG-Tours project, integration architecture for the SAG-Tours application was based on Software AG's integration tools such as EntireX Broker (see Figure 12).

At the lowest level of the integration architecture, there is a persistence layer with an Adabas DBMS. Above it, there is an application layer with a Natural runtime engine, a Natural RPC server (which calls that engine), a Software AG's EntireX Broker (which acts as an ESB that 'understands' different service access protocols) and – optionally – a so called integration server (which actually is an execution engine for a specialized business process execution language). At the highest level, there is a server-side Web presentation layer (also called GUI layer). Here the Apache Web server and the Servlet engine Tomcat are used. Like in the general integration architecture in Figure 11, a typical Web browser is used for the end user client access.

The integration can take place in three layers:

- Persistence layer. Here calls to the legacy database (e.g., Adabas) are replaced with Web services that issue the same native calls and return the requested data. These calls may be further modernized by

allowing SQL calls to be issued instead, even though data is still stored in the legacy database.

- Application layer. Here calls to the legacy procedures and programs (e.g., Natural subprograms and programs, respectively) are replaced with Web services that issue the same calls. The legacy procedures and programs that are called by the legacy application may have been written as reusable components and are candidates for reusable services.
- Presentation layer. Here 'green screens' are replaced with Web services that drive the legacy application the same way the original screens did. 'Green screens' are good candidates for the integration because many legacy applications use them to drive a single transaction; e.g., deleting a journey, adding a person / contract, etc. The integration in this layer often involves screen scrapping.

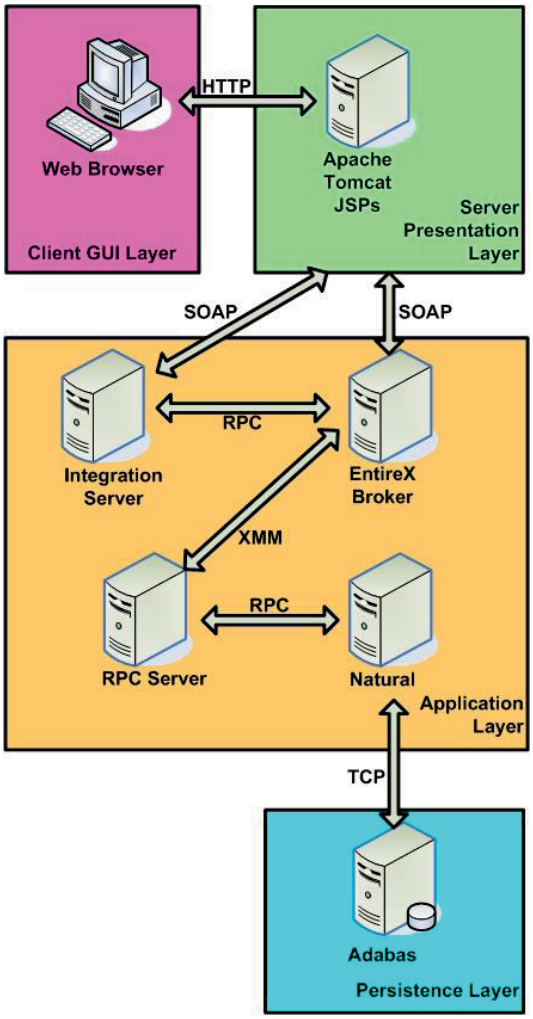


Figure 12. 3-tier system architecture that supports integration of SAG-Tours application into SOA and uses Software AG's integration tools

As shown in Figure 12, in the case study the integration took place in the application layer. This was feasible because the source code of the SAG-Tours application was available for us.

B. *Integration Steps*

Based on the components in Figure 12, the following technical integration steps have to take place:

1. Import Adabas database structures into a repository.
2. Map / import Adabas database structures for Natural subprograms.
3. Re-use existing Natural subprograms if possible. Otherwise, write suitable new ones based on the business process analysis from the previous section.
4. Define Natural subprograms to be accessible via the Natural RPC server. This server in turn is called by the EntireX ESB (also known as EntireX Broker).
5. Generate Web services stubs (here Java-based) for imported subprograms, thus exporting those stubs as Web service definitions from EntireX Broker.
6. Access those Web services from Java programs using JavaServer pages, e.g., via Axis / JAX-RPC.
7. Send the results from the JavaServer pages to the end user's Web browser.

The components as well as their usage within those integration steps are described below in more detail.

C. *Steps 1-2: Accessing Adabas from Natural via RPC*

Following the above steps, initially the existing Adabas database needs to be accessed by Natural programs. For existing Natural programs (which are re-used directly), there is no extra work just because the SAG-Tours application does this already. For new or re-written Natural programs, however, the existing Adabas database structures, which they want to access, need to be imported into a repository from the Natural tool suite.

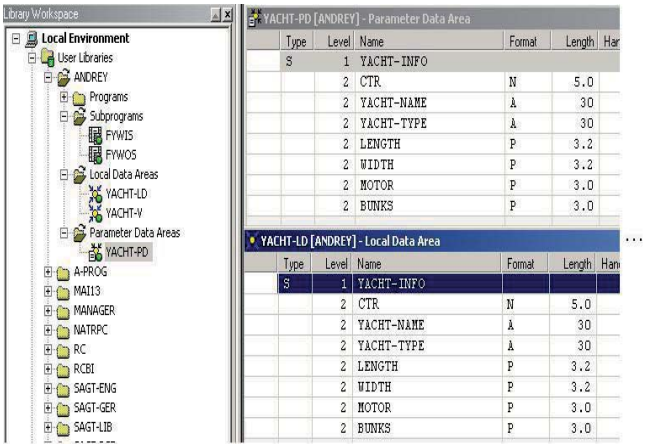


Figure 13. Data structure definitions

Natural programs and subprograms can query the repository to define their database access data structure. Such data structures are so called DEFINE DATA areas in Natural. Within such data areas, local Natural program variables are defined. Moreover, views to an underlying

database are defined by means of the USING clause within a DEFINE DATA area definition. Such a view serves as a common storage area between the Adabas DBMS and the calling Natural program. (It should be noted that Natural is not restricted to Adabas. Rather, it can also be used with other DBMSs like DB/2 and Oracle.)

Figure 13 shows a screen shot of the Adabas data mapping development environment from the Natural tool suite. As can be seen, Natural data types are defined for local and parameter data areas. These data areas have two purposes. First, they are required for the Natural subprogram itself as the communication structure with Adabas. This is done by the LOCAL DATA area (YACHT-V and YACHT-LD in Figure 13). Second, they specify the output data from the Natural subprogram in the PARAMETER DATA area (YACHT-INFO within YACHT-PD). This is then used as the mapping input by the EntireX Broker (see Section D below for more details).

The following little excerpt shows some Natural subprogram code, which fills a data area called 'YACHT-VIEW' using a FIND statement that searches for 'HUGE' yachts. This statement accesses the Adabas database from the SAG-Tours application.

```
DEFINE DATA
PARAMETER USING YACHT-PD
LOCAL USING YACHT-V
      1 COUNTER (N5.0) INIT <0> 1
      1 LIMIT (N5.0) INIT <0>
END-DEFINE

FIND ALL YACHT-VIEW WITH
      YACHT-VIEW.YACHT-TYPE = 'HUGE'
      . . .
      [code to check and increment COUNTER
      and to expand the size of the output
      data structure YACHT-INFO if required]
      . . .
      MOVE YACHT-VIEW.YACHT-NAME TO
      YACHT-INFO.YACHT-NAME(COUNTER)
      . . .
END-FIND

MOVE COUNTER TO YACHT-INFO.COUNTER
END
```

The query result in the 'YACHT-VIEW' data area is then moved to the output PARAMETER DATA 'YACHT-PD / YACHT-INFO' area. From this area, a 'YACHT-INFO' data structure in Figure 13 is filled. The 'YACHT-INFO.COUNTER' variable is filled with the number of tuples, which were returned from the FIND statement. Eventually, the newly filled YACHT-INFO data structure (conceptually similar to a 2-dimensional array, technically several 1-dimensional arrays) is returned as the Natural subprograms result.

D. Steps 3-4: Accessing Natural via RPC

Following the above steps, the Adabas database can be accessed by means of Natural programs. This is pretty much the way, the SAG-Tours application works. Now in order for this application to be re-used as services within the application modernization context of the SAG-Tours project, those programs need to become technically accessible from the outside. For this purpose, the (Natural) remote procedure call (RPC server in Figure 12 is used. Natural subprograms (which run on remote machines) are accessed via a RPC. This is conceptually comparable to protocols such as Java Database Connectivity (JDBC), which are frequently used for Java-based remote access to relational databases.

Within the integration architecture in Figure 12, the EntireX Broker actually uses a RPC for such remote Natural access. The EntireX Broker is Software AG's integration turn table, which thus conceptually serves as the core of an ESB as it is known from a SOA-based integration architecture (see [7][13] for more detail).

Two major pre-requisites for this approach exist. First, he Natural programs need to be callable as subprograms. This just requires a well-written Natural subprogram with clearly defined input / output data areas. Another option is to have Natural programs as a base, which can reasonably easy be modified to fulfill this requirement. However, one of those options is due to Natural coding practices not to seldom given for existing Natural code and it does hold for the SAG-Tours application.

Second, the semantic structure of the existing subprograms must be 'good enough' to be re-usable in a modernized business context. To ensure this, we did the business process analysis of the existing Natural programs as described in the previous section.

Since most of the existing programs were easily understandable, e.g., comparable to functions like 'DELETE JOURNEY' or 'FIND-AVAILABLE-YACHTS', this was a manageable task for us (see [16] for more details). However, such an analysis might not be an easy task for more complex existing Natural code.

It should be noted that as a 'side effect', all the students were able to read and write Natural code afterwards (even though they had no previous experience in Natural coding).

E. Step 5: Re-using integrated legacy code as Web services

Having integrated the Natural subprograms using EntireX Broker, one now wants to re-use them within non-Natural contexts. In the case of the SAG-Tours project, Web services are the means of choice. Thus, a Web Services Description Language (WSDL) based service interface definition and SOAP access to the integrated Natural 'services' needs to be enabled.

Easily enough, the EntireX Broker development environment can generate all the required code. It utilizes the PARAMETER DATA-areas form the above steps to enable a mapping specification from the Natural procedure parameters to XML data types. Of course, this requires a suitable PARAMETER DATA areas for each Natural

subprogram (either newly written or existing), which is to be mapped to another service description.

The mapping itself is based on a XML mapping specification (XMM), which is used by EntireX Broker at runtime for data type conversions (see XMM in Figure 12).

Now that the task of technically integrating the existing Natural subprograms is achieved utilizing the EntireX Broker, those programs can directly be accessed as Web services based on SOAP. EntireX Broker does the internal mapping and provides a WSDL based service endpoint for each of the exposed 'Natural'-Web services.

F. Step 6-7: Using Web services from JSP clients

As shown in Figure 12, we then just used Java code within JavaServer Pages (JSPs), which in turn called the Web services above.

Instead of hard-coding JSPs and Java code, the Web services could also be called using Software AG's business process engine. This component in Figure 12 is called integration server. We tried this exemplary as well – it worked fine except for some minor data type issues. But we did not explore it in more depth due to given time limits – the SAG-Tours project was limited to one year.

Eventually the JSPs allowed for an easily developed GUI front-end for the end user. In the real-world scenario, there is also the possibility to call the Web services from other external programs. Since our legacy components are completely enclosed as Web services, they can easily be embedded in a larger SOA environment.

VII. LESSONS LEARNED

Looking back at the SAG-Tours project, we can derive a good bit of experiences, which might also be valuable for other legacy modernization projects. These experiences are described below as learned lessons (both technical and pedagogical).

A. Technical Lessons

From a technical point of view, the most interesting insight gained by the SAG-Tours project has been related to mainframe legacy software. In particular, it was interesting to see how easy or difficult it was to integrate a legacy mainframe application into modern software architecture and how object-oriented programmers could cope with this task. (The students involved in the SAG-Tours project were reasonably experienced in Java coding.)

1) Complexity of integration:

- *Integration effort*: The integration of existing legacy mainframe applications into a SOA is not too hard any more. Whereas this may have been true a couple of years ago, the SAG-Tours project showed that the integration is rather simple. This has been proven in the case study where final-year bachelor students succeeded to carry out the project. The students had no previous experience in mainframes and the technology used in the SAG-Tours application.
- *Tool support*: The integration of Adabas / Natural legacy applications is very well supported by Software AG's integration tools. Since Software AG's integration technology stack was given us as a pre-requisite, a general conclusion on tool support cannot be drawn. It will be interesting to evaluate this aspect in follow-up projects; i.e., whether integration tools provided by different vendors can also be used to implement our integration approach.
- *Effort dependencies*: As expected, the exact effort required depends heavily on the size and number of the target processes and (even more) on the amount of knowledge of and documentation on the existing code in the legacy application itself.

2) Integration Approach:

- Regarding the conceptual integration approach to the legacy modernization, we used a combination of top-down and bottom-up approaches. However, this may not be viable in all situations. Factors that have to be taken into consideration in order to choose the right approach include:
  - *Quality of Service (QoS)*: the screen scraping approach can never yield better QoS than the original application at best. This time, the comprehensive mainframe integration approach might open up possibilities for improving the QoS externally.
  - *Time to market requirements*: if it is important to have the legacy application usable in a service environment as fast as possible, regardless of the technology used, the bottom-up approach will be a better choice.
  - *Effort (time and money)*: In most cases, comprehensive mainframe integration (which is based on services) will be more costly from a short project point of view because of the conceptual complexity. This effort could, on the other hand, well pay off in the long run because such integration has the potential to increase re-using of components and may simplify software maintenance.
  - *Knowledge about existing code*: if there is only a black-box like knowledge of what the existing application components do and not how they do it in detail, then the comprehensive mainframe integration approach may not be feasible at all. This is especially true if existing components have to be modified or extended for the integration (as in the case study). This scenario which seems unrealistic at first sight can actually be found in many organizations nowadays.
- In summary, we feel that our combined integration approach is ideal for many situations because it joins the long-term potential of the top-down approach with the technological ease of the bottom-up approach.

B. Pedagogical Lessons

From a pedagogical point of view, there are several lessons learned as well, which are interesting from higher computer science education perspective. Within the SAG-Tours project the following conclusions were derived:

1) Usage of complex technologies in bachelor projects

- *Possible project type*: A technology environment can involve different technical skills such as different programming languages, several tools, and different hardware and networking technologies, as well as conceptual knowledge in software / system architectures, project management techniques, etc. The usage of such a complex environment is still feasible, although certainly not an easy project for students.
- *Enhanced motivation*: Not only was the SAG-Tours project feasible, but also the students were highly motivated because they felt the 'real world' characteristics of the project. During a visit to Software AG, they were shown recent integration software improvements and a roadmap, and they got a guided tour including visiting Software AG's real IBM mainframes. All these things enhanced student motivation very much. At the end of the SAG-Tours project, the students highly ranked the value of the project for their computer science education.
- *Team dependency*: Although legacy modernization projects are doable by students, at least reasonable motivated students are required, who want to dig into the game. The skills of students involved in the SAG-Tours project have been 'average' to 'above average'. A project team of 'below average' students would likely not have finished the project with this success. As said before, the students had a good knowledge of Java, and database and networking technologies but no prior exposure to Web services, integration software or even mainframe technology.
- *Reasonable environment*: As known from software projects in general, a proper organizational framework is required. This includes a project room with dedicated project team time and at least a drawing board, dedicated machines, occasional pizzas, a project poster to show etc.

2) SOA / Mainframe technology

- *SOA principles / Web services*: As expected usage of Web services and SOA principles such as service-oriented design, process analysis, well-defined interfaces, etc., resulted in a reasonably better 'interface-oriented' software and system design.
- *'Interesting mainframes'*: Mainframe technology in general was seen as a quite interesting topic, especially when the students recognized that many topics such as transaction processing have been around for quite a while in computer science history.
- *Willingness to 'struggle through it':* Especially the interplay of all the technologies and a mostly new terminology for (although partially known) concepts did require from the students some willingness to 'struggle through it'. This clearly demanded student motivation. But once that motivation had been raised, the SAG-Tours project clearly became a self-runner for the supervising professors.

VIII. Conclusion and Future Work

There is a common belief that SAO enablement on the application layer (also known as business logic wrapping) is a very expensive approach to modernization of a legacy application in case of a lack of legacy skills because of huge efforts required to understand the legacy application. However, the case study showed that this approach can require rather few efforts if the right technologies are used.

In other words, integration of legacy applications into a SOA is neither impossible nor too complex. Simple evidence of this fact is that in the scope of the SAG-Tours project, final-year bachelor students were able to do this within a year, with an average effort of one day per week for 10 students. The students were experienced in Java coding and network technologies in general. But they had no previous experience in Natural coding, mainframe technologies (Adabas in particular) or integration of legacy applications.

However, for a full SOA, we have to add some more components to complete the integration architecture in Figure 12. But the fact that it was possible without any major problems in the context of the SAG-Tours project carried out by bachelor students [8] shows that it is not necessary to have a disproportional knowledge or to make huge efforts for such integration.

Whether a particular integration would be possible in a heterogeneous system environment (e.g., without using newly offered components from the same vendor) or what efforts would be required could be evaluated in future work within another legacy modernization project.

Acknowledgments

We would like to especially thank our students [16] from the SAG-Tours project, who were instrumental in implementing the above concepts.

Irina Astrova's work was supported by the Estonian Centre of Excellence in Computer Science (EXCS) funded mainly by the European Regional Development Fund (ERDF).

References

[1] A. Koschel, C. Kleiner, and I. Astrova. Mainframe application modernization based on service-oriented architecture: a practical industry cooperation case study. Proceedings of IARIA Computation World: Future Computing, Service Computation, Cognitive, Content, Patterns, ComputationWorld 2009, 298 – 301, 2009.

[2] S. Conrad, W. Hasselbring, A. Koschel, and R. Tritsch. Enterprise application integration: Grundlagen - Konzepte - Entwurfsmuster – Praxisbeispiele. Spektrum Akademischer Verlag, 2005.

[3] C. Date, An introduction to database systems 5th Edition, Volume I, 1992.

[4] A. Erradi, S. Anand, and N. Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. In IEEE SCC, pages 257–260. IEEE Computer Society, 2006.

[5] S. Englet. Wiederverwendung von Legacy Systemen durch einen bottom up Ansatz bei der Entwicklung einer SOA. In H. Hegering, A. Lehmann, H. Ohlbach, and C. Scheideler, editors, GI Jahrestagung (1), volume 133 of LNI, pages 96–100. GI, 2008.

[6] W. Keller. Enterprise Application Integration. Erfahrungen aus der Praxis. Dpunkt-Verlag, 2002.

[7] D. Krafzig, K. Banke, and D. Slama. Enterprise SOA: Service-oriented architecture best practices. Prentice Hall, 2005.

[8] C. Kleiner and A. Koschel: Legacy vs. Cutting edge technology in capstone projects: What works better? Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE09), 2009.

[9] G. Lewis, E. Morris, and D. Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering, pages 15–23,Washington, DC, USA, 2006. IEEE Computer Society.

[10] G. Lewis, E. Morris, D. Smith, and L. O'Brien. Serviceoriented migration and reuse technique (smart). In STEP '05: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice, pages 222–229, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Software AG, SAG-Tours: SOA integration projects, Application Document (Technical Report), Darmstadt, 2007.

[12] D. Smith. Migration of legacy assets to service-oriented architecture environments. In ICSE COMPANION '07: Companion to the proceedings of the 29th International Conference on Software Engineering, pages 174-175,Washington, DC, USA, 2007. IEEE Computer Society.

[13] G. Starke and S. Tilkov (Edts.). SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen. Dpunkt-Verlag, 2007.

[14] T. Erl. SOA Design Patterns, Prentice Hall, 2009.

[15] G. Lewis, E. Morris, and D. Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering, pages 15–23,Washington, DC, USA, 2006. IEEE Computer Society.

[16] A. Koschel, C. Kleiner, M. Safris, A. Budina, O. Efimov, A. Morozov, W. Schaefer, D. Schaefer, S. Kirstein, W. Zlobin, A. Kolesnikov, and A. Brockwitz. Abschlussdokumentation für das Bachelor-Projekt 'SAG Tours'. FH Hannover, Fakultät IV. 2008.

[17] G. Canfora, A. Fasolino, G. Frattolillo, and P. Tramontana. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. Journal of Systems and Software, 81(4):463–480, 2008.

[18] H. Sneed. Integrating legacy software into a service-oriented architecture. CSMR'06, IEEE CSP, 2006.

[19] SAP. NetWeaver Open Integration Platform. https://www.sdn.sap.com/irj/sdn/developerareas/netweaver Last access: June 2010.

[20] K. Channabasavaiah and K. Holley. Migrating to a service-oriented architecture. IBM White paper, 2004.

[21] Oracle. Oracle IT modernization series: The types of modernization. Oracle White paper, 2006.

[22] F. Mohammed. Oracle SOA Suite. Sys-Con XML Journal, 2007.

[23] Sofware AG. webMethods ApplinX. https://www.softwareag.com/ApplinX Last access: June 2010.

[24] Microsoft Corporation. Enabling real-world SOA through the Microsoft Platform. 2006.

[25] J. Ziemann, K. Leyking, T. Kahl, and W. Dirk. Enterprise model driven migration from legacy to SOA. Software Reengineering and Services Workshop, 2006.

[26] A. Erradi, S. Anand, and N. Kulkarni. Evaluation of strategies for integrating legacy applications as services in a service oriented architecture. In IEEE SCC, pages 257–260. IEEE Computer Society, 2006.

[27] T. Suganuma, T. Yasue, T. Onodera, and T. Nakatani. Performance pitfalls in large-scale java applications translated from cobol. In OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object oriented programming systems languages and applications, pages 685–696, New York, NY, USA, 2008. ACM.

[28] Oracle. Oracle IT modernization series: Approaches to IT modernization. Oracle White paper, 2009.

[29] T. Laszewski. SOA and the mainframe: Two worlds collide and integrate. http://www.theserviceside.com/tt/articles/content/SOAandtheMainframe/article.html, 2009 Last access: June 2010.

Human-Computer Interaction Design Patterns: Structure, Methods, and Tools

Christian Kruschitz
Department of Informatics-Systems
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
chris@isys.uni-klu.ac.at

Martin Hitz
Department of Informatics-Systems
University of Klagenfurt
Klagenfurt am Wörthersee, Austria
hitz@isys.uni-klu.ac.at

***Abstract*—Design patterns play an important role when managing design knowledge for later reuse. In the Human-Computer Interaction (HCI) community, design patterns are an often used tool for sharing design knowledge among user interface (UI) designers as well as non UI experts. An HCI design pattern consists of several different components. The first component is the *structure* of a pattern, which encapsulates the description of the problem, its context, and the solution suggested by the pattern. *Relationships* and *semantics* are important when design patterns are used in *pattern management tools*. To make sure that the developed patterns satisfy their users, it is important to *evaluate* and *validate* the patterns' content.**

Keywords – HCI patterns, History, Organization, Evaluation, Validation, Standardization .

I. INTRODUCTION

This paper is an extended version of [41] (PATTERNS09), and gives an in-depth overview on the literature and research on Human-Computer Interaction **(**HCI) design patterns.

HCI design patterns are an important tool for knowledge sharing in the domain of Human-Computer Interaction. To avoid reinventing the wheel again and again, design patterns identify and document best practice solutions to support user interface designers in their daily work in order to improve their productivity and make the design process more efficient.

Over the past years, many research activities in the area of design patterns have aimed to make them easier to use. The research focused on the pattern structure, organizing principles, semantics, relationships, evaluation of the usefulness of patterns, and tool support. This paper gives an overview of the above-mentioned topics.

We start with a historical overview of design patterns from the birth of the pattern concept to today's activities in the community. In Section III, we provide definitions of relevant terms. Section IV deals with the pattern structures from the early beginnings in architectural design to pattern forms, which are currently used by the HCI design pattern community.

The following sections deal with research topics on design patterns, starting with the *Organizing Principles* which are focusing on the categorization schemes of design patterns for easier retrieval of the right pattern for a given design problem within an collection or pattern language. Section VI shows how to identify relationships among design patterns. Relationships represent a key concept to gain the full reuse potential from individual patterns. Proper consideration of relationships promises even more powerful search and navigation opportunities. Section VII describes research approaches on how to enrich design patterns with semantic information. By using ontologies, it is possible to share HCI design patterns across different collections and it is easier to identify patterns for a specific design problem. When using patterns in interface design it is important that the used pattern is valid for the problem to be solved. Therefore, Section VIII presents some approaches of how to evaluate and validate HCI design patterns. Section IX introduces some software tools, which have been developed in the past years. The last section deals with standardization approaches.

II. HISTORY

Christopher Alexander, architect and mathematician, first talked about patterns in his PhD thesis which was subsequently published as the book "Notes on the Synthesis of Form" in 1964 (see timeline in Figure 1). Christopher Alexander laid the cornerstone of the later well known concept of design patterns. Alexander argues that design problems are getting more and more complex so that they exceed the designer's abilities to come up with a solution from scratch. Furthermore, problems cannot show its own solution, only a set of requirements, when combined together, the requirements create a new idea [4]. From 1975 to 1979 Alexander published several books on the concept of design patterns and pattern languages [3][5][6]. Although his concept was originally meant to support reuse of architectural design knowledge, it found its way into the HCI community where it was first mentioned 1986 by Donald Norman and Stephen Draper [51].

Ward Cunningham and Kent Beck have adopted this principle to object-oriented programming (OOP) and user interface (UI) implementation in 1987 [8][55]. They presented five patterns for designing window-based user interfaces in Smalltalk:

- WindowPerTask
- FewPanes
- StandardPanes

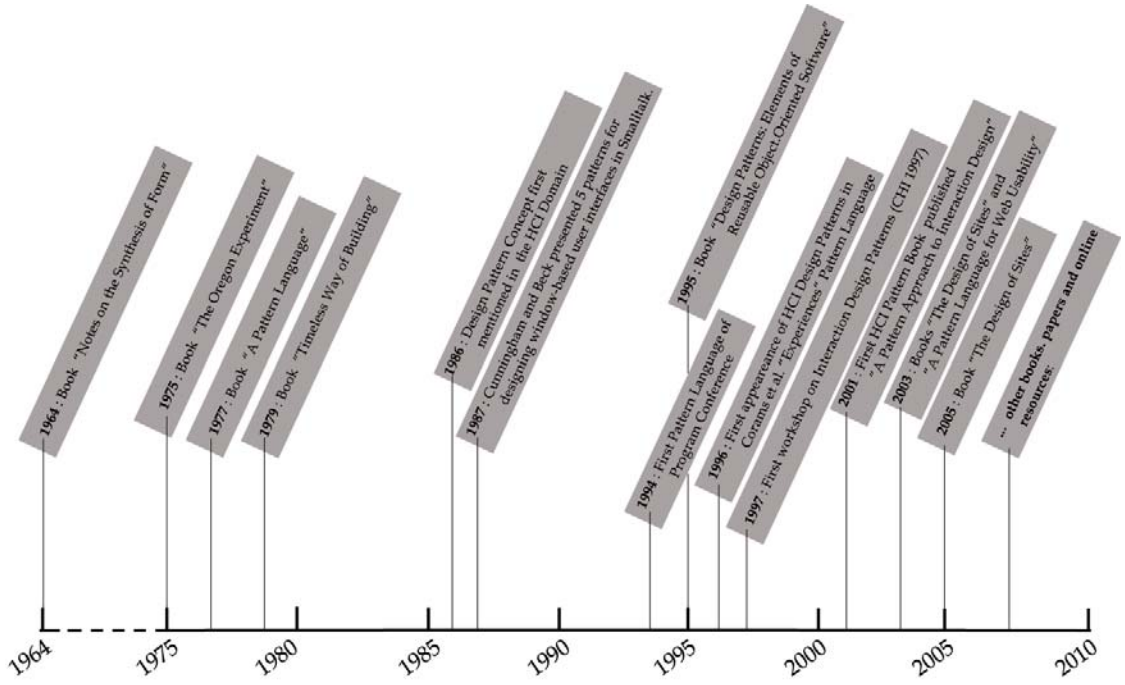


Figure 1: HCI Design Patterns Main Activities Timeline

- NounsAndVerbs
- ShortMenus

This small "pattern language" was intended to give novices in Smalltalk the possibility to use the language with all its strengths and avoid pitfalls. Cunningham and Beck were surprised of the good interfaces their users designed.

The Hillside Group, which now sponsors pattern conferences all over the world, has organized the first PLoP (Pattern Languages of Programs) conference in October 1994 with 80 participants.

Design patterns made their breakthrough in the software engineering community when Erich Gamma et al. published one of the bestselling books in software engineering, "Design Patterns: Elements of Reusable Object-Oriented Software" [25]. This book was awarded in the Journal of Object Oriented Programming (September 1995 Issue) "the best OO book of 1995" and "the best OO book of all times". From this time on many design patterns and pattern languages in software engineering as well as in user interface engineering have been published.

In HCI, the actual start of the design pattern era was 1996 when Tod Coram and Jim Lee published the first design patterns of a pattern language for user-centered interface design [14]. Their intention was to provide high level patterns with which user interface designers could build graphical user interfaces which are pleasurable and productive to use. In 1997 the first CHI workshop on pattern languages in user interface design was organized. The participants explored the use of a pattern language in user interface design to make HCI knowledge reusable in different applications [7]. At this time, user interface toolkits have emerged to support user interface designers and software engineers. However, the workshop participants stressed that a more general description of user interface design know-how, which is detached from a specific implementation platform, would be desirable and agreed that design patterns could be an appropriate tool. Design patterns reside on a higher level of abstraction than UI toolkits and are not bound to source-code for a specific implementation of the addressed problem. Furthermore, patterns are written in such a general way that they give pattern users the possibility to decide how specific widgets should be arranged to concretize the patterns' solutions.

Other pattern workshops focusing on HCI design patterns followed (see Table 1). Beside the discussion about the concept of design patterns in the HCI domain in workshops around the world, several books were published addressing design patterns and pattern languages for the HCI domain:

- "A Pattern Approach to Interactive Design" provides design patterns for interactive exhibits and user interface design [11].
- "The Design of Sites" is a comprehensive pattern language to help developing customer-centered websites [18].
- "A Pattern Language of Web Usability" provides a pattern language for the design of usable websites [27].
- "Designing Interfaces" is a collection of HCI design patterns which addresses how to build desktop and mobile user interfaces [60].
- "Patterns for Computer-Mediated Interaction" provides patterns for the design of Human-Computer-Human Interaction (HCHI) [56].

- "Designing Social Interfaces" written by the curator of the Yahoo!Design Pattern Library. This book provides patterns for designing a usable social website [15].
- "User-Centered Interaction Design Patterns for Interactive Digital Television Applications" shows how television applications can be designed based on design patterns [42].

Conference	Year	Title	Ref.
INTERACT	1999	Usability Pattern Languages	[28]
ChiliPLoP	1999	CHI Meets PLoP	[9]
CHI	2000	Pattern Languages for Interaction Design: Building Momentum	[24]
CHI	2001	Patterns: What's in it for HCI (Panel)	[10]
CHI	2003	Perspectives on HCI Patterns	[22]

Table 1: Workshops on HCI Design Patterns

Beside the publication of books, the World Wide Web is the perfect medium to disseminate and publish HCI design patterns across the HCI community. Over time, many repositories were published, some have disappeared and others are still a point of reference to UI designers. Some of them are listed below with a short comment on each of them. For a more detailed description of a selected set of the mentioned Web repositories, cf. Section VIII.

- "Common Ground" is Jennifer Tidwell's pattern language for HCI design [59].
- "Designing Interfaces" is the companion website to the same named book [60].
- "Little Springs Design – Mobile UI Design Resources" provides a design pattern collection for designing UIs for mobile devices [45].
- "UI Patterns – User Interface Design Pattern Library", describes design patterns for desktop and mobile phone UI design [61].
- "Yahoo!Design Pattern Library" is a very popular design pattern collection by Yahoo! [66].
- "Welie.com – Patterns in Interaction Design" is a huge design pattern repository which addresses patterns for Desktop- and Webdesign [64].
- "Portland Pattern Repository" maybe the oldest pattern repository [53].

Beside the above mentioned design pattern repositories there exists several design pattern portals providing a collection of references to design pattern resources. These are:

- "The Interaction Design Patterns Page", a collection of links to interaction design pattern resources [19].
- "hcipatterns.org", provides information to HCI design patterns web resources, books and other related stuff like papers [29].
- "The Pattern Gallery", a listing of design pattern forms with a short statement [23].
- "The Hillside Group", is the organization who organizes the PLoP conferences. A good resource to start with the design pattern concept [58].

In the following section, we define the terms *HCI design pattern*, *design pattern catalogue/collection*, and *pattern languages*. Furthermore, we describe the components of HCI design patterns (see Figure 2), and describe the developments of the last years.

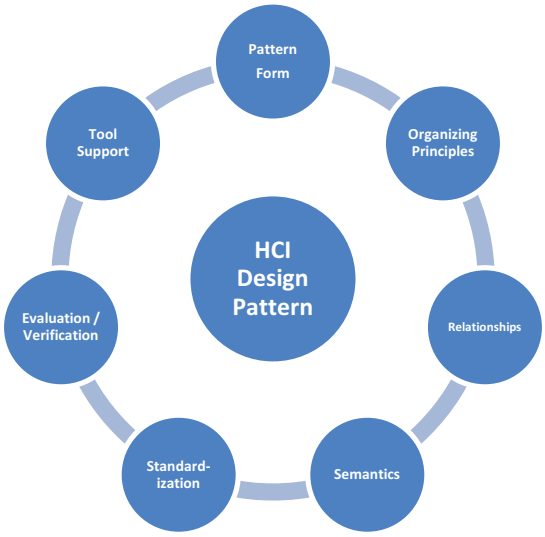


Figure 2: Components of HCI Design Patterns

III. DEFINITIONS

A. HCI Design Pattern

An HCI design pattern describes a recurring problem together with a proven solution. An HCI design pattern, in the following referred to as "pattern" or "design pattern", has a well-defined form, which is dependent on the individual author's preferences. A pattern form should be used consistently across a pattern language or pattern collection. This makes it easier for pattern users to understand the problem, context, and solution of a pattern throughout a pattern collection / language. The pattern itself, when it is part of a collection or a pattern language, may have references to other patterns.

B. Design Pattern Catalogue / Collection

Patterns are stored in design pattern catalogues or collections. The patterns in such a catalogue are categorized to support faster navigation within the repository. In this case, patterns show almost no relationships among each other and thus do not form a fully interconnected system. Instead several patterns stand more or less alone and have no or few connections to predecessor or successor patterns. Furthermore, such a collection usually does not completely cover a specific application domain.

C. Pattern Language

In contrast to a pattern catalogue / collection, a pattern language is a *complete* set of patterns for a given family of

design problems in a given domain. A pattern language describes problems by means of high-level design patterns, which may be solved by lower-level design patterns. The design patterns are connected through relationships, so that they constitute a network.

In a pattern language, the "words" are the patterns, while the connections between patterns represent the "rules of grammar" which are situated in the pattern itself. When words and rules of grammar are combined, a "sentence" is generated. Sentences can be built in many different forms when the rules of grammar are followed. So there is not only one path through a pattern language, it offers several possibilities to solve a design problem. A good example is "The Design of Sites" by van Duyne et al., a pattern language that allows designers to articulate an infinite variety of Web designs [18]. Figure 3 visualizes a part of a pattern language with focus on online shopping [62].

IV. PATTERN FORM

Patterns are written by researchers and UI designers in a well-defined format, the so-called pattern form. This form is dependent on the author's preferences but several canonical forms have been established in the history of design patterns. These are described below in more detail.

A. Alexandrian Form

Christopher Alexander has invented the concept of design patterns as a problem / solution pair and presented them in a common format [3], which consists of:

- **Picture:** Shows an archetypal example of the pattern in use.
- **Introductory Paragraph:** This sets the pattern in the context of other, larger scale patterns.
- **Headline:** A short description of the problem.
- **Body:** Detailed description of the problem.
- **Solution:** The solution of the pattern which is written as a design instruction.
- **Diagram:** Sketches the solution in the form of a diagram.
- **Closing Paragraph:** Gives references to other patterns and describes how this pattern relates to with other, smaller patterns.

This pattern form is used with minor changes by Todd Coram and Jim Lee [14], Jan Borchers [11], Ian Graham [27], Mark Irons [37], Douglas van Duyne et al. [18], and Eric Chung et al. [13].

B. Software Engineering Design Patterns

There are influential approaches stemming from the software engineering domain, which are briefly described below:

THE GANG OF FOUR FORM

This form is used in the book "Design Patterns: Elements of Reusable Object-Oriented Software" [25] and for many other OO software design patterns from different authors.

- **Pattern Name and Classification**: The pattern name describes in a word or two what the pattern is about and the classification groups the pattern with similar problems.
- **Intent**: A short statement what the pattern does and some words about its rationale and intent.
- **Also Known As**: Alternative names for the pattern.
- **Motivation**: A scenario how the class and object structures solve the addressed problem.
- **Applicability:** Describes the situation in which the design pattern can be applied.
- **Participants**: Addresses which classes and/or objects participate in the design pattern.
- **Collaborations**: Represents how the participants collaborate with each other.
- **Consequences**: Tells the user how the pattern supports its objectives.
- **Implementation**: Describes how to implement the pattern and how to overcome common pitfalls.
- **Sample Code**: Contains some code fragments on how to implement the pattern.
- **Known uses**: Examples of implementations proving the value of the pattern.
- **Related Patterns**: References to other patterns which are closely related.

THE PORTLAND FORM

The Portland Pattern Form [53] is not as clearly structured as the others. The patterns are structured as text paragraphs. Ward Cunningham describes the form he uses in the Portland Pattern Repository as follows:

> *"Each pattern in the Portland Form makes a statement that goes something like: 'such and so forces create this or that problem, therefore, build a thing-a-ma-jig to deal with them.' The pattern takes its name from the thing-a-ma-jig, the solution. Each pattern in the Portland Form also places itself and the forces that create it within the context of other forces, both stronger and weaker, and the solutions they require. A wise designer resolves the stronger forces first, then goes on to address weaker ones. Patterns capture this ordering by citing stronger and weaker patterns in opening and closing paragraphs."*

THE COPLIEN FORM

James Coplien used the so-called Canonical Form to describe his patterns. This form is also called *Coplien Form* because he was one of the more famous pattern writers in the early stages of the software patterns movement [1].

- **Name**: Describes the name of the pattern.
- **Problem**: Addresses which problem will be solved by the pattern.
- **Context**: Tells the user in which context the pattern can be applied.

- **Forces**: This element describes (possibly conflicting) requirements and their impact on the design pattern.
- **Solution**: Shows the user how to balance the forces and solve the problems.
- **Resulting Context**: States which context is generated by applying the pattern.
- **Rationale**: Describes why the solution is implemented in such a way.
- **Author**: Name of author and creation date.

Software engineers have adapted the Alexandrian Form to describe their patterns. Significant changes are the introduction of the content elements – *Implementation*, *Sample Code* and *Participant*s, which describe OO programming facets like source code and UML diagrams.

C. HCI Design Pattern Forms

UI PATTERN FORM

This pattern form was developed at the INTERACT patterns workshop in 1999 [28]. It comprises seven content elements:

- **Name**: Shortly describes the pattern's intent.
- **Sensitizing Example**: This component should sensitize the reader to the application of the pattern. It is usually a screenshot or drawing of the pattern's solution.
- **Problem Statement**: Describes the conflicts (trade-offs) between "forces" guiding the design approach.
- **Body:** Textual description of the pattern's intent.
- **Solution Statement**: Tells what to do (and not how to do it).
- **Technical Representation:** This example solution is more detailed and intended to inform HCI experts about the pattern's solution.
- **Related Patterns:** References to successor patterns which enhance or are similar to the pattern.

TIDWELL FORM

Jenifer Tidwell is using a very minimalistic form, which is used throughout her book "Designing Interfaces" and the accompanying website [60].

- **Name:** Describes the pattern's intention and defines a unique reference number.
- **Sensitizing Image:** This image sensitizes the reader to the pattern's solution.
- **What:** Short problem statement.
- **Use When:** Describes the context in which this pattern can be used.
- **Why:** Describes the design rationale.
- **How:** Represents the solution part of the pattern.
- **Examples:** Screenshots of the instantiated pattern with a short description.

HCI design pattern authors are not using content elements such as *Implementation* or *Source Code* for their patterns. It is not necessary to provide source code for demonstration purposes of the pattern's solution. The problem is that interaction principles are implemented in many different programming languages. Therefore, the pattern is written in a more abstract way than software patterns. The pattern form is significantly stronger based on the Alexandrian Form.

More pattern forms, which were recently used, can be found at Sally Fincher's portal [23].

V. ORGANIZING PRINCIPLES

Alexander has organized his pattern language into levels of physical scale. He starts with high-level patterns which describe the size and distribution of towns and proceeds in several steps to low-level patterns which describe individual rooms [3].

TASKS	INFORMATION INTERACTION
Retrieval	Retrieval tasks have (static) information passing from the artefact to the user(s). The flow is usually initiated by the user(s).
Monitoring	Monitoring tasks have (dynamic) information passing from the artefact to the user(s). The information may come from 'beyond' the artefact. The flow is usually initiated by the artefact.
Controlling	Controlling tasks have information passing from the artefact to the user(s) and a separate flow from the user(s) to the artefact. The flow may be initiated by either the user(s) – proactive control – or by the artefact – reactive control
Construction	Construction tasks have the user(s) putting new information into the artefact
Transaction	Transaction tasks have the user(s) putting linked changes into the artefact. They are often accompanied by a corresponding change in the outside world.
Modification	Modification tasks have user(s) changing information already in the system. They may be modifying 'attribute values' or 'structure'
"Calculation"	Calculation tasks have the user(s) putting information into the system which it then transforms and passes back to the users (not necessarily synchronous).
Workflow	Workflow tasks have the system providing information to the user(s) which they then transforms and passes back to the system (not necessarily synchronous).
Communication	Communication tasks have one group of users putting information into the system that it passes to another group of users.

Table2: Organizing Principle by Fincher and Windsor [21]

In analogy, an organizing principle for HCI patterns, as Fincher and Windsor mentioned, should allow users to find patterns they need within a large repository. An organizing principle should meet at least the following objectives (cited from [24]):

- ***Taxonomise*** *– It must allow finding and selecting material from a large repository.*
- ***Proximate*** *– It must allow users to locate supporting, perhaps inter-related, patterns applicable to their solution.*

- ***Evaluative*** *– The problem should be considered from different viewpoints. So that it is possible to evaluate and change the users approach or to confirm the quality of their existing solution.*
- ***Generative*** *– It would be advantageous to support users to consider the problem from different points of view and allow for building new solutions, which have not been previously considered.*

Fincher and Windsor have adapted the Alexandrian structure of scale to UI design, starting with a high-level category *Society*, and descending via *System*, *Application*, *UI Structure* and *Component* to the low-level categories *Primitive* and *Physical Detail*. However, they do not consider this categorization sufficient for UI designers to find a pattern for their problem. So they suggested a second and a third structure. The second one is based on the design-by-type-of-task, where they have defined tasks based on the information flow, which includes categorizations such as *Task-Retrieval*, -*Monitoring*, -*Controlling*, -*Construction* and others (see Table 2). The reason for the last categorization structure is as the authors stated in their article: "*It is as common, as 'natural', for UI designers to structure their design not around the nature of the interaction (the 'how'), but the stuff that is to be interacted with (the 'what')".* So they suggested another category to satisfy UI designers which comprises categories such as *Volume*, *Complexity*, *Structure,* and *Dynamics*. *Structure* is further subdivided into *amorphous*, *sequential*, *hierarchical*, *directed acyclic graph,* and *web*. *Dynamics* is subdivided into *creation / termination*, *rate of change* and *patterns of change*.

Another approach has been put forward by Mahemoff and Johnston [46]: UI patterns can be assigned to four different categories. First, the *Task* category comprises all patterns addressing actions users might perform. Second, the *User Profile* category gathers patterns focusing on user groups. Third, *User-Interface Elements* helps designers and programmers to understand when to use a specific interface element or widget. Finally, *Entire System* patterns capture the issues of specific kinds of systems.

Van Welie and van der Veer [62] are organizing their patterns by means of "scaling the problem". As design is considered a top-down activity, their categorization is top-down as well. Problems are scaled from high-level problems like *Business Goals* to more detailed problems like *Task Level* and *Action Level* as shown in Figure 3. Another possibility to scale or group design patterns suggested by van Welie and van der Veer are to organize them according to their *Function* or to *Problem Similarity*, where *Function* can be subdivided into *Navigation*, *Searching*, *Product*, *Display*, *Layout*, and other sub-categories. Yet another organization principle suggested by van Welie and van der Veer is to categorize patterns according to *user tasks* and *user type*. A user task can be selecting things, finding things and sorting. This can be done by different types of users, namely novice users, intermediate users, and expert users.

VI. RELATIONSHIPS

Relationships between design patterns are a key concept to gain the full reuse potential of individual design patterns. In HCI patterns, relationships are typically described very briefly, only specifying the connections to other patterns which may be applicable to a particular design problem. However, proper consideration of relationships promises

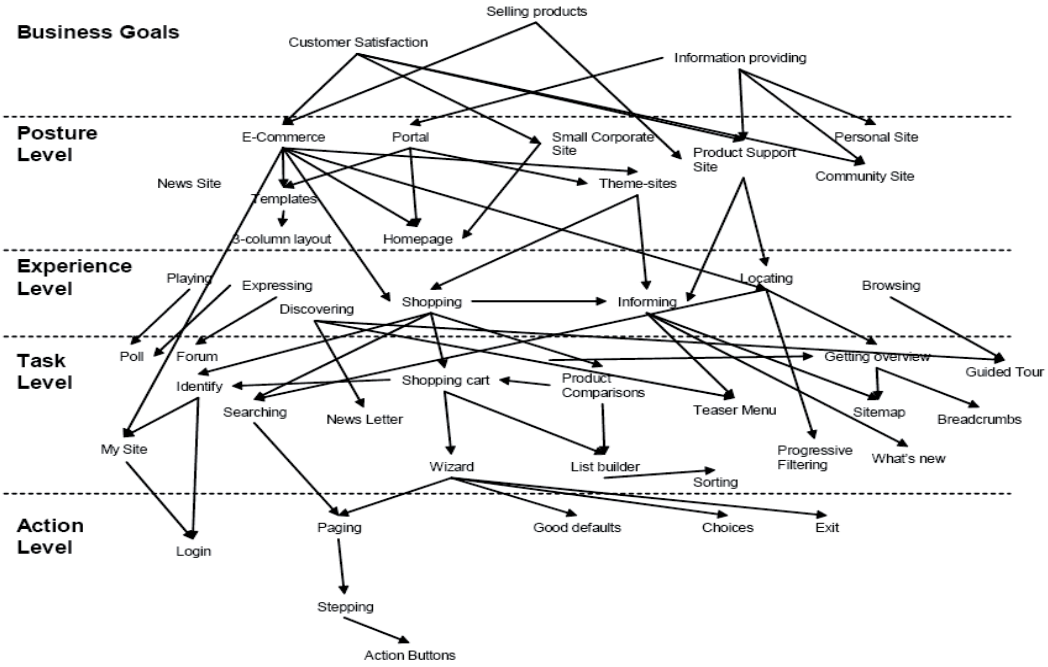


Figure 3: A part of a pattern language for Web design with focus on shopping [62]

even more powerful search and navigation opportunities. In the past, software engineering researchers have proposed possible categorization approaches for relationships in the domain of OOP patterns.

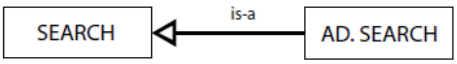


Figure 4: Specialization - The AD.SEARCH Pattern "is-a" specialization of the SEARCH PATTERN

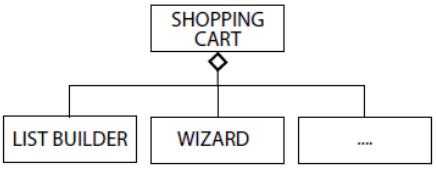


Figure 5: Aggregation - SHOPPING CARD consists of one or more design patterns.

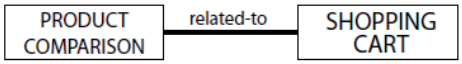


Figure 6: Association - SHOPPING CARD is "related to" PRODUCT COMPARISON

Primarily, relationships help to understand the complex interdependencies among design patterns. Pattern users can use relationship information in addition to the aforementioned pattern classifications to identify patterns which are applicable to specific design problems. Furthermore, relationships can be exploited for browsing large re-use repositories [36]. To improve and quicken the finding process, the browsing paradigm can be combined with the search paradigm as well. First the user searches for a specific design problem and based on the query results it is possible to browse through the repository to identify the best matching solution.

Noble's [50] proposal consists of three primary relationships (a pattern *uses* another pattern, a pattern *refines* another pattern, a pattern *conflicts* with another pattern) and a number of secondary relationships such as *used by*, *refined by*, *variant*, *similar*, *combines*, and others.

Zimmer's [67] approach deals with the classification of relationships in Gamma's [25] design patterns collection. He classifies relationships into three categories: a pattern *uses* another pattern in its solution, a pattern *is similar* to another pattern, and a pattern can be *combined* with another pattern. Beside this categorization, it is possible to modify existing relationships to use their altered version between different patterns. The application of categorized relationships allows to structure patterns in different layers. Zimmer has identified three semantically different layers: *basic design patterns and techniques*, *design pattern for typical software problems*, and *design patterns specific to an application domain*. Van Welie and van der Veer have indentified relationships similar to the relationship between classes in the software engineering domain [62]. They are using *Association*, *Specialization* and *Aggregation* to describe their identified relationships among design patterns. To illustrate the relationships they have provided examples how the relationships work. Figures 4, 5 and 6 show a summary of their explanations.

Beside the relationships discovered by van Welie we have investigated another one. It is called Anti-Association. It is similar to Association but it is a connection to a so-called anti-pattern, a pattern describing a bad solution approach.

VII. SEMANTICS

Beside the relationships, semantics of design patterns can be described using ontology. Throughout the Web community there exist many design patterns which describe the same problem but with a different vocabulary. So it is difficult to understand and to access this design knowledge. Therefore, an ontology or formalized semantics are necessary to provide a common vocabulary and a machine processable form of design patterns to be used by pattern management tools.

Over the years several approaches have been developed to overcome the aforementioned problem. Below we describe some of the research activities on this topic regarding HCI design patterns.

Montero et al. describe Web design patterns using DAML+OIL [48]. In their approach they are dealing with knowledge from two different areas. On the one hand there is the *Hypermedia Models* area which describes the elements of Web applications and is defined in four basic terms:

- **Node** – a place holder which contains a number of content elements.
- **Content** – a unit of information.
- **Link** – a connection between two or more nodes or contents.
- **Anchor** – the source or target of a link.

On the other hand, the *Design Patterns* area which represents design patterns with respect to their essential content elements. Therefore, a pattern in their ontology is defined in five different terms:

- **Name** – identifies the design pattern.
- **Category** – is used for classifying the pattern.
- **Problem** – describes the context in which the pattern can be applied and the problem it addresses.
- **Solution** – shows how the problem can be solved.
- **Related Patterns** – is referencing other similar or complementary patterns.

The ontology itself is specified in DAML+OIL [57], and is subdivided into three layers. The first layer represents the pattern and hypermedia elements and is the basis for the second layer, which represents the set of hypermedia design patterns. Finally, the instances of the hypermedia design pattern layer represent the third layer. For a more detailed specification and examples see [48].

Another approach is used by Scott Henninger and colleagues [31][32][33][34][35]. They are focusing on the development of a Web-based ontology to represent design patterns which are computed by agents. Their research goal is to put the loosely coupled pattern collections into strongly coupled pattern languages which represent the context in which usability patterns can be applied. Furthermore, it was important to find mechanisms for validating design patterns. Tool support plays a crucial role in their approach because it should be possible to get useful patterns for a specific design problem when going through a question and answer (Q&A) sequence. The results of which will be computed by an inference engine.

Henninger uses OWL (Web Ontology Language) [52] to define a metamodel for intelligent pattern languages. The metamodel describes pattern properties. Some of the properties Henninger is using are [33]:

- **hasProblem** – describes the design needs of an actor for which the pattern was created.
- **hasForces** – addresses constraints and tradeoff in choosing the solution suggested in the pattern.
- **hasSolution** – tells the user which actions must be taken to solve the problem.
- **hasContext** – sets the context where the design pattern can be useful
- **hasRationale** – describes why the solution is effective.

Beside these generic properties other local semantics and range restrictions are defined in the metamodel due to the fact that the metamodel supports different types of design pattern concepts (i.e. OOP-design patterns, HCI design patterns). Furthermore, the metamodel contains several types of semantic relationships to describe the connections between design patterns:

- **uses** – Pattern A uses Pattern B if the usage is optional [67].
- **requires** – Pattern A requires Pattern B [67].
- **alternative** – Two patterns are alternative if they share the same problem and context but exhibit different solutions [33].
- **conflictsWith** – Pattern A conflicts with Pattern B if they should not be used together in a design [33].

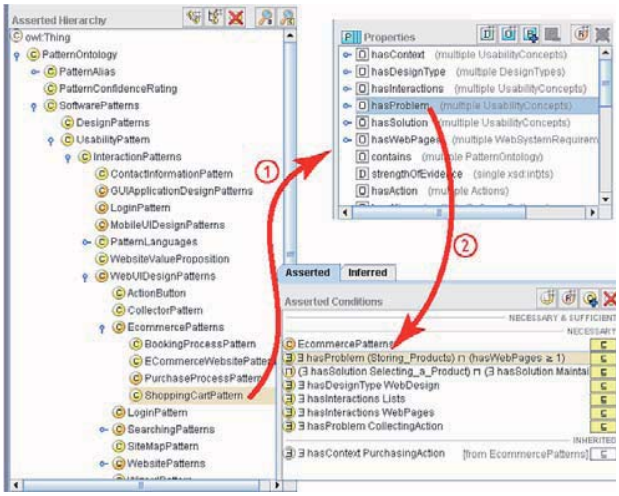


Figure 7: OWL Description of a Usability Design Pattern [33]

Figure 7 shows a part of an instance of a usability pattern with the developed metamodel. The metamodel builds on the HCI design pattern standardization approach by Fincher et al. called PLML (cf. Section IX) and is enriched with semantically meaningful pattern descriptions and relationships between patterns. The pattern in Figure 4 is a SHOPPING CART pattern which addresses the problem of storing products that a user has selected. This is accomplished with the property restriction "hasProblem (Storing_Products ⊓ hasWebPages ≥ 1)".

The restriction is defined in OWL DL (OWL Description Logic) to formalize the properties for OWL reasoners. In addition to the property *hasProblem* the pattern has other properties which define the problem and requirements on possible solutions. For a more in-depth description of Henninger's design pattern metamodel see [33].

To facilitate tool support Henninger combines BORE [30, 32] and the semantic Web representation of design patterns. BORE (Building on Organizational Repository of Experiences) is used to demonstrate semantic Web technologies to support the design of user interfaces. The design tool can define a methodology with a set of activities which describe the development process. It is using Q&As to customize the methodology which consists of all possible activities which are necessary to design a user interface. BORE builds on the experiences of many usability projects and various contexts and it uses a rule-based representation that captures the requirements of the system.

VIII. Evaluation and Validation of Design Patterns

There are a lot of HCI design patterns available in the community. Some are good and some are less valuable. The usefulness of a pattern is often subject to the eye of the beholder. But how do we measure the usefulness of design patterns according to quality criteria or formal metrics? We have investigated two approaches which are dealing with the evaluation of patterns and pattern catalogues in HCI as well as in the software engineering domain in a structured way.

Wurhofer et al. presents a *Quality Criteria Framework* which features five main quality criteria for HCI design patterns [63] and which is based on approaches from different researchers [10][39][47][49]. Figure 8 shows a summary of the quality criteria suggested by Wurhofer. In the following we give a short summary of each criterion suggested by the framework.

- **Findability** – means that a pattern must be found easily within a pattern language / collection. This implies that they must have a meaningful categorization and pattern name.
- **Understandability** – demands that the patterns' content elements (name, problem, solution,…) must be written in an clear and simple to understand manner. This is achieved by the below described sub-criteria:
  - *Completeness of Information* – states that that the pattern must carry all relevant information (forces, problem, context, solution, example, etc.).
  - *Language* – means that the pattern must be written in easy understandable terms and short sentences.
  - *Problem-Centeredness* – all parts of the pattern should be centered on the problem and the problem solution relationship must be clear.
  - *Balance between Concreteness and Abstractness* – the pattern should not be too abstract nor to detailed.
  - *Comprehensiveness of Pattern Parts* – this criterion ensures that each part of the pattern covers everything important to the user.
- **Helpfulness** – ensures that each pattern is written in such a style that the pattern gives the user as much information as possible to implement it worries. This criterion is achieved through six sub-criteria:
  - *Improvement of Design / Architecture* – the quality of a pattern is verified if it helps to improve the design or development of a system.
  - *Problem Solving* – the pattern should help to avoid common pitfalls by using common solutions to the addressed problem.
  - *Support of Communication* – states that the design pattern should serve as the common "language" for all stakeholders.
  - *Capturing of Knowledge* – this criterion stands for the reuse aspect of the design pattern. It should capture relevant knowledge in its domain to the user.
  - *Memorability* – the main idea of a pattern must be kept in mind of the stakeholders. This can be achieved using an appropriate and easy to remember pattern name or a good sensitizing image.
  - *Feasibility* – the patterns solution should be realized easily.
- **Empirical Verification** – means that a pattern based on empirical studies has a higher quality than patterns based on personal experience.
- **Overall Acceptability** – states how much a pattern user agrees with the pattern's content. To fulfill this criterion it is important to support the pattern user's subjective acceptance of a pattern. This can be achieved by increasing the *Overall Believe in Pattern* and the *Overall Agreement with Pattern*.

With this criteria catalogue it is possible to validate design patterns according to their quality.

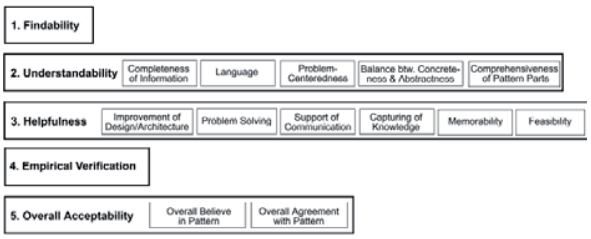


Figure 8: Components of the Quality Criteria Framework [63]

Another approach has been developed by Cutumisu et al. [16]. They propose how to evaluate the effectiveness of a design pattern catalogue or compare different catalogues according to their effectiveness. The authors developed a metric with which it is possible to validate patterns within an existing design pattern catalogue. They defined four metrics which are dependent on a specific application. That means the metrics take the patterns which are used in a specific application and compare them using various formulae to the used pattern catalogue. Cutumisu et al. define the four metrics as follows:

- ***usage*** *– is the ratio of patterns used in the application that come from the catalogue to the total number of patterns in the catalogue.*
- ***coverage*** *– is the ratio of catalogue patterns used in the application to the total number of patterns used in the application.*
- ***utility*** *– is the ratio of pattern instances in the application whose patterns are in the catalogue to the total number of patterns used in the application that come from the catalogues.*
- ***precision*** *– is the ratio of the total number of patterns used in the application that come from the catalogue to the number of adaptations required for these pattern instances.*

If a pattern catalogue has a high *usage*, *coverage*, *utility,* and *precision* it is, according to Cutumisu et al. a good pattern catalogue. Although the pattern metrics are designed for software patterns, it is easy to adapt them to the HCI design pattern domain. For a more detailed description and the equations for each of the metrics see [16].

IX. TOOL SUPPORT

There are various tools which are exploiting the reuse potential of HCI design patterns. These tools can be categorized into online libraries / catalogues, pattern management tools, and pattern-based UI design tools. Due to space limitation we describe, in our mind, the most important ones.

A. Pattern Libraries / Collections

Pattern libraries or collections are focusing on the categorization and dissemination of patterns via the Internet. Sometimes there are basic mechanisms provided to create and submit patterns to a repository. Such an online pattern library is the *Yahoo! Design Pattern Library* [43, 66], a part of the Yahoo! Developer Network. The founder's intention of the Yahoo! Library was to provide a tool to increase the consistency and usability across Yahoo! and the productivity of the UI design team. Today the design pattern library is an often-used tool for UI designers and researchers. Currently, there are about 47 patterns in six categories available. Each pattern undergoes an extensive review process within Yahoo!. They are reviewed, revised, and rated. After review, the patterns are published and made available to the public. All patterns in this library are under the Creative Commons Attribution 2.5 License (June 2009). Main features of the library are:

- A *blogging tool* for discussing patterns in the library.
- A *history function* which helps users to see which changes were made over time.

Further online pattern libraries are Welie.com [64], which provides 130 UI design patterns with the possibility to export them to PLML [22]. Furthermore, website users can comment and discuss certain patterns.

As an addition to her *Designing Interfaces* book [60], Tidwell provides a pattern library with selected patterns where she updates and publishes new patterns. This library is available at [60].

B. Pattern Management Tools

Pattern management tools are focusing on manipulating patterns, navigating through pattern libraries, and providing mechanisms to add relationships between patterns to create a pattern language. They are easy to access and pattern users can communicate with others via the pattern repository.

MOUDIL (Montreal Online Usability Patterns Digital Library) [26] is a comprehensive framework for capturing and disseminating patterns. It provides features and tools like:

- Submission of patterns in *different formats*.
- International *review and validation* of submitted patterns.
- A *pattern editor* for adding semantic information to the patterns.
- A *pattern navigator* which allows navigating in different ways through the pattern library.
- A *pattern viewer* which provides different views of the pattern.

Unfortunately, the prototype of this pattern library is not longer available online.

Currently under development is another online pattern management tool which employs XPLML [40], an improved version of PLML. XPLML provides a set of common content elements, and it is possible to add semantic information to design patterns. The tool will offer features such as:

- A *pattern editor* with functions to support pattern authors in writing and updating design patterns.
- A *design pattern language visualization tool* for presenting relationships between patterns in a pattern language.
- The *pattern form transformation* allows pattern users to change the presentation form of a design pattern. For example, if a user prefers the Alexandrian form, the tool provides mechanisms to change the pattern form from e.g. Tidwell's to the preferred (Alexandrian) form in order to maximize user acceptance.
- A *wiki functionality* which should involve all interested users in developing new and improving existing patterns.

C. Pattern-based UI design tools

The last category describes pattern-based UI design tools. They provide functions for using design patterns in UI design activities. Patterns are used for generating user interfaces in a semi-automatic way. These tools usually provide a defined set of UI patterns, which can be used within the tool as building blocks to create the UI system.

PIM (Patterns in Modeling) [54], a model-based UI development tool, aims to support UI designers in composing the UI models through pattern application. With *PIM* it is possible to develop user interfaces on a more abstract and conceptual way. This helps designers to handle very complex systems more easily. Users can put their attention on conceptual properties rather than being distracted by technical and implementation details.

A further tool, developed by Ahmed and Ashraf, is called *Task Pattern Wizard* [2]. It is based on XUL (XML User Interface Language) [65] to describe the patterns and models. UI design patterns are used as modules for establishing task, dialog, presentation, and layout models. The tool guides the UI designer through the pattern adaption and integration process and it provides functions for using, selecting, adapting, and applying patterns within the proposed framework PD-MBUI (Pattern-Driven and Model-Based User Interface). The framework tries to unify the pattern-driven and model-based approaches, two methods for UI and software engineering. A more detailed description of the framework is given in [2].

DAMASK, developed by Lin and Landay [44], is a prototyping tool to produce Web UI's across different devices with the support of design patterns. The tool relies on two components. The *layer component* specifies which parts of the UI can be used across all devices and which can only be used on a single device. The second component is the *pattern component*: In *DAMASK,* an HCI design pattern consists of pre-defined UI elements that are optimized for each device. The pattern repository of *DAMASK* has 90 patterns from "The Design of Sites" [18] which can be extended by the UI designer. The UI designer sketches out a UI for one device and *DAMASK* constructs an abstract model from which it generates the UI's for the other devices. Once the first layout is established, the UI designer

can refine this layout and *DAMASK* changes the UI's for the other devices accordingly. Furthermore, the tool provides a function for testing the established UI's.

X. STANDARDIZATION APPROACHES

To our knowledge, the only serious standardization approach was started at a workshop at a Human-Computer Interaction conference in 2003. Due to the vast amount of design pattern forms, Fincher et al. [22] proposed a standard pattern form for HCI patterns called PLML (pronounced "pell mell"). The goal was to provide a standard pattern form where common elements should help pattern authors and users to use design patterns across different collections. PLML is specified in XML and comprises 16 content elements on which the workshop participants agreed. It turns out that only van Welie's pattern collection [64] makes use of PLML. He provides an export function to transform patterns from his collection to PLML. However, this approach suffers from certain technical limitations as Kamthan points out [38]. He mentions that the design principles behind the PLML DTD are not specified and that elements are not strictly enough defined, because of the broad use of the XML `ANY` element in the specification. Kamthan also points out that PLML does not describe semantic relationships between patterns, which are necessary when using PLML in a pattern language.

Since the publication of PLML, researchers tried to improve it. PLML v. 1.2. developed by Deng et al. [17], is an augmented PLML with some additional elements but does not solve serious shortcomings such as the lack of formalized relationships among patterns.

XI. CONLUSION AND FUTURE WORK

The concept of HCI design patterns is widely accepted tool to represent design knowledge in a reusable format. In the last years many concepts concerning the components of HCI patterns were proposed, such as pattern forms, organizing principles, standardization approaches, ontology, evaluation and verification of patterns. This diversity leads to blurred conceptualization and may confuse especially novice users. To exploit the full reuse potential of patterns, a unification of the above discussed components should be established, which does not constrain pattern authors in their work but supports pattern users by easing understanding and instantiation of patterns to specific design problems [40]. Therefore a universal pattern form needs to be established and enriched with semantics. This article shows that there are many HCI design pattern resources available on the WWW and because of the vast amount of different design patterns available there exists many different forms as well. To overcome the problem, an appropriate ontology would help to share and disseminate HCI design patterns among different repositories and help the authors and pattern users to work more efficient with the provided design knowledge. Therefore, a lot of research must be undertaken which includes analyzing the different design pattern forms and examining the content elements' semantics. Furthermore, to agree on a "standard" pattern form it is necessary to discuss the results of the above mention research with the HCI design pattern community to agree on a unified HCI design pattern structure.

REFERENCES

[1] M. Adams, J. Coplien, R. Gamoke, R. Hanmer, F. Keeve, and K. Nicodemus, "Fault-Tolerant Telecommunications System Patterns", in Pattern Languages of Program Design 2, pp. 549-562, Addison-Wesley, 1996

[2] S. Ahmed and G. Ashraf, "Model-based User Interface Engineering with Design Patterns", in Journal of Systems and Software, vol. 80, pp. 1408-1422, 2007

[3] C. Alexander, S. Ishikawa, and M. Silverstein, "A Pattern Language", Oxford University Press, 1977

[4] C. Alexander, "Notes on the Synthesis of Form", Harvard University Press, 1964

[5] C. Alexander, "The Oregon Experiment", Oxford University Press, 1975

[6] C. Alexander, "The Timeless Way of Building", Oxford University Press, 1979

[7] E. Bayle, "Putting it all together: Towards a Pattern Language for Interaction Design: A CHI workshop", in SIGCHI Bulletin, vol. 30, pp. 17 – 23, 1998

[8] K. Beck and W. Cunningham, "Using Pattern Languages for Object-Oriented Programs", OOPSLA'87:Workshop on the Specification and Design for Object-Oriented Programming, 1987

[9] J. Borchers, "CHI Meets PLoP: An Interaction Patterns Workshop", SIGCHI Bulletin, vol. 32, 2000

[10] J. Borchers and J. Thomas, "Patterns: What's in it for HCI?", CHI'01: Extended Abstracts on Human Factors in Computing Systems, ACM Press, pp. 225 - 226, 2001

[11] J. Borchers, "A Pattern Approach to Interactive Design", Wiley, 2001

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stahl, „Pattern-Oriented Software Architecture: A System of Patterns", vol. 1, Wiley, 1996

[13] E. Chung, J. Hong, J. Lin, M. Prabaker, J. Landay, and A. Liu, "Development and Evaluation of Emerging Design Patterns for Ubiquitous Copmuting", in Proc. Of Designing Interactive Systems, 2004

[14] T. Coram and J. Lee, "Experiences – A Pattern Language for User Interface Design, 1996, Available at: http://www.maplefish.com/todd/papers/Experiences.html, Accessed on: June 30, 2010

[15] C. Crumlish and E. Malone, "Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience", O'Reilly Media, 2009

[16] M. Cutumisu, C. Onuczko, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, J. Siegel, and M. Carbonaro, "Evaluating Pattern Catalogs: The Computer Games Experience", ICSE '06: Proceedings of the 28th International Conference on Software Engineering, ACM, pp. 132—141, 2006

[17] J. Deng, E. Kemp, and G. Todd, "Focussing on a Standard Pattern Form: The Demelopment and Evaluation of MUIP", in Proc. of the Seventh ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction, pp. 83-90, ACM Press, 2006

[18] D. van Duyne, J. Landay, and J. Hong, "The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience", Addison-Wesley, 2003

[19] T. Erickson, "The Interaction Design Patterns Page", Available at: http://www.visi.com/~snowfall/InteractionPatterns.html, Accessed on June 30, 2010

[20] S. Fincher, "The Pattern Gallery", Available at: http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html, Accessed on June 30, 2010

[21] S. Fincher, J. Finlay, S. Greene, L. Jones, P. Matchen, J. Thomas, and P. Molina, "Perspectives on HCI Patterns: Concept and Tools", CHI'03: Extended Abstracts on Human Factors in Computing Systems, ACM Press, pp. 1044 – 1045, 2003

[22] S. Fincher, "Perspectives on HCI Patterns: Concepts and Tools (introducing PLML)", Interfaces, vol. 56, pp.26-28, 2003

[23] S. Fincher, "The Pattern Gallery", Available at: http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html, Accessed on June 30, 2010

[24] S. Fincher and P. Windsor, "Why Patterns are not enough: Some Suggestions Concerning an Organising Principle for Patterns of UI Design", CHI'2000 Workshop on Pattern Languages for Interaction Design: Building Momentum, 2000

[25] E. Gamma, R. Helm, R. Johnson, andJ. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley Reading, 1994

[26] A. Gaffar, H. Javahery, A. Seffah, and D. Sinning, "MOUDIL: A Comprehensive Framework for Disseminating and Sharing HCI Patterns", CHI'03 workshop: Perspectives on HCI patterns: Concepts and Tools, 2003

[27] I. Graham, "A Pattern Language of Web Usability", Addison-Wesley, 2003

[28] R. Griffiths, L. Pemberton, and J. Borchers, "Usability Pattern Language Workshop", at INTERACT'99, Available at: http://www.it.bton.ac.uk/staff/rng/UPLworkshop99/PositionPapers.html, Accessed on: June 30, 2010

[29] "HCIPATTERNS.ORG", Available at: http://www.hcipatterns.org/, Accessed on June 30, 2010

[30] S. Henninger, "Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle", ICSR '96: Proceedings of the 4th International Conference on Software Reuse, IEEE Computer Society, 1996

[31] S. Henninger, M. Keshk, and R Kinworthy, "Capturing and Disseminating Usability Patterns with Semantic Web technology", Workshop at CHI 2003: Perspectives on HCI Patterns: Concepts and Tools, 2003

[32] S. Henninger, "Tool Support for Experience-Based Software Development Methodologies", in Advances in Computing, vol. 59, pp. 29 – 82, 2003

[33] S. Henninger and P. Ashokkumar, "An Ontology-Based Metamodel for Software Patterns", 18th International Conference on Software Engineering and Knowledge Engineering (SEKE2006), pp. 327 – 330, 2006

[34] S. Henninger, "Disseminating Usability Design Knowledge through Ontology-Based Pattern Languages", Proc. Semantic Web User Interaction Workshop (ISWC2006). 2006

[35] S.Henninger and P. Ashokkumar, "An Ontology-Based Infrastructure for Usability Design Patterns", Proc. First International Workshop Semantic Web Enabled Software Engineering, 2005

[36] M. Hitz and H. Werthner, A Graph Oriented Approach to Enhance Reusability in *-bases, WISR'92: 5th Annual Workshop on Software Reusability, 1992

[37] M. Irons, "Patterns for Personal Web Sites", Available at: http://www.rdrop.com/~half/Creations/Writings/Web.patterns/, Accessed on: June 30, 2010

[38] P. Kamthan, "A Critique of Pattern Language Markup Language", Interfaces, vol. 68, pp. 14-15, 2006

[39] D. Khazanchi, J. Murphy, and S. Petter, "Guidelines for evaluating patterns in the IS domain", MWAIS 2008 Proceedigs, 2008, Paper 24, Available at : http://aisel.aisnet.org/mwais2008/24, Accessed on: June 30, 2010

[40] C. Kruschitz, "XPLML: A HCI Pattern Formalizing and Unifying Approach", in Proc. of the 27th International Conference Extended Abstracts on Human Factors in Computing Systems, pp. 4117 – 4122, ACM, 2009

[41] C. Kruschitz and M. Hitz, "The Anatomy of HCI Design Patterns", Proc. of Computation World; Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, IEEE Computer Society, pp. 202 – 207, 2009

[42] T. Kunert, "User-Centered Interaction Design Patterns for Interactive Digital Television Applications", Springer, 2009

[43] M. Leacock, E. Malone, and C. Wheeler, "Implementing a Pattern Library in the Real World: A Yahoo! Case Study", ASIS&T IA Summit, 2005

[44] J. Lin and J. Landay, "Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces", in Proc. 26th International Conference on Human Factors in Computing Systems, pp. 1313-1322, ACM, 2008

[45] "Littles Springs Design – Mobile UI Design Resources", Available at: http://patterns.littlespringsdesign.com/index.php/Main_Page, Accessed on: January 18, 2010

[46] M. Mahemoff and L. Johnston, "Pattern Language for Usability: An Investigation of Alternative Approaches", in Proc. Third Asian-Pacific Conference in Computer Human Interaction, pp. 25-31, IEEE Coomputer Society, 1998

[47] K. McGee, "Patterns and Computer Game Design Innovation", IE'07: Proc. of the 4th Australasian Conference on Interactive Entertainment, pp. 1 – 8, 2007

[48] S. Montero, P. Díaz, and I. Aedo, "Formalization of Web Design Patterns using Ontologies", Proc. Of the 1st International Atlantic Web Intelligence Conference, Springer-Verlag,Vol. 2663, pp. 179 – 188, 2003

[49] S. Nieburg, K. Kohler, and C. Graf, "Engaging Patterns: Challenges and Means Shown by an Example", Engineering Interactive Systems, Springer, pp. 586 – 600, 2008

[50] J. Noble, "Classifying Relationships Between Object-Oriented Design Patterns", in Proc. of the Australian Software Engineering Conference, 1998, IEEE Computer Society

[51] D.A. Norman and S.W. Draper, "User-Centered System Design: New Perspectives on Human-Computer Interaction.", Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1986

[52] "OWL – Web Ontology Language", Available at: http://www.w3.org/TR/2004/REC-owl-guide-20040210/, Accessed on June 30, 2010

[53] "Portland Pattern Repository", Available at: http://c2.com/ppr/, Accessed on January 5, 2010

[54] F. Radeke, P. Fobrig, A. Seffah, and D. Sining, „PIM Tool: Support for Pattern-Driven and Model-Based UI Development", 5th International Workshop: Task Models and Diagrams for User Interface Design, LNCS: Programming and Software Engineering, vol 4385, pp. 82-96, 2007

[55] T. Schuemmer and S. Lukosch, "Patterns for Computer-Mediated Interaction", John Wiley & Son, 2007

[56] R. Smith, "Panel in Design Methodology", OOPSLA'87:Addendum to the Proceedings on Object-Oriented Programming Systems, Languages and Applications, pp. 91-95, ACM, 1987

[57] "The DARPA Agent Markup Language", Available at: http://www.daml.org, Accessed on June 30, 2010

[58] "The Hillside Group", Available at: http://hillside.net, Accessed on June 30, 2010

[59] J. Tidwell, "Common Ground", Available at: http://www.mit.edu/~jtidwell/common_ground.html, Accessed on June 30, 2010

[60] J. Tidwell, "Designing Interfaces", O'Reilly, 2005 Available at: http://www.designinginterfaces.com, Accessed on June 30, 2010

[61] "UI Patterns - User Interface Design Pattern Library", Available at: http://ui-patterns.com, Accessed on June 30, 2010

[62] M. van Welie and G. van der Veer, "Pattern Languages in Interaction Design: Structure and Organization", Human-Computer Interaction – INTERACT'03, pp.527-534, IOS Press, 2003

[63] D. Wurhofer, M. Obrist, E. Beck, and M. Tscheligi, "Introducing a Comprehensive Quality Criteria Framework for Validating Patterns", Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Computation World, pp. 242 – 247, 2009

[64] "Welie.com – Patterns in Interaction Design", Available at: http://www.welie.com, Accessed on June 30, 2010

[65] "XML User Interface Language", Available at: https://developer.mozilla.org/En/XUL, Accessed on June 30, 2010

[66] "Yahoo! Design Pattern Library", Available at: http://developer.yahoo.com/ypatterns/, Accessed on June 30, 2010

[67] W. Zimmer, "Relationships between Design Patterns", Pattern Language of Program Design, Addison-Wesley, 1994

Metrics for the Evaluation of Adaptivity Aspects in Software Systems

Claudia Raibulet, Laura Masciadri
Dipartimento di Informatica Sistemistica e Comunicazione,
Universitá degli Studi di Milano-Bicocca
Milan, Italy
raibulet@disco.unimib.it, laura.masc@gmail.com

***Abstract*—Runtime adaptivity is related to the ability of the information systems to perform changes by themselves and on themselves during their execution. The engineering of runtime adaptivity is one of the most challenging issues to address in today's information systems. This is due to the fact that runtime adaptivity requires additional elements at the architectural or structural levels. Moreover, it increases the dimension and computation of a system. Its advantages are mostly related to the improvement of performances, enhancement of the functionalities' quality and automation of administrative tasks. In this paper we propose a set of metrics for the description and evaluation of adaptive properties of the information systems and of the frameworks which provide support for the development of adaptive systems. They aim to provide a concrete mechanism to analyze the quality of the design of adaptive systems, to determine the type of adaptivity of a system or to compare the adaptive features of different systems. Metrics are grouped into six categories: architectural, structural, performance, interaction, documentation, and miscellaneous. They have been identified and specified by analyzing several case studies which address runtime adaptivity issues through different approaches with different objectives in various application domains and several frameworks for the design and implementation of adaptive systems.**

Keywords-adaptivity, adaptive systems, evaluation, software metrics.

I. INTRODUCTION

Runtime adaptivity [3], [5], [7], [10], [15] indicates the ability of a system or software to perform changes by itself and on itself during its execution. The objectives of changes may be of various natures and may concern different issues which range from addressing unpredicted situations to ensuring the optimal working of a system's resources or to improving the performances of a system. Essentially, they result from the need to address the growing complexity of emerging systems and to improve productivity and performance, as well as to automate configuration, re-configuration, control and management tasks [15].

Due to the wide range of possible objectives, types and solutions related to runtime adaptivity, it would be very useful to have common mechanisms to evaluate and compare adaptive approaches in order to choose the most appropriate solution for the current needs, to integrate various solutions, or to make these solutions cooperate to achieve complex tasks.

We tried to address these issues by considering the available solutions described in the scientific literature [2], [5], [7], [8], [16] in order to determine how adaptivity is actually achieved, the main characteristics of the design of adaptive systems, as well as the advantages outlined by the authors of the adaptive systems.

The conclusions are summarized as follows. Adaptivity is a complex task. Independently of what it is changed at runtime, an adaptivity pattern consisting of four main steps (which should be implemented by any adaptive system) can be specified: monitoring (to retrieve information about the context or status of a system which is exploited at run-time in the adaptation process), analysis of the monitored information, decision (to determine whether changes should be made or not and, in the affirmative case, to choose the best solution for the current situation) and application of identified changes [3], [5], [10], [14].

Adaptivity requires additional elements at the architectural or structural levels in order to implement these steps. Even if it is considered a non-functional requirement, it influences the execution of a system, its interaction with the external world, and its performances. Therefore, its design and implementation are fundamental for a system's lifecycle.

Authors describe the advantages of adaptive systems in terms of performances, simplified and enhanced interaction with the users, and automation of administrative tasks. However, the evaluation of the described solutions is *adaptive* and *case study oriented*: the authors provide their point of view and outline the strong aspects of their solutions through a particular vocabulary/terminology. Therefore, it is difficult, if not impossible, to evaluate and compare adaptive systems.

Furthermore, the scientific literature presents also various frameworks [5], [7], [16], each introducing a different approach for the design and implementation of adaptive mechanisms. For example, the Rainbow framework [5] proposes a control loop which defines mechanisms to monitor the runtime properties of a system, to evaluate constraint violations, and to perform global and module level adaptations on a running system. All these mechanisms are provided at the architectural level. On the other hand, the Adaptive Server Framework (ASF) [7] describes an infrastructure of components and services which facilitates the construction of adaptation from a behavioral perspective. Hence, when developing an adaptive system, on which basis

there can be evaluated which of these frameworks is more appropriate for the current requirements?

In this context, we propose a set of metrics which may be adopted in the description, design, and evaluation of the adaptive properties of information systems and frameworks. The metrics are grouped into the following categories: architectural, structural, interaction, performance, documentation, and miscellaneous [12]. The architectural and structural metrics are mostly related to the design issues of adaptive systems; while the interaction and performance metrics reflect the advantages regarding the usability of the adaptive systems. Even if it may play a secondary role, the documentation category may be considered an indication on the usability and reusability, personalization, and the advantages of adaptive systems, as well as about their overall quality. Furthermore, it is a valuable indication on the usability and the overall quality of a framework for adaptive systems.

For the presentation of the metrics of adaptivity, this paper considers four of the available case studies, which in our opinion are representative for this topic: a Web-based client-server system and a video conferencing system which exploit the Rainbow framework [5], an adaptive image server which uses the Adaptive Server Framework [7], and the AHA! [2] system for adaptive e-learning.

The rest of the paper is organized as follows. Section II provides an overview on four representative adaptive case studies. Section III introduces the categories and subcategories of metrics defined in this paper. The application of the defined metrics to the four cases studies introduced in Section II is discussed within Section IV. The similar approaches for the evaluation of runtime adaptivity are addressed in Section V. Conclusions and future work are dealt with in Section VI.

II. CASE STUDIES ON RUNTIME ADAPTIVITY

This section introduces four of the case studies we have considered in the identification and specification of metrics. In Section X, an analysis of these case studies from the metrics point of view is presented.

A. Web-based Client-Server (WebCS)

In this case study Web-clients make requests of contents to various Web-server groups [5]. Adaptivity is related to the system's performances and more precisely to the response time the clients perceive for their requests to the Web servers. The two factors which influence the response time are the servers' load and the available bandwidth. To address this performance issue through adaptivity, an architectural level approach is adopted consisting in the definition of an architectural style enriched with adaptation operators and strategies for the dynamic aspects of a system [5]. Furthermore, each client has associated an invariant which verifies if the response time is less than a predefined value. If this is not true an adaptivity strategy is invoked. The results of the application of the adaptive strategies are reflected at the architectural level.

B. VideoConferencing (VConf)

This case study deals with the management of videoconferences in which participants may use various videoconferencing tools and communication protocols [5]. Adaptivity is related to the performance (determined essentially by the available bandwidth) and cost (determined essentially by the gateway costs) aspects. As in the previous example, adaptivity is addressed at the architectural level through an architectural style. Each handheld device and gateway has associated an invariant which establishes the range of the accepted values. Whenever an invariant is violated, an adaptive strategy is invoked. The results of the application of the adaptive strategies are reflected at the architectural level.

C. Adaptive Image Server (AIS)

In this case study, clients send to a server requests for images specifying the minimum and maximum resolution for the requested images [7]. In a non-adaptive scenario, the server provides the images with their current resolutions. In an adaptive scenario, the server scales the images resolution in order to optimize the overall performances of the system. In this case, performances are translated into the improvement of the throughput and the reducing of the response time which are determined by the resolution and the quality of an image. Furthermore, adaptivity takes into consideration also its overhead introduced in the system: the computation load (of the CPU) of the server because the time needed to process images may influence significantly the response time. This is compared to the latency determined by sending non-modified images.

D. Adaptive Hypermedia Architecture (AHA!)

AHA! [2] is an adaptive e-learning system. Adaptivity regards the content (visualized to a user as pages) and the navigation in the content (implemented through links) based on the knowledge level of a user. The information offered by AHA! is organized hierarchically (consisting in fragments-pages-courses) through a domain model. Each element in this domain model may have associated one or more concepts.

When a user requires a page, based on (1) the user's model (consisting in concepts which have associated a set of attributes among which his knowledge level) and (2) the adaptivity rules defined by the adaptation model, an adaptivity engine (I) builds the requested page (inserting the content and the navigation elements) accordingly to the user's current knowledge level, (II) updates the user model (through the rules defined by the adaptation model by considering the concepts inserted in the requested page) and (III) visualizes the page.

III. METRICS FOR RUNTIME ADAPTIVITY EVALUATION

In the definition of the metrics we have assumed that the functional part of a system is designed first, (or more generally a system should provide a version of its functionalities which does not exploit adaptivity), and further it is enriched with adaptive mechanisms.

Each metric is presented through its name, description and further explanations wherever necessary.

We have defined five categories of metrics each of them may be further divided into subcategories. Furthermore, we have defined additional aspects which may be considered to evaluate adaptivity and which are grouped in a miscellaneous category due to the fact that they capture different facets of adaptivity. An overview on these metrics is presented in Figure 1.

The architectural category of metrics aims to capture the main features of adaptivity which emerge at the system level. These characteristics are visible and meaningful when considering a global perspective on the architecture of a system. The metrics in this category are further divided in two subcategories: architectural growth and architectural separation of concerns.

The structural metrics are collocated at a lower abstraction level then the metrics in the previous category. They concern the actual implementation aspects of an adaptive system. The metrics in this category are further divided in three subcategories: structural growth, structural separation of concerns, and personalization.

The interaction category focuses on the advantages provided by runtime adaptivity in terms of the automation of the human tasks. Hence, it considers both the interactions of the administrators and the final users with an adaptive system.

The performance metrics aim to evaluate the quality of the functionalities provided by the adaptive systems. Hence, the aspects they consider are visible and meaningful for the final users of a system.

The documentation category provides information on the quality of a design of an adaptive system or a framework for the implementation of adaptive systems. It defines meaningful metrics for the usability and understandability of the adaptive properties.

The miscellaneous indexes propose further evaluation mechanisms which may be exploited to analyze the overall effort necessary to implement adaptive functionalities.

These categories of metrics are described in detail in the following sections.

A. Architectural Metrics

During the presentation of the architectural metrics we use the term *elements* as defined by [1] for software architecture: "The software architecture of a program or computing system is a structure or structures of a system, which comprise software *elements*, the external visible properties of those elements and the relationships among them". Hence, the term element expresses an architectural unit (e.g., components and connectors [6]). The metrics are

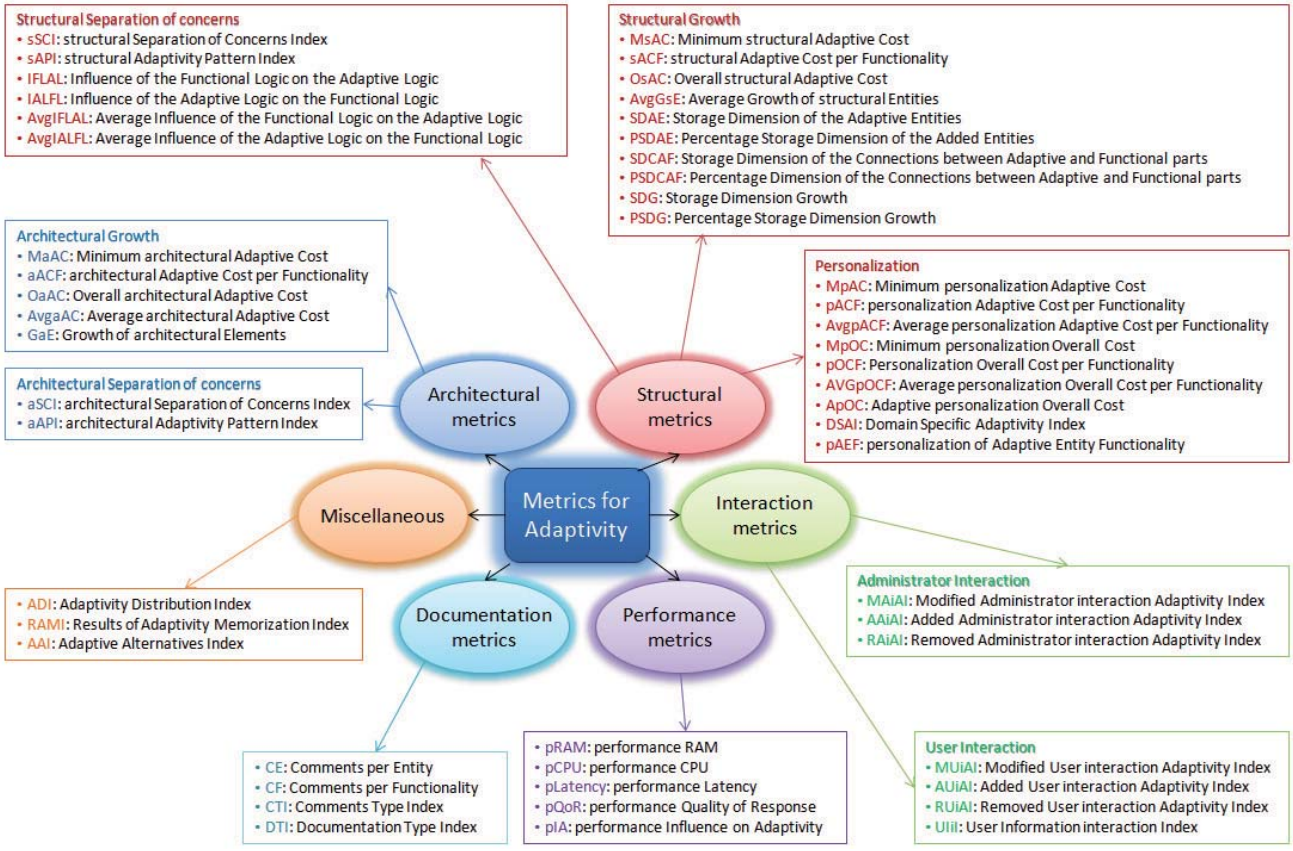


Figure 1. Overview on the metrics for runtime adaptivity

valid and applicable to any architectural element.

Generally, the systems addressing runtime adaptivity through architectural mechanisms are composed of two parts: functional and adaptive. The adaptive part is usually composed of four main conceptual elements corresponding to the adaptation steps: monitoring, analyzing, deciding and changing (see Figure 3-A).

The architectural metrics concern two main aspects: separation of concerns and architectural growth.

The separation of concerns regards two aspects: (1) the separation between the functional logic and the elements ensuring adaptivity, and (2) the separation among the elements implementing the four steps of adaptivity. It is expressed through two metrics.

aSCI: architectural Separation of Concerns Index

$$aSCI = \frac{Number\ of\ adaptive\ elements}{Number\ of\ functional\ elements\ modified}$$

This metric indicates the degree of dependence between the functional logic and the adaptive elements of a system at the architectural level. It enables the evaluation of the separation of concerns at the architectural level by comparing the number of elements inserted in the adaptive part (which provide exclusively adaptive functionalities) and the number of the functional elements which have been modified (to interact with the adaptive ones).

aAPI: architectural Adaptivity Pattern Index

$$aAPI = Number\ of\ adaptive\ conceptual\ elements$$

This metric indicates the separation of concerns between the main conceptual (types of) elements implementing the four steps defined by the adaptivity pattern at the architectural level. If the value of this metric is four, then the adaptive part of a system defines at least a conceptual element for each of these four steps. If it is zero, the adaptive part of the architecture is totally integrated with the functional one. The values between zero and four suggest that two or more adaptivity steps are provided by the same conceptual architectural element.

These two metrics provide useful information related to the modularity, reusability and maintainability of the adaptive part of a system.

The architectural growth regards the number of elements introduced by the adaptive part of an architecture. It is express through five metrics.

MaAC: Minimum architectural Adaptive Cost

$$MaAC = Minimum\ number\ of\ elements\ for\ adaptivity$$

This metric indicates the minimum number of elements which should be added to make a system adaptive independently of the number of functionalities it provides. Essentially, this metric expresses the fix cost of adaptivity at the architectural level. It considers the adaptive elements necessary to make the first functionality adaptive.

aACF: architectural Adaptive Cost per Functionality

$$aACF = Number\ of\ elements\ for\ the\ i^{th}\ functionality$$

This metric indicates the number of elements which should be added to make the i-th functionality adaptive. It may be seen as a variable cost for introducing adaptivity per functionality at the architectural level.

OaAC: Overall architectural Adaptive Cost

$$OaAC = MaAC + \sum_{i=2}^{n} aACF$$

The sum between the last two metrics expresses the architectural growth in number of elements needed to add adaptivity (see Figure 2). The results obtained through this metric are dependent on the order in which the adaptive functionalities of a system are actually implemented. If the function has a linear evolution, then the adaptive functionalities may be considered independent of each other, hence they need independent components (see Figure 3-A). Otherwise, if the function has a logarithmic evolution, then the adaptive functionalities may share common components achieving implicitly reusability issues (see Figure 3-B).

AvgaAC: Average architectural Adaptive Cost

$$AvgaAC = \frac{OaAC}{n}$$

This metric expresses the average growth per functionality at the architectural level due to the introduction of adaptivity. It indicates the average number of elements which have been added for each functionality. Hence, for each functionality it is added 1/n (where n is the total number of functionalities) of the fix costs related to the introduction of adaptive mechanisms in a system.

As for the OaAC metric, if this function has a linear behavior then we can assume that the adaptive functionalities are independent of each other (see Figure 3-A).

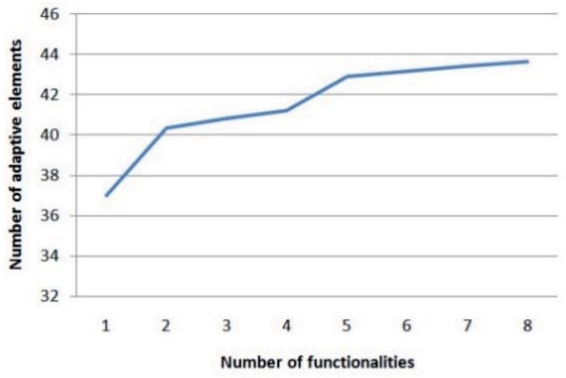


Figure 2. Overall architectural Adaptive Cost

Otherwise, if the function has a logarithmic behavior, there is a reuse of several of the already inserted elements in the adaptive part (see Figure 3-B). In this case the behavior of the function may be influenced by the order in which functionalities are made adaptive (because some of them share common elements). Figure 3 shows the generic components implementing the four steps of the adaptivity pattern: 1a for the monitoring, 2a for the analyzing, 3a for the deciding, and 4a for the changing.

GaE: Growth of architectural Elements

$$GaE = \frac{OaAC}{Number\ of\ functional\ elements} * 100$$

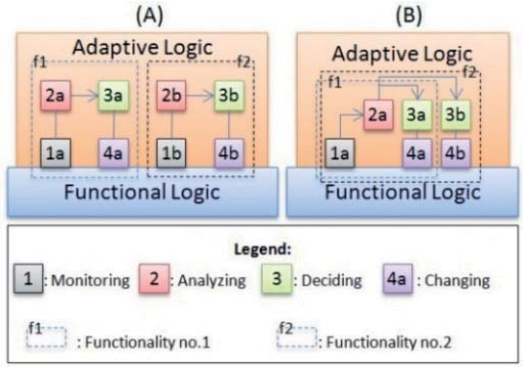


Figure 3. Adaptive functionalities and elements

This metric expresses the percentage growth at the architectural level due to the introduction of adaptivity.

B. Structural Metrics

During the presentation of the structural metrics we use the term *entity* to denote the software units. For example, in an object-oriented system an entity is a class.

The systems addressing runtime adaptivity at the structural level are composed of two types of entities: functional and adaptive. The structural metrics concern three main aspects: separation of concerns, structural growth and personalization.

As for the architectural metrics, the separation of concerns regards two aspects: (1) the separation between the functional entities and the entities ensuring adaptivity, and (2) the separation among the conceptual (types of) entities implementing the four steps of adaptivity. It is expressed through six metrics.

sSCI: structural Separation of Concerns Index

$$sSCI = \frac{Number\ of\ adaptive\ entities}{Number\ of\ functional\ entities\ modified}$$

This metric indicates the degree of dependence between the functional and adaptive entities at the implementation level. It enables the evaluation of the separation of concerns by comparing the number of entities inserted in the adaptive part (which provide exclusively adaptive functionalities) and the number of the functional entities which have been modified (to interact with the adaptive ones). Theoretically, the obtained value should be similar to the one resulted for the aSCI metric. Actually, the two values may be significantly different being determined by the adopted implementation strategy: an approach based on few entities (each implementing more functionalities) or a highly modular one (each entity implementing few functionalities). Hence, the two values may differ from each other for different design methodologies at the architectural and the implementation levels.

sAPI: structural Adaptivity Pattern Index

$$sAPI = Number\ of\ adaptive\ conceptual\ entities$$

This metric indicates the separation of concerns between the main conceptual (types of) entities implementing the four steps of adaptivity at the implementation level. If the value of this metric is four, then the adaptive part of a system defines at least a conceptual entity for each of these four steps. If it is zero, the adaptive entities are totally integrated with the functional one. The values between zero and four suggest that two or more adaptivity steps are provided by the same conceptual entity.

IFLAL: Influence of the Functional Logic on the Adaptive Logic

$$IFLAL = Number\ of\ inputs\ in\ the\ adaptive\ logic$$

IALFL: Influence of the Adaptive Logic on the Functional Logic

$$IALFL = Number\ \ of\ outputs\ from\ the\ adaptive\ logic$$

These two metrics provide information about the role of the adaptive part of a system in its overall functionality. Higher is the value for the IFLAL, stronger is the influence of the application domain or contextual aspects on the adaptive part of a system (see Figure 4-B). Vice-versa, the IALFL metric indicates the degree of influence of the adaptive logic on the functionalities provided by a system (see Figure 4-A). Comparing the values denoted by these two metrics it is possible to determine the degree of influence of one part on the other.

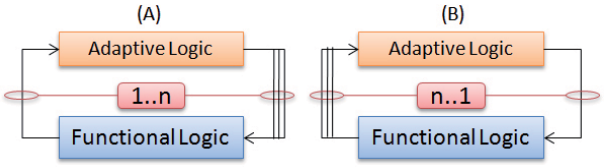


Figure 4. Influence of the adaptive part on the functional one (A) and vice-versa (B)

AvgIFLAL: Average Influence of the Functional Logic on the Adaptive Logic

$$AvgIFLAL = \frac{IFLAL}{Number\ of\ functionalities}$$

AvgIALFL: Average Influence of the Adaptive Logic on the Functional Logic

$$AvgIALFL = \frac{IALFL}{Number\ of\ functionalities}$$

The last two metrics provide information about the average number of inputs (respectively outputs) in the adaptive part for each of the functionalities of a system. The information they provide can be seen as a complexity degree of the provided functionalities based on the influence of the adaptive part of a system.

When AvgIFLAL is significantly greater than AvgIALFL the adaptive logic is strongly related to the application domain and the strategies of the adaptive part consider more factors in their logic than those they can influence in the functional part of a system. Vice-versa, when AvgIALFL is significantly greater than AvgIFLAL we expect that the functional part of a system has different behaviors in the presence of the adaptive part than in its absence due to the strong influence it has from the adaptive entities.

The structural growth regards the number of entities introduced by the adaptive part of a system. It is express through ten metrics.

MsAC: Minimum structural Adaptive Cost

$$MsAC = Minimum\ number\ of\ entities\ for\ adaptivity$$

This metric indicates the minimum number of entities to be added to a system to become adaptive independently of the number of the functionalities it provides. Essentially, this metric expresses the fix cost of adaptivity at the implementation level.

sACF: structural Adaptive Cost per Functionality

$$sACF = Number\ of\ entities\ for\ the\ i^{th}\ functionality$$

This metric indicates the number of entities which should be added to make the i-th functionality adaptive. It may be seen as a variable cost for introducing adaptivity per functionality at the implementation level. The results of this metric may influence the interpretation of the AvgIFLAL and AvgIALFL: minor is the reusability of the entities implementing the adaptive steps, more precisely are the considerations derived from these metrics.

OsAC: Overall structural Adaptive Cost

$$OsAC = MsAC + \sum_{i=2}^{n} sACF$$

The sum between the last two metrics expresses the structural growth in number of entities needed to add adaptivity at the implementation level. The observations made for the OaAC are valid also for OsAC.

AvgGsE: Average Growth of structural Entities

$$AvgGsE = \frac{OsAC}{n}$$

This metric expresses the average growth per functionality at the implementation level due to the introduction of adaptivity. It indicates the average number of entities which have been added for each functionality. Hence, for each functionality it is added 1/n (where n is the total number of functionalities) of the fix costs related to the introduction of adaptive mechanisms in a system.

SDG: Storage Dimension Growth

$$SDG = KB_{withAdaptivity} - KB_{withoutAdaptivity}$$

PSDG: Percentage Storage Dimension Growth

$$PSDG = \frac{SDG}{KB_{withoutAdaptivity}} * 100$$

These two metrics indicate the physical storage growth in kilo bytes, and respectively in percentage, due to the presence of the adaptive mechanisms in a system. Adaptive mechanisms include both adaptive entities and their link with the functional entities.

SDAE: Storage Dimension of the Adaptive Entities

$$SDAE = KB_{AllAdaptiveEntities}$$

PSDAE: Percentage Storage Dimension of the Adaptive Entities

$$PSDAE = \frac{SDAE}{SDG} * 100$$

These two metrics indicate the physical storage growth in kilo bytes, and respectively in percentage, needed to store the adaptive entities.

SDCAF: Storage Dimension of the Connections between Adaptive and Functional parts

$$SDCAF = KB_{withAdaptivity} - (KB_{withoutAdaptivity} + SDAE)$$

PSDCAF: Percentage Dimension of the Connections between Adaptive and Functional parts

$$PSDCAF = \frac{SDCAF}{SDG} * 100$$

These two metrics indicate the physical storage growth in kilo bytes and percentage, due to the entities which have been defined for the link and communication between the functional and adaptive part of a system.

The personalization category of metrics regards the changes which are made on a framework for runtime adaptivity or an adaptive system in order to apply it to an actual case study. These metrics are defined from the developers' point of view and try to capture the effort needed to adapt the framework or a system to other solutions.

It consists of nine metrics.

MpAC: Minimum personalization Adaptive Cost

$$MpAC = \frac{\begin{array}{c}Number\ of\ personalized\ entities\ in\ the\\ adaptive\ part\ for\ the\ 1^{st}\ functionality\end{array}}{\begin{array}{c}Number\ of\ entities\ in\ the\ adaptive\\ part\ for\ the\ 1^{st}\ functionality\end{array}} * 100$$

This metric indicates the percentage of entities which are personalized considering only the minimum number of entities necessary to make a system adaptive. Hence, it is calculated through the number of personalized entities for making one (e.g., the first) functionality adaptive.

If the result is 100%, then it may be assumed that the adaptive entities are totally generic or that the factors which influence the adaptive logic are totally domain dependent due to the fact that all the necessary adaptive entities have been personalized.

If the result is sensibly less than 100%, then it can be considered that the adaptive entities are dependent on general factors which may be considered generic enough to be reused without modifications in many application domains or case studies.

pACF: personalization Adaptive Cost per Functionality

$$pACF = \frac{\begin{array}{c}Number\ of\ personalized\ entities\ in\\ the\ adaptive\ part\ for\ each\ functionality\end{array}}{\begin{array}{c}Number\ of\ entities\ added\ in\ the\\ adaptive\ part\ for\ each\ functionality\end{array}} * 100$$

This metric indicates the percentage of entities which are personalized for each adaptive functionality considering the number of adaptive entities added for this particular functionality in the adaptive part. This metric may be influenced by the order in which adaptive functionalities are added (two or more functionalities may exploit common adaptive entities, and hence these entities are added only for the first inserted functionality).

If the result is greater than 100% it means that the new functionality required the modification of already available adaptive entities which have been added previously for other functionalities.

If the result is 100% or less, then it is considered that a number of adaptive entities equal or less then the number of the added functionalities have been modified (without being able to specify if only new added entities have been personalized).

AvgpACF: Average personalization Adaptive Cost per Functionality

$$AvgpACF = \frac{MpAC + \sum_{i=2}^{n} pACF}{n}$$

This metric indicates the average cost for introduction of adaptive mechanisms per functionality.

MpOC: Minimum personalization Overall Cost

$$MpOC = \frac{\begin{array}{c}Number\ of\ personalized\ entities\ in\ the\\ entire\ system\ for\ the\ 1^{st}\ functionality\end{array}}{\begin{array}{c}Number\ of\ entities\ in\ the\\ adaptive\ part\ for\ the\ 1^{st}\ functionality\end{array}} * 100$$

This metric indicates the percentage of entities which are personalized in the entire system (functional and adaptive) with respect to the minimum number of entities in the adaptive part necessary to make a system adaptive. As in the case of MpAC, it is calculated for the first functionality chosen to be made adaptive.

If the result is equal to the one obtained for the MpAC, then no entity in the functional part has been modified. If these two results differ, then functional entities have been

modified too. Greater is the difference between the values of the two metrics, more significant are the modifications in the functional part of the system.

This metric may be useful during the analysis of the separation between the functional and adaptive parts, as well as of the integration of the adaptive part in a system.

pOCF: personalization Overall Cost per Functionality

$$pOCF = \frac{Number\ of\ personalized\ entities\ in\ the\ entire\ system}{Number\ of\ entities\ in\ the\ adaptive\ part\ for\ each\ functionality} * 100$$

This metric indicates the percentage of entities which are personalized in the entire system (functional and adaptive) with respect to the total number of adaptive entities necessary to provide a functionality.

If the result is equal to the one obtained for the pACF, then no entity in the functional part has been modified (besides the modifications made for one functionality). If these two results differ, then functional entities have been modified too. Greater is the difference between the values of the two metrics, more significant are the modifications in the functional part of the system.

Comparing this metric to the previous one MpOC, usually it can be observed that the modifications performed for the first functionality made adaptive may be greater than those performed for each of the other functionalities (because there may be entities which are used further by all functionalities).

AvgpOCF: Average personalization Overall Cost per Functionality

$$AvgpOCF = \frac{MpOC + \sum_{i=2}^{n} pOCF}{n}$$

This metric indicates the average overall cost for introduction of adaptive mechanisms per functionality. If the result is equal to the one obtained for the AvgpACF, then the functional part is not modified. If these two results differ, then functional entities have been modified too. Greater is the difference between the values of the two metrics, more significant are the modifications in the functional part of the system.

ApOC: Adaptive personalization Overall Cost

$$ApOC = \frac{Number\ of\ personalized\ entities\ in\ the\ adaptive\ part}{Total\ number\ of\ personalized\ entities\ in\ the\ entire\ system} * 100$$

This metric indicates the percentage of the personalization of the adaptive part with respect to the personalization of the entire system (functional and adaptive) to make it adaptive.

DSAI: Domain Specific Adaptivity Index

$$DSAI = \frac{Number\ of\ domain\ influence\ factors}{Number\ of\ total\ influence\ factors} * 100$$

This metric indicates the percentage of the factors specific to the application domain which influence the adaptive part of a system. Higher is this value, higher is the number of personalized entities in the entire system.

pAEF: personalization of Adaptive Entity Functionality

$$pAEF = \frac{Number\ of\ personalized\ functionality\ of\ an\ adaptive\ entity * 100}{Total\ functionalities\ of\ an\ adaptive\ entity}$$

This metric indicates the percentage of functionalities personalized for an adaptive entity. For example, in an object-oriented system this regards the methods signature. If this value is low, then modifications are made mostly inside the functionalities (in the definition of methods and not in their declarations).

C. Interaction Metrics

The purpose of the interaction metrics is to evaluate the variations in the interaction between administrators or users and the adaptive and non-adaptive versions of a system.

MAiAI: Modified Administrator interaction Adaptivity Index

$$MAiAI = \sum \begin{cases} 1\ if\ an\ task\ is\ modified \\ 0\ otherwise \end{cases}$$

This metric indicates the modified tasks which should be performed by the administrator. These tasks are necessary both in the non-adaptive and adaptive versions of the system. The introduction of the adaptive part may have made them more or less complex.

AAiAI: Added Administrator interaction Adaptivity Index

$$AAiAI = \sum \begin{cases} 1\ if\ an\ task\ is\ added \\ 0\ otherwise \end{cases}$$

This metric indicates the new added tasks which should be performed by the administrator after the introduction of the adaptive part. These tasks were not necessary in the non-adaptive version of the system.

RAiAI: Removed Administrator interaction Adaptivity Index

$$RAiAI = \sum \begin{cases} 1 \; if \; an \; task \; is \; deleted \\ 0 \; otherwise \end{cases}$$

This metric indicates the removed tasks which should not be further performed by the administrator after the introduction of the adaptive part. These actions were necessary in the non-adaptive version of the system.

Greater is the RAiAI and/or lower is the AAiAI, more efficient is the introduction of the adaptivity mechanisms from the administration of the system point of view. However, these three metrics do not provide information on the complexity of the administration tasks. This would be very useful to complement the MAiAI metric in order to check whether adaptivity has simplified or not the administration tasks (for those which have not been removed).

The user interaction metrics concern two aspects: the variations in the interaction between users and a system in the absence and presence of adaptivity, as well as the provisioning of the parameters needed for adaptivity.

MUiAI: Modified User interaction Adaptivity Index

$$MUiAI = \sum \begin{cases} 1 \; if \; an \; task \; is \; modified \\ 0 \; otherwise \end{cases}$$

This metric indicates the modified tasks which should be performed by the users. These tasks are necessary both in the non-adaptive and adaptive versions of the system. The introduction of the adaptive part may have made them more or less complex.

AUiAI: Added User interaction Adaptivity Index

$$AUiAI = \sum \begin{cases} 1 \; if \; an \; task \; is \; added \\ 0 \; otherwise \end{cases}$$

This metric indicates the new added tasks which should be performed by the users after the introduction of the adaptive part. These actions were not necessary in the non-adaptive version of the system.

RUiAI: Removed User interaction Adaptivity Index

$$RUiAI = \sum \begin{cases} 1 \; if \; an \; task \; is \; deleted \\ 0 \; otherwise \end{cases}$$

This metric indicates the removed tasks which should not be further performed by the users after the introduction of the adaptive part. These tasks were necessary in the non-adaptive version of the system.

Greater is the RUiAI and/or lower is the AUiAI, more efficient is the introduction of the adaptivity mechanisms from the users' point of view. However, these three metrics do not provide information on the complexity of the user interaction tasks. This would be very useful to complement the MUiAI metric in order to check whether adaptivity has simplified or not these tasks (for those which have not been removed).

UIiI: User Information interaction Index

$$UIiI = \begin{cases} 0 \; if \; all \; parameters \; are \; system \; side \\ 1 \; otherwise \end{cases}$$

This metric indicates if the monitored parameters are available on the system side or they should be gathered through the interaction with the users.

D. Performance Metrics

The performance metrics concern five main aspects related to the usage of the system resources (in terms of RAM and CPU), the response time, the improvement of the response quality in the presence of adaptivity and influence of the performance factors on the adaptive strategies. Usually, these metrics reflect the goals of the adaptive systems.

pRAM: performance RAM

$$pRAM = \frac{RAM \; usage \; in \; presence \; of \; adaptivity}{RAM \; usage \; in \; absence \; of \; adaptivity} * 100$$

This metric indicates the variation of the RAM usage due to the computational overhead introduced by the adaptive part of a system.

pCPU: performance CPU

$$pCPU = \frac{CPU \; usage \; in \; presence \; of \; adaptivity}{CPU \; usage \; in \; absence \; of \; adaptivity} * 100$$

This metric indicates the variation of the CPU usage due to the computational overhead introduced by the adaptive part of a system.

pLatency: performance Latency

$$pLatency = \frac{Response \; time \; in \; presence \; of \; adaptivity}{Response \; time \; in \; absence \; of \; adaptivity} * 100$$

This metric indicates the variation of the system's responses in the presence of adaptivity with respect to the response in the absence of adaptivity.

pQoR: performance Quality of Response

$$pQoR = \frac{\substack{Quality\ of\ response\ in\ presence \\ of\ adaptivity}}{\substack{Quality\ of\ response\ in\ the\ absence \\ of\ adaptivity}} * 100$$

This metric indicates the variation of the quality of the system's responses in the presence of adaptivity. Generally, the obtained value for this metric should be greater than 100% in order to overcome the increments introduced by one or more of the previous three metrics.

pIA: performance Influence on Adaptivity

$$pIA = \begin{cases} 0\ if\ there\ is\ no\ influence \\ 1\ otherwise \end{cases}$$

This metric indicates if the adaptive strategies are influenced by the first three performance metrics. For example, the adaptive part may decide to apply a strategy which uses less RAM or CPU, or which provides a response in less time by paying in the quality of the response (offering a medium, rather than a high quality).

E. Documentation Metrics

The documentation metrics concern two aspects: the comments and the available documentation related to an adaptive system. Their roots are in the general software metrics and they have been interpreted and adapted to provide useful information related to the design of adaptive issues.

CE: Comments per Entity

$$CE = \frac{Number\ of\ comment\ lines}{Number\ of\ entity\ lines} * 100$$

This metric indicates the percentage of the number of comment lines for each adaptive entity. It may be computed for the entities which do not need personalization and those which need personalization (but before their actual personalization). If these two values differ significantly we may suppose that the last category has been predisposed to be personalized and due to the available comments, the personalization may be performed easier.

CF: Comments per Functionality

$$CF = \frac{Number\ of\ comment\ lines}{Number\ of\ functionality\ lines} * 100$$

This metric indicates the percentage of the number of comment lines for each functionality offered by an adaptive entity. It may be computed for the functionalities which do not need personalization and those which need personalization (but before their actual personalization). The considerations for the previous metric hold also for the present one.

CTI: Comments Type Index

$$CTI = \frac{Number\ of\ comment\ lines\ per\ category}{\substack{Number\ of\ comment\ lines\ in\ an \\ adaptive\ entity}} * 100$$

This metric indicates the percentage of the comments of a given type (e.g., auto-generated, formal language, natural language, commented code) considering all the comments in an adaptive entity.

DTI: Documentation Type Index

$$DTI = \frac{Documentation\ quantity\ per\ category}{Overall\ available\ documentation} * 100$$

This metric indicates the percentage of the documentation of a given type (e.g., descriptive, samples, personalization examples for adaptive entities, auto-generated) considering all the available documentation.

F. Miscellaneous Metrics

We have identified three more aspects which should be considered for evaluation of adaptive systems.

ADI: Adaptivity Distribution Index

It regards the distribution of the adaptive elements and entities on the physical nodes of an adaptive system. ADI provides information on the replication of adaptive elements and entities inside a system.

RAMI: Results of Adaptivity Memorization Index

It indicates if the results of each adaptive step are stored temporarily or persistently in the system in order to provide or optimize the adaptive functionalities.

AAI: Adaptive Alternatives Index

It is related to the way in which the various alternatives of adaptivity are provided. Alternatives may be of two types: horizontal and vertical. In the horizontal mode, adaptivity optimizes the usage of the resources to provide the required functionality; while, in the vertical mode, adaptivity optimizes the quality of information to provide the required functionality.

IV. ANALYSIS OF CASE STUDIES THROUGH METRICS

This section analyzes the four case studies introduced in Section II from the adaptivity metrics point of view. Two premises should be made here. First, the case studies are well-defined and thought to outline their adaptivity features. However, several observations have been extracted from

TABLE I. ADAPTIVE ASPECTS RELATED TO FOUR CASE STUDIES

Case Study	Goal of Adaptivity	Adaptivity Type	Main Steps of Adaptivity				Models	Applicable Categories of Metrics
			Monitoring	Analyzing	Deciding	Changing		
Web-based Client Server	Automate administration tasks Performance optimization	Architectural	Number of servers Bandwidth Response time	Constraints evaluation through invariants	Adaptive strategy	Mapping / reflecting the changes at the functional level	Rainbow – definition of specific architectural styles	Architectural separation of concern Architectural growth Administrator interaction Performance Miscellaneous
Video-conference	Performance optimization	Architectural	Bandwidth Gateway cost	Constraints evaluation through invariants	Adaptive strategy	Mapping / reflecting the changes at the functional level	Rainbow - definition of specific architectural styles	Architectural separation of concern Architectural growth Performance Miscellaneous
Adaptive Image Server	Performance optimization Resource usage optimization	Behavioral	Image resolution Bandwidth CPU load Throughput	Adaptive policies guide the analysis of the monitored parameters and determine the actions to be applied The results are stored in a temporary repository to be used directly in identical situations		Modify image	Adaptive Server Framework – personalization of the framework components	Architectural separation of concern Architectural growth Structural separation of concerns Structural growth Personalization User interaction Performance Miscellaneous
AHA!	Enhance functionalities	Content Navigation	User knowledge level	Rule based users' update profiles Rule based adaptation of content Rule based adaptation of navigation		Modify content Modify navigation		Structural separation of concerns Structural growth Administrator interaction User interaction Performance Miscellaneous

their study. Second, the information we used to analyze these case studies consisted exclusively in their description in various articles. Hence, only qualitative observations have been drawn.

Table I summarizes the main adaptive aspects of the four case studies in terms of the goals of runtime adaptivity, the type of the addressed issues, the main conceptual steps of adaptivity, the models or frameworks used to achieve adaptivity, and the categories of metrics which provide meaningful information for their evaluation.

The analysis of the case studies through the adaptivity metrics has lead to the following considerations.

The goals of exploiting runtime adaptivity are of various natures: automate administration tasks, resource usage optimization, or enhancement of functionalities. A recurring goal is related to the performance optimization. The question is what to measure to evaluate performance aspects? The authors of three case studies indicate as one of the performance aspects the response time, which is expressed through the pLatency metric. Only WebCS uses this terminology, while the other two focuses on the bandwidth (which influences the response time). WebCS and VConf are proposed by the same authors. These are simple examples, however they point out the importance of having the same metrics for the same performance issues and moreover, for their evaluation and comparison in different case studies.

Adaptivity may regard various aspects (architectural, behavioral, content and navigation in these cases). Independent of these aspects, it is fundamental to fulfill the separation of concerns metrics at the architectural and/or structural levels. The last two case studies merge the analysis and decision steps in one single step. This is mostly related to the fact that the adaptivity issues are strongly related to the application domain (e.g., specific information as images or learning content) and the factors which are considered in input of the adaptive process.

The separation of concerns and growth metrics at the architectural or structural levels provide information on the modularity, reusability, flexibility, extensibility and scalability of an adaptive system.

Furthermore, in three of the case studies changes are visible at the functional level. The metrics providing information about this aspect are the structural and performance ones. AIS updates also the adaptive knowledge by storing the information resulted from the various adaptive steps in a repository in order to use it in similar cases without performing the same computation again. This characteristic is described through the RAMI index. AIS considers also the computational overhead introduced by the adaptivity part and expressed through the pCPU performance metric.

There are various changes which may be applied to address the same performance issues. For example, in WebCS, to improve the response time an architectural

approach is used: a server is added to the system. This implies that horizontal adaptive alternatives are available (see Figure 5-A). In AIS, the same performance issue is addressed by modifying images, which have different dimensions and hence, may be changed less or more depending on the current request and the status of the system. This implies the availability of vertical adaptive alternatives (see Figure 5-B). As in the case of the AAI index, in the first case, the usage of the resources which provide adaptive functionalities is optimized, while in the second case, the quality of the information needed to provide adaptive functionalities is optimized. In WebCS the administrator tasks have been automated in that the adaptive mechanisms decide whenever a server should be connected or disconnected from the system in order to ensure its performances. In this context, interaction metrics provide information on the efficiency of the adaptivity modules from the point of view of the reduction of the overhead introduced by the interaction with the administrators.

The first three case studies exploit the adaptive concepts (e.g., elements and entities) defined by a general model (e.g., the Rainbow and ASF framework). Moreover, WebCS and VConf are based on a common approach: Rainbow. In these cases the personalization metrics are useful to evaluate the adaptation and usability of the models' concepts in various contexts.

The AHA! approach does not exploit or personalize any model. One of the reasons motivating this characteristics is that adaptivity is specific to the application domain and furthermore, only domain-specific content and its provisioning are adapted. The functioning of AHA! in the absence and in the presence of adaptivity differs significantly: in a non-adaptive scenario the same content is presented in the same way to all users, while in an adaptive scenario users have to be identified and the content representation and provisioning modified. These modifications are properly described through the structural growth metrics, user interaction, and pQoR metric.

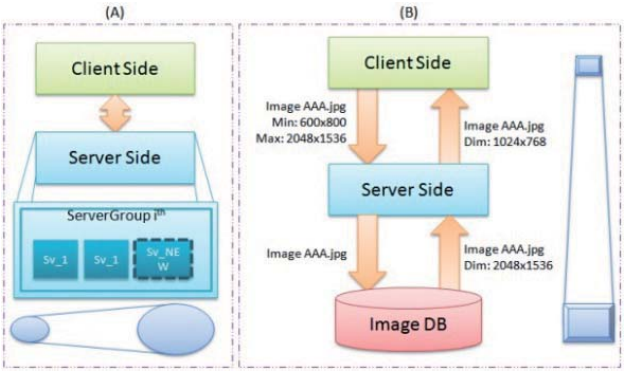


Figure 5. Adaptive alternatives in WebCS and AIS

The last column in Table I lists the categories and subcategories (as shown in Figure 1) of the metrics which are meaningful to evaluate the adaptive features of each case study. The documentation category has been not mentioned because the only documentation we have considered is composed of the articles describing these case studies.

V. Related Work

There are various approaches for the evaluation of runtime adaptivity properties. Typically, they are defined for specific application domains and/or consider a particular perspective on adaptivity.

For example, a methodology for empirical evaluation of adaptive systems is presented in [21]. It considers the use of adaptivity to reduce the complexity of the interaction between users and information systems. Hence, it addresses adaptivity from the users' point of view. The methodology defines six steps to achieve this goal: evaluation of reliability and external validity of input data acquisition, evaluation of the inference mechanism and accuracy of user properties, appropriateness of adaptation decisions, change of system behaviour when the system adapts, change of user behaviour when the system adapts, and change and quality of total interaction. Each of these steps is addressed independently in the context of a framework which enables the evaluation of reliability and external validity of input data acquisition, inference mechanisms and accuracy of user properties, adaptation decisions, and overall interaction (including system and user behaviour and usability).

Or, [19] proposes a set of primary features based on which adaptive hypermedia systems may be evaluated. These features are categorized as follows: adaptation, software quality, software engineering, and technology. This approach considers only a specific type of systems.

Metrics for the evaluation of adaptivity in information systems are introduced in [20], which identifies three generic indexes applicable at the architectural level: element adaptability index (which is 1 for adaptable elements, and 0 otherwise), architecture adaptability index (defined as the sum of all element adaptability indexes divided per total number of elements), and software adaptability index (defined as the sum of the architecture adaptability indexes for all the architecture of a software divided per the total number of architectures of that software). The same authors propose a framework called the Process-Oriented Metrics for Software Architecture Adaptability (POMSAA) [4] to calculate scores for the adaptability of software architectures. The quantitative scores are computed based on the satisficing degree [4] of a non-functional requirement, which in this case regards adaptivity. Both these works consider adaptability only at the architectural level.

A more detailed set of evaluation mechanisms is presented in [11]. This work proposes the evaluation of self-* systems from three points of view: (1) the methodology adopted for their development, (2) the performances offered at runtime, and (3) the intrinsic characteristics of such systems. More specifically, this paper focuses on aspects related to performance, robustness, computational complexity, and decentralization and local algorithms. Even

if this approach is strongly related to the multi-agent domain, it may be adopted for other application domains.

To our knowledge there is no work in the scientific literature addressing the evaluation of the frameworks for adaptivity through metrics.

VI. Conclusions and Further Work

This paper has proposed a set of metrics for runtime adaptivity. These metrics should be considered as a starting point towards the identification and specification of what should it be evaluated and how should it be evaluated in adaptive system.

Our initial work on this topic has been previously presented in [13], [17], [18]. In [17] we have focused our attention on the feasibility of the definition of measurable evaluation mechanisms for adaptive systems, and on the usability of these metrics as design hints in the development process of new adaptive systems and as formal approaches for the evaluation of the existing adaptive systems. A first set of metrics for the evaluation of adaptive systems has been presented in [18]. The aim of this work was to provide a concrete mechanism for the evaluation of the adaptive features of information systems. In [13] we have extended the evaluation of adaptive properties also to the frameworks which provide support for the development of adaptive systems. It is fundamental to understand their personalization and documentation aspects in order to evaluate the effort necessary to develop an adaptive system based on such a framework.

The advantage of such metrics is the specification of a common vocabulary for different design, implementation, and performance issues of adaptivity. They provide a common means for the evaluation of adaptive systems, as well as for the comparison of information systems from the adaptivity point of view.

From the software engineering point of view, the architectural and structural metrics suit best as hints in the analysis and design phases of adaptive systems, as well as concrete mechanisms to evaluate the design of adaptive systems. They provide valuable information about the modularity, maintainability, re-usability, or scalability of adaptive systems. Structural metrics are particularly relevant for the implementation and its evaluation in adaptive systems. The personalization sub-category defines common mechanisms to evaluate and choose an appropriate solution for the issues of the current system in the case a framework or a previous solution should be exploited. The documentation metrics may complement the architectural and structural categories in order to offer additional information on the design of runtime adaptivity. These metrics provide additional information on the quality of the design of adaptive systems and frameworks.

The interaction metrics provide information on the advantages of runtime adaptivity from the administrators and users points of view. This is significantly important in various application domains such as e-learning, finance or healthcare.

On the other hand, the performance metrics provide information on the advantages of exploiting runtime adaptivity from the resource usage and overall systems' quality points of view. They are of a determinant significance for all the actors of adaptive systems.

The miscellaneous indexes capture conceptual and distributed aspects of adaptive systems. They are more related to the deployment and efficiency of such systems.

The metrics proposed in this paper have been identified through a process similar to the reverse engineering by considering the available relevant case studies addressing runtime adaptivity issues. Hence, they regard those aspects which are outlined as advantages of the design and exploitation of adaptivity by the authors of these case studies.

Further work will concern the validation and revision of these metrics by applying them to more case studies. Moreover, we will consider the extensibility of this set of metrics also towards standard software engineering metrics for non-functional properties [9] which may be adopted and adapted for the evaluation of runtime adaptivity.

A future development is related to the application of these metrics from the initial phases of the development of adaptive systems. Hence, they should be considered during the identification and specification of the non-functional requirements regarding the runtime adaptivity properties. In this way, it will be possible to indicate a range of acceptable values which should be satisfied by the final system in order to be successfully deployed and exploited.

References

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison Wesley, USA, 2003

[2] P. De Bra, A. Aerts, B. Berden, D. de Lange, B. Rousseau, T. Santic, D. Smits, and N. Stash, "AHA! The Adaptive Hypermedia Architecture", *Proceedings of the 14th ACM Conference on Hypertext and hypermedia*, 2003, pp. 81-84

[3] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, "*Software Engineering for Self-Adaptive Systems"*, LNCS 5525, Springer, 2009

[4] L. Chung and N. Subramanian, "Process-Oriented Metrics for Software Architecture Adaptability", *Proceedings of the 5th International Symposium on Requirements Engineering*, 2001, pp. 310-311

[5] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based Self-Adaptation with Reusable Infrastructure", *IEEE Computer*, Vol. 37, No. 10, 2004, pp. 46-54

[6] D. Garlan and M. Shaw, "An Introduction to Software Architecture", Technical Report CMU/SEI-94-TR-21, 1994

[7] I. Gorton, Y. Liu, and N. Trivedi, "An extensible and lightweight architecture for adaptive server applications", *Software – Practice and Experience Journal*, Vol. 38, No. 8, 2007, pp. 853-883

[8] J. He, T. Gao, W. Hao, I.-L. Yen, and F. Bastani, "A Flexible Content Adaptation System Using a Rule-Based Approach", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 1, 2007, pp. 127-140

[9] ISO IEC 9126-1 Standard, http://www.iso.org, 2001, June 2010.

[10] G. Karsai, A. Ledeczi, J. Sztipanovits, G. Peceli, G. Simon, and T. Kovacshzy, "An Approach to Self-Adaptive Software Based on Supervisory Control", LNCS 2614, 2003, pp. 77-92

[11] E. Kaddoum, M-.P. Gleizes, J.-P. Georgé, and G. Picard, "Characterizing and Evaluating Problem Solving Self-* Systems", *Proceedings of the ADAPTIVE 2009 Conference*, IEEE Press, 2009, pp.137-145.

[12] L. Masciadri, "A Design and Evaluation Framework for Adaptive Systems", MsC Thesis, University of Milano-Bicocca, Italy, 2009

[13] L. Masciadri and C. Raibulet, "Frameworks for the Development of Adaptive Systems: Evaluation of Their Adaptability Feature Software Metrics", *Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA 2010)*, 2009, pp. 309-321

[14] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing Adaptive Software. Computer", *IEEE Computer Society*, Vol. 37, No. 7, 2004, pp. 56-64

[15] C. Raibulet, "Facets of Adaptivity", *Proceedings of the 2nd European Conference on Software Architecture*, LNCS 5292, 2008, pp. 342-345

[16] C. Raibulet, F. Arcelli, S. Mussino, M. Riva, F. Tisato and L. Ubezio, "Components in an Adaptive and QoS-based Architecture", *Proceedings of the ICSE 2006 Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 65-71

[17] C. Raibulet and L. Masciadri, "Evaluation of Dynamic Adaptivity through Metrics: an Achievable Target?", *Proceedings of the Joint IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture* 2009, pp.341-344

[18] C. Raibulet and L. Masciadri, "Towards Evaluation Mechanisms for Runtime Adaptivity: from Case Studies to Metrics", *Proceedings of the ADAPTIVE 2009 Conference*, IEEE Press, 2009, pp. 146-152

[19] H. Sadat and A. A. Ghorbani, "On the Evaluation of Adaptive Web Systems", *Proceedings of the Workshop on Web-based Support Systems*, 2004, pp. 127-136.

[20] N. Subramanian and L. Chung, "Metrics for Adaptability", *Journal of Applied Technology Division*, 1999, pp. 95-108.

[21] S. Weibelzahl, "Evaluation of Adaptive Systems", *Lecture Notes Computer Science LNCS 2109*, 2001, pp. 292-29

A Quality Criteria Framework for Pattern Validation

Daniela Wurhofer, Marianna Obrist, Elke Beck & Manfred Tscheligi

Christian Doppler Laboratory for "Contextual Interfaces"
HCI & Usability Unit, ICT&S Center
University of Salzburg
Salzburg, Austria
Email: {firstname.lastname}@sbg.ac.at

***Abstract*—Patterns represent an important tool for communicating, documenting, and looking up best practices for both novice and expert system developers and designers. Working in the field of patterns not only requires a well-structured approach to develop new patterns but also a guidance to validate patterns in order to ensure a high quality. Although there are a number of different patterns and pattern languages available, it is still unclear how to validate patterns in a structured way. Within this paper, we aim to fill this gap by introducing a Quality Criteria Framework developed on the basis of existing pattern research. Particularly, five main quality criteria for patterns will be presented and discussed in detail. The idea of our framework is to provide structured guidance for validating patterns in a comprehensive way by using quality criteria. In order to show the applicability of the quality criteria framework in practice, a case study using selected criteria from the framework for validating an existing pattern collection was conducted. This case study showed the appropriateness of our framework for validating patterns and iterating them on the basis of the validation results.**

***Keywords*-patterns; validation; quality framework; validation methods; case study.**

I. INTRODUCTION

This paper aims to give a detailed overview on current research on the validation of patterns and to thoroughly introduce a comprehensive framework for validating patterns based on our previous work presented at the PATTERNS 2009 conference [1]. This work is part of our research towards contextual user experience patterns within the Christian Doppler Laboratory for "Contextual Interfaces". The framework was already applied in a specific context, namely the case study presented in this paper (audiovisual systems) and is currently explored in other contexts. Thereby, knowledge gained on contextual user experience [2] is preserved by using the pattern approach.

In general, patterns are characterised by capturing useful design solutions and generalizing them to address similar problems [3]. Borchers [4] defines design patterns as a structured textual and graphical description of a proven solution to a recurring design problem. This also underpins the reusability of patterns which was especially emphasized by Martin, Rouncefield, and Sommerville, [5]. Tidwell [6] stresses the fact that patterns are neither heuristics nor complete step-by-step descriptions of how to solve a problem but descriptions of best practices.

Since patterns have been introduced in urban architecture in the 1970s by Christopher Alexander [7], they have turned out to be an important tool for communicating, documenting, and looking up best practices for both novices and experts in different domains. According to van Welie [8], documenting and looking up best practices improves the quality of design solutions and can reduce time and effort for designing new projects considerably, provided that the patterns themselves are of high quality. Moreover, patterns facilitate communication between different stakeholders (e.g., designers and programmers) as they support the forming of a collective vocabulary and thus avoid misunderstandings and ambiguities [9].

Patterns have become popular in different domains, involving architecture [10], software engineering (e.g., [11][12][13]), human-computer interaction and interface design (e.g., [14][15][16][6][17][18][19]), ubiquitous computing [20], game design [21], and pedagogics [22].

Despite the broad application range of patterns, there is still a lack of research on the validation of patterns. In particular, there is a lack of consistent quality criteria for patterns and pattern languages as well as appropriate validation methods. According to Dearden & Finlay [23], the evaluation of how useful selected patterns are in practice is important but has been hardly considered up to now. Similarly, McGee [21] claims that there is a need for materials to support the creation and revision of patterns. Although there have been some attempts to validate patterns, these validations have focused on specific application domains or have only regarded selected aspects. Thus, a comprehensive framework for validating patterns in different application areas involving all relevant criteria which account for the quality of a pattern is still missing.

Within this paper we aim to fill this gap by presenting a common and comprehensive validation approach for patterns. By introducing a so-called quality criteria framework, a novel way of validating patterns based on clearly defined quality criteria is described. In order to define a

comprehensive framework, we first collected and analyzed existing criteria, guidelines and requirements for patterns. Based on this extensive desktop research, we developed our own validation framework. To prove the applicability of our framework in practice, we finally applied it for improving an existing pattern collection. Therefore, we conducted a case study with an existing pattern collection. On the basis of quality criteria chosen from our framework, selected patterns of the collection were iterated and improved using both qualitative (interactive pattern workshop) and quantitative (pattern checklist) methods.

The present paper is structured as follows: In the first part of the paper we will introduce related work and discuss existing criteria, guidelines, and requirements for the patterns we use for clustering and developing a comprehensive quality criteria framework. The second part of the paper is dedicated to the development and description of our quality criteria framework, in which each component is presented in detail. Next, we show how we applied the developed quality criteria framework for validating and improving an existing pattern collection. Finally, we discuss insights gathered from the validations done on the basis of our framework and give an outlook for future work.

II. Related Work

Patterns have to be evaluated to prove their quality. However, the question of how to evaluate patterns profoundly is still vague and remains a challenge for those who develop and improve patterns and pattern languages.

A. Quality Criteria for Patterns

There are several collections of criteria, guidelines, and requirements available which aim at defining what makes patterns and pattern languages high-quality. The five presented collections range from a focus on single patterns themselves to an overall view on pattern languages and frameworks, deriving from different pattern domains, e.g., design, human-computer interaction (HCI), and software development. We have analyzed these collections with regard to differences and similarities of the used quality criteria as presented in Table I. Based on the related work, we have clustered the collected criteria for patterns and/or pattern languages and used them as a starting point for our quality criteria framework.

One collection of quality requirements for pattern languages was compiled by Niebuhr, Kohler, and Graf [24]. Based on the challenges they experienced when identifying and developing patterns, they state four successive quality criteria which should be considered. The *"problem fit"* criterion is achieved when a pattern has successfully been identified as appropriate to a design or development problem. A high-quality pattern makes it easy to understand its content, and consequently the idea of the pattern (*"understandability"*). Once the pattern has been discovered and understood, it needs to offer a valid solution to the problem (*"correctness"*). The final challenge for the pattern user lies in realizing the pattern solution to solve the design or development problem (*"concretization"*).

Another attempt to describe requirements for the quality of patterns comes from McGee [21], who described general characteristics of patterns. A pattern is required to be *"operational and precise"* in order to be transferable to a concrete solution and to be *"positive"*, which means that it demonstrates 'good practices' instead of bad ones. Further, a pattern should be *"flexible"* in such a way as to offer several solutions to a problem and *"debatable"*, meaning that it is comprehensible enough to be discussed. Furthermore, a *"testable"* pattern allows an empirical confirmation of improvements through pattern implementation and an *"end-user oriented"* pattern strives for a consideration of end-users' perspectives. McGee [21] also introduced the characteristic *"positive"* for patterns, which we have not considered as valuable as a quality criterion for our framework, since patterns represent best practices and are therefore positive by definition.

Furthermore, Khazanchi, Murphy, and Petter [25] defined the following guidelines for evaluating patterns according to Christopher Alexander's vision of a 'quality without a name'. The *"plausibility"* criterion is related to the consistency of knowledge embedded in the pattern of existing knowledge in the field of design or development. Thus, the pattern has to reach a level of believability among the pattern users. When a pattern achieves the *"feasibility"* criterion it can be operationalized and applied to a problem. Further, the description of a pattern has to be understandable (*"effectiveness"*) which comprises qualities like e.g., comprehensiveness, consistency, and completeness. Consistency is not only necessary among parts of a pattern, but also within patterns which belong to the same problem area (*"pragmatic"*). Furthermore, pattern descriptions need to include *"empirical"* evidence in order to verify the intended pattern output. Finally, Khazanchi, Murphy, and Petter [25] consider a pattern to be *"predictive"* by nature, when it is reliable in its effect every time it is applied.

A great deal of the previously described criteria is examining in detail the structural quality of patterns and pattern languages, but misses a broader view on the context of their usage. Borchers [4] defined a set of requirements for an interdisciplinary pattern language framework for the design of interactive systems, which also includes domain-specific aspects. He emphasizes that a pattern framework requires to be understandable for people from different disciplines (*"cross-discipline readability"*). The involvement of different domains (e.g., HCI, software engineering) additionally leads to the necessity of a *"domain-independent, uniform, well-defined format"* for pattern languages. Again, *"empirical evidence"* of the pattern is regarded as important to prove the pattern's validity. Further, the collection of patterns should

Table I
COLLECTION OF QUALITY CRITERIA FOR PATTERNS AND/OR PATTERN LANGUAGES BASED ON A LITERATURE REVIEW AS WELL AS THE DERIVED CRITERIA USED WITHIN THIS PAPER (ON THE RIGHT COLUMN).

Niebuhr et al. (2008)	**McGee (2007)**	**Khazanchi et al. (2008)**	**Borchers (2001)**	**Dearden et al. (2008)**	***This Paper***
Problem Fit					*Findability*
			Domain-appropriate, design-supporting hierarchy		
		Pragmatic			
Understandability	Debatable	Effectiveness			*Understandability*
	Flexible				
			Cross-discipline readability		
			Design dimension coverage		
			Domain-independent, uniform, well-defined format		
Concretisation	Operational and precise	Feasibility		Generative design	*Helpfulness*
			Lifecycle integration		
Correctness	Testable	Empirical	Empirical evidence		*Empirical verification*
		Predictive			
		Plausibility			*Overall acceptability*
	End-user oriented				
	Positive				
				Empowering users	
				Life-enhancing outcomes	

be arranged in a hierarchical order, following the logic of the design/development process (*"domain-appropriate, design-supporting hierarchy"*). The next criterion, *"design dimension coverage"*, is used to consider domain-specific dimensions for the pattern language. The descriptions of interaction patterns, for example require the inclusion of a temporal dimension as an important characteristic of interactions. Furthermore, a pattern language framework requires to give hints on how it can be integrated into the software development lifecycle (*"lifecycle integration"*).

The last criteria collection of Table I refers to the stakeholders who deploy patterns. The two main stakeholders are usually designers and developers who have to be supported by patterns in specific ways [26]. In addressing a participatory design approach, Dearden, Finlay, Allgar, and McManus focused on the suitability of pattern languages as design tools for users acting as non-professional designers [27]. In other words, patterns and pattern languages do not only have to be valuable for professional designers and developers, but also be comprehensible for non-professionals. In line with the position of Christopher Alexander, Dearden, Finlay, Allgar, and McManus discussed three criteria for evaluating patterns. First, pattern languages have to be written in a way which enables users to generate complete designs (*"generative design"*). Second, the pattern language needs to be valuable for *"empowering users"* to participate in a design process and third, the deployment of the pattern should lead to *"life-enhancing outcomes"* for the users.

As shown in Table I, quality criteria of patterns and/or pattern languages from different sources and theories sometimes overlap, and sometimes they put their focus on different aspects.

In this paper, we aim at a conflation of the different aspects into a unified quality criteria framework, which should aid the validation of patterns and pattern languages and thereby the iterative development/improvement of the patterns. Based on the related work we have identified five types of quality criteria, which are listed in the right column of Table I and discussed in detail in the following sections. A further challenge is how to apply these criteria for the validation of patterns and pattern languages, namely finding the right methodological validation approach.

B. Validation of Patterns

For validating patterns against certain quality aspects, mainly two approaches are considered:

1) Expert/stakeholder based evaluation of patterns fusing for instance heuristics, checklists, workshops, etc.
2) Practical usage and evaluation of patterns with stakeholders, such as designers and developers.

The first approach is based on heuristics, guidelines, and peer-reviewing. An established way of investigating the quality of patterns together with peers (designers, developers of a system) is to conduct shepherding and writer's workshops (see [28][29]). Specific requirements and as well as checklists were also used for validating patterns.

Borchers [4] evaluated whether the developed interaction pattern framework meets a set of requirements. In order to get insights about the didactic usefulness of patterns, he

additionally distributed a survey to HCI design students. This survey revealed information about the amount of memorized patterns as well as the patterns' usefulness for their project and for reuse in future design projects [4]. It turned out that the students considered patterns as useful and easy to use. However, a constraint of this study is the fact that it lacks comparison with other types of design advices.

Similar to Borchers [4], McGee [21] provided his design students with a game design pattern guideline as well as a so-called evaluation checklist in the form of questions. These materials were intended to support the pattern creation as well as to find weaknesses of existing patterns. In general, they can support developers in creating and improving innovative games.

Dearden, Finlay, Allgar, and McManus [27] evaluated the usefulness of patterns as tools for participatory design. Therefore, they defined three criteria and investigated these criteria by involving six participants in a participatory design task. According to the authors, the results of their study show that patterns can have a benefit in empowering users to participate in the design process. A comparison with alternative methods was not made.

The second approach of empirical evidence is based on many researchers' claim (e.g., [30]) that it is important to put patterns into use to judge their quality. For example, Chung et al. [20] conduced a controlled study with designers to evaluate the helpfulness of design patterns for ubiquitous computing. Two groups of designers had to complete tasks, one with and the other one without the help of design patterns. Afterwards, the results from both groups were analyzed comparing the quality differences in the design output and investigating the usefulness of the patterns for design. From their findings the authors conclude that the patterns supported both new and experienced designers who are not familiar with ubiquitous computing. In particular, their patterns facilitated the generation and communication of ideas and avoided design problems early in the design process.

Cowley and Wesson [31] conducted an experimental study on the usefulness of patterns, investigating the use of design patterns in comparison to the use of guidelines. The task was to evaluate and redesign an existing website. For this task, an experimental group used a selection of patterns, whereas a control group used guidelines similar to the selected patterns. Based on their preliminary results, the authors concluded that developers were more positive about design patterns than guidelines with respect to potential for evaluation, redesign, and new design.

Kotzé, Renaud, and Biljon [22] compared the effectiveness of patterns and anti-patterns in education. In order to identify the differences, they conducted two studies. According to their findings the authors claim that patterns are easier to learn from than anti-patterns. Thus, they consider the pattern-approach as more promising for educational purposes. In a more general way, their results can also be interpreted as evidence that negatively framed guidance is harder to learn from than positively framed guidance.

In a qualitative case study, Segerståhl and Jokela [32] investigated and compared the usability of two popular pattern collections. In the practical context of an industrial development project, they explored and compared Tidwell's 'Common Ground" collection, and Van Welie's pattern collection. As a result of their study, the researchers gave suggestions on how to improve collections of design patterns and thus make them easier to use.

In our paper we address the challenge of how to apply the criteria for the validation of patterns, namely finding the right methodological validation approach, and to validate selected UX patterns using our quality criteria framework.

III. Quality Criteria Framework

In order to have a theoretical basis for the validation of patterns, we developed a quality criteria framework based on existing research as presented in the previous section. The framework aims to summarize and extend knowledge in this area and therefore represents a "meta-view" on what constitutes a high-quality pattern.

As shown in Table I, five superior quality criteria representing the most important characteristics of high-quality patterns were defined. The following superior criteria were identified for the quality criteria framework and further divided into sub-criteria if appropriate - see overview in Figure 1. Clearly, the present categorization is not always selective, i.e. some sub-criteria could also be subsumed under another criterion. This particularly applies to the sub-criteria of understandability and helpfulness. For example, the sub-criterion problem-centeredness (subsumed under the criterion understandability) could also be subsumed under the criterion helpfulness. In the following sections, each criterion will be described in more detail.

A. Findability

Our criterion called *"findability"* states that a pattern has to be found easily and quickly within a pattern collection/pattern language. It is based on the assumption that if it takes too much time or effort for a potential pattern user to find a suitable pattern for a specific problem, the adoption of patterns fails already at the beginning. Therefore, the fact that a pattern can be easily found within a set of patterns seems to be an essential indicator for the quality of a pattern collection/language. When investigating this criterion, it could, for instance, be checked if the patterns of a pattern collection/pattern language are organized in a hierachical manner, guiding the user top-down to a suitable pattern.

The findability criterion is in line with a requirement for an interdisciplinary pattern language framework defined by

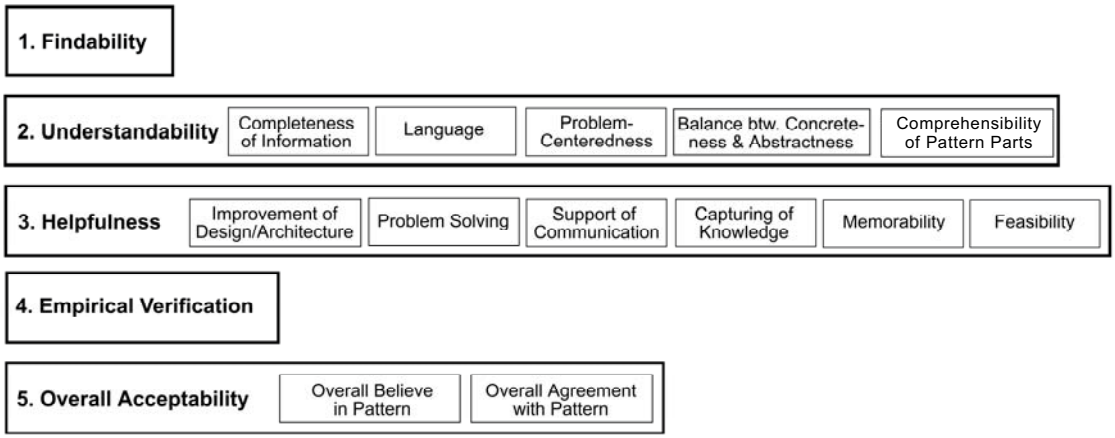


Figure 1. Overview on Components of the Quality Criteria Framework.

Borchers [4], namely the requirement asking for a "domain-appropriate, design-supporting hierarchy". Similarly, Khazanchi, Murphy, and Petter [25] consider the consistency of a pattern within other patterns in a problem class as "pragmatic", and Niebuhr, Kohler, and Graf [24] mention a quality requirement for patterns called "problem fit". The identified *"findability"* quality criteria stands for its own and does not comprise sub-criteria.

B. Understandability

Our criterion *"understandability"* deals with the fact that the pattern must be easily understood by its users. Ensuring the comprehensibility of every pattern part (name, problem, forces, etc.) improves the applicability of the pattern in practice. For assessing this criterion, it could be asked if one finds the name of the pattern meaningful and can figure out the main idea of the pattern (when reading its name). This mainly addresses the quality of the sub-criterion comprehensibility of pattern parts (see enumeration below).

Niebuhr, Kohler, and Graf [24] state that the wording and notation of the pattern description must be understandable in order to successfully identify and apply the pattern. McGee [21] indicates that a pattern should be debatable as well as flexible in the sense that there is more than one solution. According to Khazanchi, Murphy, and Petter [25], a guideline for evaluating patterns is to prove the pattern's effectiveness. In line with this, Borchers [4] poses the requirements of a "cross-discipline readability", "design dimension coverage", and "domain-independent, uniform and well-defined format". In order to describe the understandability criterion as well as possible and thus provide a basis for the operationalization of this criterion, we defined the following sub-criteria:

a) *Completeness of Information*
A pattern should contain all relevant description of forces, problems, solutions, and examples to clarify its notion. The quality of a pattern therefore depends on its completeness. A pattern should be considered as "complete" when the necessary information is given in the pattern.

b) *Language*
A pattern should use a language which is easy to understand. For example, the terms used should be well-known and the sentences should not be too complex. Overall, patterns should be written in a way which is acceptable and appealing to every user, regardless of the discipline he comes from. The clearness of a pattern as well as a well-readable writing style are thus indicators for the quality of a pattern/pattern language.

c) *Problem-Centeredness*
A pattern should be centered around a problem. Therefore, all parts of a pattern (e.g., name, forces, solution) should be derived from the problem. For example, the relationship between the problem and the solution should be clear. A pattern in which all parts are related to the problem description therefore represents a pattern of high quality.

d) *Balance between Concreteness and Abstractness*
A pattern should neither be too abstract nor too concrete. If it is too abstract, one can not figure out how to apply the pattern to other applications/systems. If it is too concrete, the solutions can not be generalized. A high-quality pattern should therefore have a good balance between concreteness and abstractness.

e) *Comprehensibility of Pattern Parts*
All parts of a pattern description should be comprehensible to the pattern users. One should know what

is meant by them. For example, the name of the pattern should be meaningful so that the main idea of the pattern can be figured out instantly. The stated forces should provide enough background information, and the context of application should be clear. The provided solutions should be concrete enough and should not impose new questions. The examples given should be comprehensible and plausible. A high-quality pattern should therefore be characterized by a high comprehensibility of each single pattern part.

C. Helpfulness

This category implies that the pattern has to be (or is supposed to be) helpful for the pattern user. For being helpful, the implementation of the pattern has to be feasible for the pattern user, meaning that the pattern description gives the user sufficient information about how to realize the pattern in practice. The category helpfulness can be further divided into "subjective" and "objective" helpfulness, differentiating between the supposed and the actual helpfulness (objectively measured by case studies, etc). Therefore, this category is based on the assumption that if the pattern/pattern collection is not supposed to be helpful, it will not be used (subjective helpfulness) or that if the pattern/pattern collection has not been helpful, it will not be reused (objective helpfulness). The helpfulness criterion could be investigated for example by letting participants summarize the main content of a pattern which was presented to them some time ago, in one sentence. The results give insight about the quality of the sub-criterion memorability (see enumeration below).

In their paper, Khazanchi, Murphy, and Petter [25] refer to this characteristic as "feasibility". Similarly, Niebuhr, Kohler, and Graf [24] identify the process of transferring a rather abstract pattern description to a concrete solution as "concretization". Borchers [4] claims that a pattern language has to specify a way how the patterns can be integrated in the development lifecycle. McGee [21] characterizes a good pattern as "operational and precise", and Dearden, Finlay, Allgar, and McManus [27] state that pattern languages should support "generative design".

In order to describe the helpfulness criterion as comprehensive as possible and thus provide a basis for the operationalization of this criterion, we defined the following sub-criteria:

a) *Improvement of Design/Architecture*
   A pattern should serve as a design or development aid. With the help of a pattern, the development of new applications and the improvement of existing applications is supported. Therefore, the high quality of a pattern is indicated by the fact that the pattern helps to improve the design or development of systems (depending on the application area of the patterns).
b) *Problem Solving*
   A pattern should be able to provide best practices and solutions to common problems. Upon knowing proven solutions or best practices beforehand, one can avoid certain problems. Therefore, a pattern which helps to avoid common problems represents a pattern of high quality concerning this criterion.
c) *Support of Communication*
   Designers, developers, and researchers do not always speak the same "language". Patterns should therefore serve as a common ground for discussions about design and development issues. A pattern of high quality should therefore provide a common basis for designers, developers, and researchers and thus support (interdisciplinary) communication.
d) *Capturing of Knowledge*
   A pattern represents a tool for capturing previously gained knowledge. The knowledge described in a pattern should appear relevant to the pattern user. A pattern which captures relevant knowledge about its application domain thus represents high quality regarding this criterion.
e) *Memorability*
   A pattern has to be easy to remember in terms of both recognition and recall. When talking about a pattern, its content should be memorized thoroughly in order to support efficient communication and usage of the pattern. A pattern whose main idea can be retrieved in a quick and easy way therefore represents a pattern of high quality.
f) *Feasibility*
   A pattern should be easy to realize or implement in practice. In order to support the right implementation of a pattern, particularly the solution must be clear for the pattern user. A pattern which can be easily applied in real situations accounts for the high quality of a pattern.

D. Empirical Verification

Our criterion *"empirical verification"* describes the fact that a pattern is approved by empirical data. This can be either achieved by creating patterns which are based on results of empirical studies or by verifying existing pattern collections empirically.

We claim that an empirically verified pattern is of higher quality than a pattern which is "only" based on a person's experiences and observations. For example, if there is empirical evidence which approves the "correctness" of the pattern, the quality of the pattern is high.

Niebuhr, Kohler, and Graf [24] ask for empirical or theoretical evidence in order to ensure the "correctness" of a pattern. In line with Niebuhr et al., McGee [21] claims that a pattern should be "testable", i.e. offer the possibility to empirically test the effects of using a certain pattern. Khazanchi, Murphy, and Petter [25] claim that patterns should have an empirical nature in order to make them

verifiable. Moreover, they claim that a pattern should be "predictive", meaning that it produces the same general effect every time it is applied. Borchers [4] points out that the examples given within a pattern should contain empirical evidence of the validity of the solution whenever possible. For this criteria we defined no further sub-criteria.

E. Overall Acceptability

The criterion *"overall acceptability"* describes to what extent a pattern user believes in the pattern, meaning how much he agrees with its content. This category is based on the assumption that if a potential pattern user does not agree with the content of a pattern at all (for example because it completely conflicts with previous experiences), he will not accept the pattern and thus will not use it. Therefore, the quality of a pattern is also affected by an individual's subjective acceptance of a pattern/pattern language. In order to investigate the overall acceptability of a pattern, one could ask if the reader of a pattern finds himself nodding in agreement as he reads the pattern description. Assessing this question would be an indicator for the sub-criterion "overall agreement with pattern" (see enumeration below).

Khazanchi, Murphy, and Petter [25] refer to the term "plausibility", which means that a pattern should be coherent and consistent with the knowledge of a particular domain. According to McGee [21], a pattern should be "end-user oriented", meaning that not only developers or designers appreciate a pattern, but also end-users who interact with the system to be developed/designed. In order to describe this criterion in a more focused way and thus provide a basis for the operationalization of this criterion, we defined the following sub-criteria:

a) *Overall Believe in Pattern*
   This sub-criterion deals with the overall believe in a certain pattern. A high belief in a pattern represents a high (subjectively experienced) quality of a pattern.
b) *Overall Agreement with Pattern*
   This sub-criterion deals with the fact that a user should be convinced of a pattern, i.e. the user should "find himself nodding in agreement as he reads the pattern description" [21]. High (subjectively experienced) quality of a pattern is therefore affected by a high overall agreement with its content.

Based on the framework and its criteria defined above, a case study showing its practicability was conducted.

IV. Case Study UX Patterns: Pattern Validation based on the Framework

In the three-year CITIZEN MEDIA research project, focusing on the user experience (UX) of audiovisual networked applications, we used the pattern approach to develop so-called user experience patterns [33].

Our UX patterns are intended to show best practices for common problems in the area of audiovisual systems, providing designers and developers of audiovisual systems with proven solutions on how to improve a user's experience when interacting with an audiovisual system.

Based on user evaluation data collected in three different European testbeds (Germany, Austria, Norway) involving over 8000 users, we developed about 30 UX patterns. An actual version of the UX patterns can be found on the UX pattern website[1].

In Figure 2 the iterative development process of our UX patterns is visualized. An initial pattern collection was defined based on the results of the first evaluation phase of the research project. This collection of patterns was then iterated by conducting a writer's workshop with researchers, resulting in an extended UX pattern collection. Another iteration was made on the basis of new results achieved during the second evaluation phase as well as based on feedback given by an independent expert, resulting in a revised UX pattern collection.

Next, we wanted to make a validation of our pattern collection in order to improve its quality. However, we had to realize that there is still a lack of a common validation approach for patterns. Therefore, we aimed to fill this gap by introducing a comprehensive quality criteria framework for validating patterns in general. By means of this quality criteria framework we made two more iterations of our UX pattern collection, showing that this framework is applicable in practice.

In the following, the case study should exemplify how our quality criteria framework allows the structured operationalization and investigation of a pattern's/pattern language's quality. The following UX patterns were selected for validation from our collection of 30 UX patterns:

- Pattern 1: Self Presentation
- Pattern 2: Fun Factor
- Pattern 3: Initial Support
- Pattern 4: Real-Life Integration
- Pattern 5: Privacy Management

In order to describe the application of the framework in practice, we shortly present the two methods we used - the pattern workshop and the pattern checklist - following the expert/stakeholder based validation approach (see Section II, B). Table II gives an overview on the two conducted validation sessions.

Due to time constraints during the validation sessions, we focused on the (for us) most important criteria to improve the existing UX patterns. Moreover, not all components of our quality framework were applicable in the sessions. For instance, exploring "findability" or "empirical verification" require practical usage of the patterns.

[1] http://hciunit.org/uxpatterns/

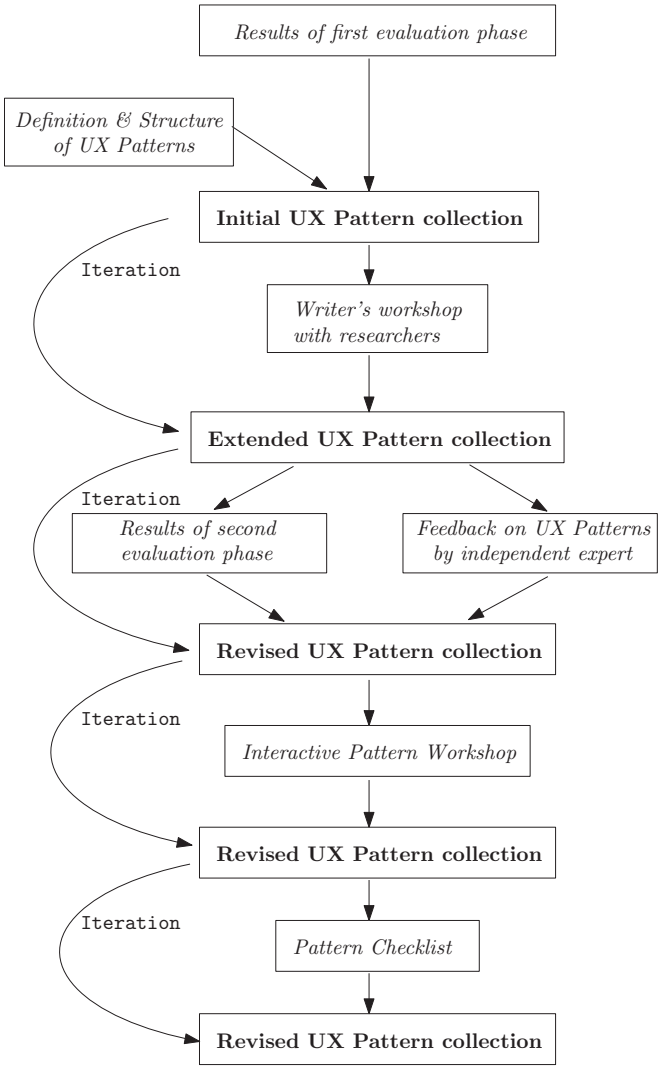


Figure 2. Steps in the UX Pattern Creation Process.

1) Interactive Pattern Workshop: In order to validate and improve the quality of our UX patterns, an interactive pattern workshop was conducted in April 2009.

Set-up: For the validation of the UX patterns, five patterns were selected from the collection and analyzed in detail during the workshop. Figure 3 presents the criteria from the quality criteria framework which were chosen to be validated in the workshop. Thus, the UX patterns were validated with regard to their understandability, helpfulness and overall acceptability.

The workshop was conducted at a stage in the pattern development process where all defined elements of a UX pattern description were available (name, problem, solution, examples). Familiarity with media design processes as well as good command of English were defined as prerequisites for the workshop participants. The participants of the workshop were six multimedia design students (5 females, 1 male) with a mean age of 22. The participants had about five years of design experience on average. Four of the participants did not know design patterns, and two of the participants knew design patterns but never used them so far.

Procedure: For each of the five patterns, the following questions were addressed in detail:

1) Is the pattern easy to understand? Especially: Is the pattern comprehensible?
2) Is the pattern helpful for the designers? Especially: Is the pattern easy to remember?
3) Do the participants accept the presented patterns?

In order to address the questions presented above, the following procedure was deployed: At first, the participants were welcomed and invited to shortly introduce themselves. Next, the workshop leaders gave a short introduction about the workshop goals, the role of patterns within the design process, and the specific role of UX patterns. Then, the workshop participants were asked to do several exercises, addressing the questions presented above.

The first question presented above (understandability) aims at investigating if the wording and the descriptive text of the UX pattern are easy to understand by the designers. Therefore, the comprehensibility of the UX pattern name was investigated in detail using the so-called "name guessing exercise". In general, the name of a pattern should be short and meaningful, expressing the aim of the pattern clearly. To investigate the comprehensibility of a pattern's name, the participants were presented the name of the pattern and then had to write one sentence about its supposed aim. After guessing about the aim, the actual aim of the pattern was presented, and the naming as well as suggestions for improvement of the naming were discussed with the participants.

Next, the comprehensibility of the other parts of the UX pattern description was evaluated by the participants. Therefore, the participants were given a so-called "comprehensibility questionnaire" (see Figure 5) to rate the comprehensibility of the pattern parts (problem, forces, solution) on a five-point scale (from "absolutely agree" to "don't agree at all"). The participants had to read through the pattern descriptions displayed on the beamer and fill in the comprehensibility questionnaire. After each pattern round, the incomprehensible parts were discussed and suggestions for improvement were collected. Additionally, the questionnaire contained two items asking the participants for an overall rating of the pattern (see Figure 7) addressing the overall acceptability (third question presented above).

In order to address the memorability of the selected UX patterns (second question presented above), a so-called "remembrance exercise" was conducted. Therefore, the participants were handed out a sheet of paper containing

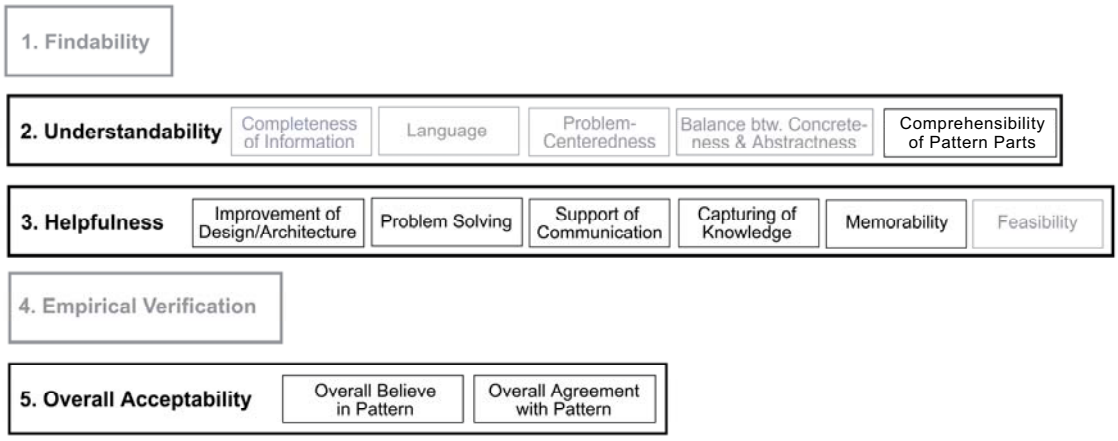


Figure 3. Overview on Components of the Quality Criteria Framework covered by the Workshop.

Table II
OVERVIEW ON THE VALIDATION SESSIONS OF THE CASE STUDY

Validation method	No. of participants	Participants' expertise	Type of feedback	Date of conduction
Interactive Pattern Workshop	6	Multimedia Design Students	Qualitative	April 2009
Pattern Checklist	6	Computer Science Students	Quantitative	June 2009

(again) only the name of a pattern. Having the name of one of the five selected UX patterns, the participants were requested to write down the main characteristics of the pattern in one or two sentences. To get insights into the supposed helpfulness in general, an open discussion about the participants' opinion on the helpfulness of the presented UX patterns was conducted. Amongst others, the participants were asked to discuss about the question if patterns can support designers in giving users a more positive experience and to describe the advantages and disadvantages of patterns. Moreover, possible difficulties in the usage of patterns as well as the usage of patterns for real application design were discussed. Additionally, the participants were asked to compare patterns with any other type of design advice (design guidelines, UI Design Principles) to judge their value. Furthermore, suggestions for improvement were also subject of the discussion.

2) Pattern Checklist: In order to validate the quality of the UX patterns a second time and thus further improve their quality, a follow-up validation session using a so-called "pattern checklist" was conducted in June 2009.

Set-up: Based on the first validation (interactive pattern workshop), a checklist covering selected criteria from the quality criteria framework was composed; the selected criteria are shown in Figure 4. For this follow-up validation, the same UX patterns as in the interactive pattern workshop were used. Overall, the UX patterns were validated with regard to their understandability, helpfulness, and acceptability. At the point of time when this follow-up validation was conducted, the UX pattern description was complete and already iterated on the basis of the first validation (see Figure 2). In order to cover the target group of designers as well as developers, the intended participants were developers this time (first validation was conducted with designers). As the UX patterns are written in English, another prerequisite for the participant selection was a good command of English. The follow-up validation was then conducted with six students of computer science.

Procedure: The main research question of the follow-up validation was if the presented UX patterns would meet the selected quality criteria. In more detail, the following questions were addressed:

1) Is the pattern easy to understand? (Covering all sub-criteria)
2) Is the pattern helpful for developers? (Covering sub-criteria a)-d))
3) Do the participants accept the presented patterns?

In order to address these questions, a checklist was composed. This checklist was intended to find quality problems in (selected) patterns, similar to a traditional heuristic evaluation in the field of usability engineering. By reading the pattern description and going through the heuristics, the evaluators (participants) should judge the patterns' compli-

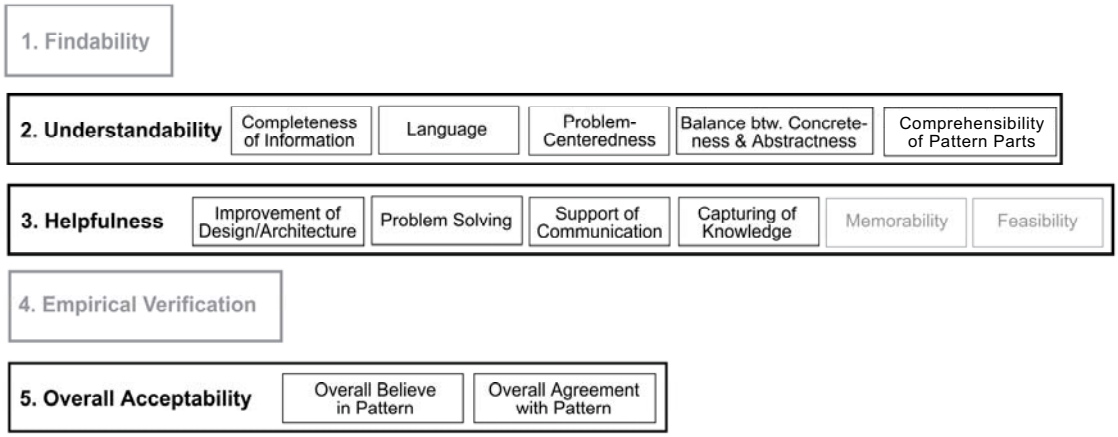


Figure 4. Overview on Components of the Quality Criteria Framework covered by the Checklist.

ance with the quality criteria.

The checklist was organized in the following manner (see Figures 5, 6 and 7): The first part of the checklist (see Figure 5) contained the comprehensibility questionnaire already used in the workshop and was intended to give insights about the comprehensibility of all pattern parts. The second part of the checklist (see Figure 6) covered all other sub-criteria selected from the framework. Each item of the checklist was intended to cover one single quality criterion. The items of the checklist were formulated like heuristics and the agreement with each item had to be indicated on a five-point rating scale (ranging from "absolutely agree" to "don't agree at all"). Again, the acceptance of the presented patterns was evaluated via the third part of the checklist (see Figure 7) .

All parts of a pattern description should be comprehensive to the pattern users. One should know what is meant by them.

	absolutely agree	rather agree	neutral	rather don't agree	don't agree at all	don't know
The **name** of the pattern is meaningful to me. I can figure out the main idea of the pattern.	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
The stated **problem** is clear to me.	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
The stated **forces** provide me enough background information.	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
I know to which **context** the pattern is applicable.	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
The provided **solutions** are concrete enough and don't impose new questions.	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
The given **examples** are comprehensible and plausible.	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]

Figure 5. Part one of the Pattern Checklist - Comprehensibility Questionnaire.

The procedure of the validation session was the following: First, there was an introduction phase. During this phase, the participants were informed about the goals of the validation, the pattern approach and the specific role of UX patterns. Furthermore, the quality criteria were explained so that the participants were familiar with them before the validation started. Then, the participants were asked to introduce themselves, what they study and how much experience and knowledge they have with patterns.

Next was the pattern review phase. In this phase, the participants were given one of the five selected patterns as well as the pattern checklist. The participants were then asked to rate how much they think the quality criteria applied to each of the patterns. In case of a low rating they were asked to further state a reason. This phase was followed by the discussion phase. After each pattern review, the review ratings per pattern (and found problems) were discussed in the group. Finally, the validation session was closed with an overall discussion on the presented patterns.

3) Results and Implications of the Validations: The two approaches for measuring the quality of patterns with regard to certain aspects showed that the quality criteria framework turned out as a valuable method for validating selected UX patterns. Both validations yielded consensus based quality judgements of the selected UX patterns as well as suggestions for improvement of the presented patterns. Both methods complemented each other in a good way and were useful for improving the UX patterns. The workshop yielded more qualitative data and focused on discussion, whereas the checklist enabled a more structured feedback for further refinements of the patterns. The iterative deployment of both methods proved to be helpful, as feedback from the workshop could be integrated into the checklist.

At this point, we will only present the most important implications for the development and restructuring of the UX patterns based on the validations. One issue arising

A pattern should contain all relevant description of forces, problems, solutions and examples to make it clear to the user. For example all relevant forces should be considered. **I would consider the pattern as "complete", meaning that the necessary information is given in the pattern.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know
A pattern should use a language which is easy to understand. For example, the terms used are well-known and the sentences are not too complex. Overall, patterns should be written in a way which is acceptable and appealing to every user (designer, developer ...). **The "language" of the pattern is clear to me. The style in which the pattern is written is well-readable to me.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know
A pattern should be centered around a problem. Therefore, all parts of a pattern (e.g. name, forces, solution) should be build on the problem. For example, the relationship between the problem and the solution is clear. **I think the pattern is problem-centered, i.e. all parts (e.g. name, forces, solution) are related to the problem description.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know
A pattern should neither be too abstract nor too concrete. If it is too abstract, one can't figure out how to apply the pattern to other applications/systems. If it is too concrete then the solutions can't be generalized. **I think that the balance between concreteness and abstractness is good.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know
Overall, a pattern should serve as a design/development aid. With the help of a pattern, the development of new applications and the improvement of existing applications are supported. **I think that the presented pattern helps to develop better A/V systems with regard to the user's experience.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know
The pattern should be able to provide best practices and solutions to common problems. When knowing these solutions beforehand, one can avoid these problems. E.g. by applying this pattern in the development of A/V systems, one avoids the problem of users not experiencing themselves as part of a community. **I think that the presented pattern makes the user's experience while using the system more positive. Common usage problems are avoided.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know
Designers, developers and researchers do not always speak the same "language". Patterns serve as a common ground for discussing about design and development issues. **I think the presented pattern supports the communication of designers, developers and researchers by providing a common basis.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know
A pattern represents a tool for capturing knowledge gained before. Knowledge described in a pattern should appear relevant to the pattern user. **I think the presented pattern captures relevant knowledge about user experience.**	[ ] absolutely agree [ ] rather agree [ ] neutral [ ] rather don't agree [ ] don't agree at all [ ] don't know

Figure 6. Part two of the Pattern Checklist - Heuristics for Understandability and Helpfulness.

Overall, do you "**believe**" the Pattern? [ ] Yes [ ] No [ ] Don't know	Do you find yourself **nodding in agreement** as you read the Pattern description? [ ] Yes [ ] No [ ] Don't know

Figure 7. Part three of the Pattern Checklist - Acceptance Questions.

from the "guessing exercise" and "remembrance exercise" as well as from the discussions was the naming of the UX patterns. The participants indicated that the naming often did not clearly express the aim of the pattern, or that the meaning was not clear. Especially verbs in the name of the patterns (for example "Provide Personal Information") were often confusing for the participants. A poor rating of the comprehensibility of the pattern name always came along with a bad performance of the participants in the remembrance exercise, thus confirming that the name did not express the meaning sufficiently. Based on the participants' suggestions for improvement, their ratings as well as their comments on the questionnaires, the names of all patterns were iterated and verbs were removed from the pattern names.

The "problem" part of the UX patterns was also iterated in detail after the validations. The participants mainly disliked that the problem statement was too general or abstract. Thus, iterated versions of the patterns contain a more concrete problem description.

The overall agreement with the presented patterns was rather good. This confirms the results of the discussions, where the participants stated that they liked and understood the structure and the content of the patterns.

Concerning the helpfulness of the patterns, the participants first did not seem to be convinced. However, after going through some patterns, they changed their opinion and recognized patterns as an important tool for supporting the design/development process. The suggestion of the participants to group the patterns according to more general problems was realized after the first validation.

A. Lessons Learned from the Case Study

When applying our framework in practice, its structure (classification in criteria and more specific sub-criteria) turned out to be helpful for the development of evaluation materials. It allowed us to pick out relevant criteria, which we wanted to investigate within the validation sessions, and supported a more creative approach for investigating specific criteria. For example, we decided to investigate the quality of our patterns regarding their memorability, i.e. how easy they are to remember (sub-criterion five of the criterion helpfulness). The precise definition of memorability given within the framework brought us to the idea of developing a so-called "remembrance exercise". As we benefited from such a structured approach, we will also base the development of future pattern evaluation material on such a framework.

Within the interactive patterns workshop, which was conducted with design students, it turned out that our criteria mostly matched with the requirements on patterns from a designer's point of view. During the discussions, the designers considered findability, understandability and helpfulness as very important for a pattern of high quality. However, the helpfulness criterion was seen somewhat twofold: on the one hand, designers indicated that support in solving common problems by the use of patterns would be beneficial for them. On the other hand, they were critical of too much support, as this could be negative for the designer's creativity, i.e. lowering the creative thinking process.

Comparing the results of the two validation sessions, a fundamental difference can be stated. The first validation session, conducted as an interactive pattern workshop, brought many practically applicable results for improving

the selected patterns but less quantitative judgements about the quality of the patterns. In the second validation session, a pattern checklist was used for rating selected quality criteria. This brought quantitative results, but did hardly deliver suggestions for the improvement of lower ranked patterns. We therefore argue that the ideal way of validating and at the same time improving a pattern is to combine both approaches, i.e. to combine an interactive pattern workshop with a pattern checklist.

V. Conclusion and Future Work

The development of patterns is an iterative process. As the iteration should lead to an improvement in quality, structured support on how to improve a pattern's quality is needed. In this paper, we aim at giving such a structured support by introducing a so-called quality criteria framework.

To show the framework's practicability, it was applied in a case study for validating the quality of selected User Experience (UX) Patterns. Therefore, relevant criteria were selected from the quality criteria framework and validated by means of an interactive pattern workshop and a pattern checklist. The results of the workshop and the checklist provided important insights on the quality of the patterns and thus helped to improve the patterns' quality. By applying these two methods successively (on the same patterns), an iterative improvement of the patterns was achieved.

The case study indicates that our framework supports the validation of a pattern's quality. On the one hand, the framework provides a structured selection of quality aspects which should be validated and improved, and on the other hand, using the framework ensures that no criterion is ignored unintendedly. Furthermore, the methods applied in the validation sessions (interactive pattern workshop, pattern checklist) turned out to be valuable validation methods, yielding helpful implications for the improvement of a pattern's quality.

An advantage of our validation framework is its broad perspective, which should ensure a broad applicability. Current approaches (see chapter on related work) often focus on single aspects and criteria to judge patterns, but miss a broader view. With our framework we provide a comprehensive view (based on an extensive literature review) on what is important to ensure a high quality for different kinds of patterns. Thus, the risk to forget important issues is lowered.

Moreover, our framework supports the operationalization of the quality criteria by introducing sub-criteria, which can be selected depending on the pattern status and relevance within an iterative validation process. This also influences the selection of the method. Based on the experiences made during our case study we assume that a pattern in its early state benefits more from open discussion and qualitative data (e.g., interactive pattern workshop), whereas a more mature pattern can be better improved based on quantitative feedback (e.g., pattern checklist).

As future work we plan to investigate the components of our quality criteria framework in more detail. As already pointed out, the present categorization is not always selective as some sub-criteria could also be subsumed under other criteria. Therefore, we aim to find out more about the relationship of the criteria and sub-criteria, perhaps resulting in a more hierarchical framework. We further plan to validate our UX patterns on the basis of our quality criteria framework, especially dealing with the criteria not applied so far. For example, criterion 4, "Empirical Validation", should be investigated in real design and development processes. Overall, the second important validation approach, i.e. the practical usage of patterns, should be deployed for validating our patterns.

Additionally, we plan to extend our UX pattern website, not only containing the UX patterns collection but also the quality criteria framework as well as methods and tips for pattern validation.

Furthermore, we currently extend our UX pattern collection to other contexts beyond the presented case study on audiovisual systems. Within the Christian Doppler Laboratory for "Contextual Interfaces" we work on special context areas, namely on the context car and factory. By applying our quality criteria framework to these different contexts, we want to further assure the general applicability of our validation framework.

Acknowledgment

The financial support by the Federal Ministry of Economy, Family and Youth and the National Foundation for Research, Technology and Development is gratefully acknowledged (Christian Doppler Laboratory for "Contextual Interfaces"). This work was partly also funded by the CITIZEN MEDIA research project (funded by FP6-2005-IST-41).

References

[1] D. Wurhofer, M. Obrist, E. Beck, and M. Tscheligi, "Introducing a comprehensive quality criteria framework for validating patterns," in *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. IEEE Computer Society, 2009, pp. 242–247.

[2] M. Obrist, M. Tscheligi, B. de Ruyter, and A. Schmidt, "Contextual user experience: How to reflect it in interaction designs?" in *CHI '10: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2010.

[3] A. Cooper, R. Reimann, and D. Cronin, *About Face 3: The Essentials of Interaction Design*. Indianapolis: Wiley, 2007.

[4] J. Borchers, *A Pattern Approach to Interaction Design*. Chichester, England: John Wiley & Sons, 2001.

[5] D. Martin, M. Rouncefield, and I. Sommerville, "Applying patterns of cooperative interaction to work (re)design: e-government and planning," in *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2002, pp. 235–242.

[6] J. Tidwell, *Designing Interfaces : Patterns for Effective Interaction Design*. O'Reilly Media, Inc., 2005.

[7] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series)*. Oxford University Press, 1977.

[8] M. van Welie and G. van der Veer, "Pattern languages in interaction design," in *Proceedings of IFIP INTERACT03: Human-Computer Interaction*. IFIP Technical Committee No 13 on Human-Computer Interaction, 2003, p. 527.

[9] T. Erickson, "Lingua francas for design: sacred places and pattern languages," in *DIS '00: Proceedings of the 3rd conference on Designing interactive systems*. New York, NY, USA: ACM, 2000, pp. 357–368.

[10] C. Alexander, *The timeless way of building*. Oxford University Press, New York, 1979.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[12] M. Bernstein, "Patterns of hypertext," in *Hypertext*, 1998, pp. 21–29.

[13] M. Nanard, J. Nanard, and P. Kahn, "Pushing reuse in hypermedia design: golden rules, design patterns and constructive templates," in *HYPERTEXT '98: Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems*. New York, NY, USA: ACM, 1998, pp. 11–20.

[14] D. A. Norman and S. W. Draper, *User Centered System Design; New Perspectives on Human-Computer Interaction*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1986.

[15] E. Bayle, R. Bellamy, G. Casaday, T. Erickson, S. Fincher, B. Grinter, B. Gross, D. Lehder, H. Marmolin, B. Moore, C. Potts, G. Skousen, and J. Thomas, "Putting it all together: towards a pattern language for interaction design: A chi 97 workshop," *SIGCHI Bull.*, vol. 30, no. 1, pp. 17–23, 1998.

[16] T. Erickson, "The Interaction Design Patterns Page," Website, 2005, available online at http://www.visi.com/~snowfall/InteractionPatterns.html; retrieved at July 30th 2009.

[17] M. Van Welie, "A Pattern Library for Interaction Design," Website, 2005, available online at http://www.welie.com; retrieved at July 30th 2009.

[18] C. Crumlish and E. Malone, *Designing Social Interfaces*. O'Reilly, 2009.

[19] C. Kruschitz and M. Hitz, "The anatomy of hci design patterns," in *Patterns2009: Proceedings of the First International Conferences on Pervasive Patterns and Applications*. IEEE, 2009.

[20] E. S. Chung, J. I. Hong, J. Lin, M. K. Prabaker, J. A. Landay, and A. L. Liu, "Development and evaluation of emerging design patterns for ubiquitous computing," in *DIS '04: Proceedings of the 5th conference on Designing interactive systems*. New York, NY, USA: ACM, 2004, pp. 233–242.

[21] K. McGee, "Patterns and computer game design innovation," in *IE '07: Proceedings of the 4th Australasian conference on Interactive entertainment*. Melbourne, Australia, Australia: RMIT University, 2007, pp. 1–8.

[22] P. Kotzé, K. Renaud, and J. v. Biljon, "Don't do this - pitfalls in using anti-patterns in teaching human-computer interaction principles," *Comput. Educ.*, vol. 50, no. 3, pp. 979–1008, 2008.

[23] A. Dearden and J. Finlay, "Pattern languages in hci: A critical review," *Human-Computer Interaction*, vol. 1, pp. 49–102, 2006.

[24] S. Niebuhr, K. Kohler, and C. Graf, "Engaging patterns: Challenges and means shown by an example," in *Engineering Interactive Systems*. Berlin/ Heidelberg: Springer, 2008, pp. 586–600.

[25] D. Khazanchi, J. Murphy, and S. Petter, "Guidelines for evaluating patterns in the is domain," in *MWAIS 2008 Proceedings*, http://aisel.aisnet.org/mwais2008/24, 2008, p. Paper 24.

[26] C. Graf, "A requirements engineering perspective to repositories for interaction patterns," Accepted position paper for EuroPLOP 2007 Focus Group, 2007.

[27] A. Dearden, J. Finlay, L. Allgar, and B. McManus, "Evaluating pattern languages in participatory design," in *Proceedings of the Conference on Human Factors in Computing Systems*. ACM New York, NY, USA, 2002, pp. 664–665.

[28] T. Schummer, J. Borchers, J. C. Thomas, and U. Zdun, "Human-computer-human interaction patterns: workshop on the human role in hci patterns," in *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2004, pp. 1721–1722.

[29] J. O. Borchers, "Chi meets plop: an interaction patterns workshop," *SIGCHI Bull.*, vol. 32, no. 1, pp. 9–12, 2000.

[30] J. Borchers, "Interaction design patterns: Twelve theses," Position paper for the workshop Pattern Languages for Interaction Design: Building Momentum, CHI 2000, 2000.

[31] N. L. O. Cowley and J. L. Wesson, "An experiment to measure the usefulness of patterns in the interaction design process," in *INTERACT*, 2005, pp. 1142–1145.

[32] K. Segerståhl and T. Jokela, "Usability of interaction patterns," in *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM, 2006, pp. 1301–1306.

[33] M. Obrist, D. Wurhofer, and M. Tscheligi, "User experience patterns: A useful way to support social media development," in *Proceedings of NordiCHI 2008 workshop*. TAPIR Akademisk Forlag, 2008, pp. 80–85.

Adaptable and Adaptive Visualizations in Concept-oriented Content Management Systems

Hans-Werner Sehring
Experience Design and Emerging Technologies
T-Systems Multimedia Solutions GmbH
Dresden, Germany
hans-werner.sehring@t-systems.com

Abstract—**One task of content management is the publication of content. The necessary means to render content into documents are usually developed alongside other aspects of content management systems, in particular the content's schema. There are content management applications, however, that require open and dynamic content modeling and management. These concept-oriented content management (CCM) systems have been studied carefully. As a consequence, content visualization in this kind of applications has to be adaptive and cannot be statically tailored to one given content structure alone. This paper gives a roundup of CCM, discusses means to abstractly define content visualizations, and presents an approach to adaptive visualization. The paper is an extended version of [1].**

Keywords-concept-oriented content management; adaptive user interfaces; personalization; content distribution

I. INTRODUCTION

In practice, there is no sharp definition of content management. There is agreement, though, that content management has to support the separation of layout, structure, and content [2]. To this end a typical *content management system* (*CMS*) allows to define structure through a *content schema* or *content model*, to manage content as data, and to render content into documents following a specified layout through *templates* (static view components plus code for the representation of content) during *playout*.

CMSs are applied in different scenarios, though. In particular there are cases where content is itself the primary entity of interest – when digital content is considered like in digital image collections, video portals, and Web 2.0 applications – and there are cases when content is used to represent real-world entities that cannot adequately be represented by structured data.

For the latter class of applications we have introduced *Concept-oriented Content Management* (*CCM*). In addition to the above-mentioned general requirements, *Concept-oriented CMSs* (*CCMSs*) have to support personalization of both data [3] and schemata [4] as a means to express subjective interpretations of content. Additional requirements follow from these properties: content models have to be open to changes and CCMSs have to follow model changes dynamically, while users need to be able to communicate with each other in the presence of personalized content (models) that may differ to a certain degree.

Earlier papers reported on the technical foundations of CCM that allow handling schema evolution and individualized communication in CCMSs. In this paper we discuss visualization matters for such systems, both for editing content according to personalized models as well as the rendering of content into documents that can be published independently of a content schema.

The remainder of this paper is organized as follows: in Section II we revisit CCM as a content management approach. Since the technical details of the CCM approach have been described thoroughly in other publications [4], the paper gives a summary of these topics. In Section III, however, we provide a more detailed look on content modeling with CCM. Sections IV and V cover the main topic of this paper, adaptive visualization of content. In Section IV adaptable visualizations for CCMSs are discussed in general, and specifically details on view models. Section V discusses the modeling of controllers to handle interaction. The paper concludes in Section VI with an outlook on future research.

II. CONCEPT-ORIENTED CONTENT MANAGEMENT

The CCM approach has been designed for content management applications that require handling content as personalized variants rather than in one standardized form. Two major requirements to CMSs have been identified for this kind of applications: content models have to be open to schema changes (*openness*), and CCMSs have to follow model changes dynamically (*dynamics*).

As a means to meet these requirements, three major contributions have been identified for the CCM approach: a language for open content modeling, a model compiler that translates content schema definitions into CCMSs that both implement a given schema and allow communication between subsystems with different variants of a schema, and a CCMS architecture that allows systems evolution through incremental compilation.

In this section we describe the definition of CCM models and the technology to implement CCMSs.

A. Foundations of Concept-oriented Content Management

Various projects have shown the need for a form of content management that is concerned about content that represents real world entities. This form of content

management usually is employed for entities that cannot accurately be described by structured data, e.g., by records in databases. One example for a class of such entities is that of pieces of art. A piece of art can be enjoyed as content, but it also represents the process of its creation, in particular the epoch in that it has been created, the artist and her or his way of living, as well as the message that shall be conveyed by the artwork (an opinion of the artist, or a statement of the employer as it is the case for politically motivated art, advertisements, etc.).

Content that represents entities is – in contrast to structured data – subject to individual interpretations. Content can be stored, shared, etc., but it will necessarily lead to subjective views on the represented entities.

Collaborative work with content that represents entities requires support to make those aspects of interpretations explicit that are intended by the author. In accordance with observations already made by others (in fact, they have been made as early as by Cassirer's works [5]), we propose to provide a conceptual model that accompanies the content.

We call the union of content and a conceptual model that both describe the same real-world entity an *asset*.

In order to be able to express subjective views, CCMSs support personalization, in particular personalization of asset instances, models, and presentations. As already mentioned, they do so by providing model openness and systems dynamics.

B. Asset Definition Language

The *asset definition language* (*ADL*) allows expressing entity models as laid out in the previous subsection. In this subsection we give a small glimpse of the ADL syntax, while we discuss modeling with assets in Section III.

Asset schemata are given as *models*. Models contain *classes* with *members* (*content handles* and *attributes*) and instance definitions. Furthermore, classes can be imported from other models, thus allowing model reuse (see Section III).

The following code shows an example of an asset model:

```
model MyModel
from SomeOtherModel import SomeClass
class MyClass
class MySubClass refines SomeClass
let myAsset :MyClass := …
```

Here, two classes and one instance are defined as part of the model called `MyModel`, where one class is defined as a refinement of an existing class imported from another model.

In the `let` statement for the definition of the named instance `myAsset` a type constraint can be seen after the colon. By using such type information a more general type than that of the actual instance can be given.

A type is given by the name of a class. If an asterisk follows the class name then the type refers to a set-valued type over the given base type.

Classes separate the two aspects of an asset – content and concept view – in respectively named compartments:

```
class MyClass refines SomeClass {
  content someContentHandle :HandleType
  concept … ; see below
}
```

Each content handle is given by a name and a type constraint (introduced by the colon). Please also note the semicolon introducing a one-line comment.

Since assets are similar to signs considered in Semiotics, we loosely base the concept part of assets on Peirce [6], in particular his distinction of three description categories.

The first category of the conceptual model consists of attributes that contain values that are inherent to instances:

```
class MyClass {
  concept characteristic c :T
          characteristic d :T2 := … }
```

Characteristic values are not first class citizens of an asset model. The usable types (in the example: `T` and `T2`) are borrowed from an underlying implementation language. Currently, we use Java for this purpose: any Java class from the standard or other class libraries can be used as a type, and Java expressions can be used in initializations (":=").

If an asset can be related to other assets, named and typed relationships can be defined as the second kind of attribute:

```
class MyClass {
  concept relationship r1 :C
          relationship r2 :D* }
```

Here, a relationship `r1` to an instance of asset class `C` and a many-to-many relationship `r2` to instances of type `D` are defined.

The third contribution of class definitions is that of regular definitions on the type level. These apply to all instances of the respective class (and, by means of inheritance, that of subclasses). Of course, classes itself as well as the type constraints are already regular contributions. Nevertheless, the need for application-specific constraints often arises:

```
class MyClass {
  concept constraint constraint1 c = x
          constraint constraint2 c < y
            onviolation … }
```

These definitions define the value of `c` to always be equal to `x` and less than `y` (where the comparison operators are defined in a type-specific way). Changes to the asset that would violate the first constraint are forbidden and lead to runtime errors. The second constraint contains a productive rule that establishes (or at least tries to establish) a situation that conforms to the constraint.

There are seven built-in operators to check for equality ("="), inequality ("#"), ordering ("<", "<=", ">", ">="), and similarity ("~"). These are implemented in a type-specific way. E.g., "<" tests for a subtype for classes, and for a subset for asset sets.

Named asset instances can be referred to by their name. Members of instances can be accessed by the projection operator (".", e.g., `myAsset.x`. Asset sets are given by a comma-separated asset enumeration in brackets.

The asset creation and manipulation sublanguage controls the lifecycle of asset instances. It allows creating and modifying asset instances through operations like:

```
create MyClass { c := x }
create MyClass someMyClassInstance
modify someAsset { c := x }
modify someAsset someOtherAsset
delete someAsset
```

The statements are available in intentional form (giving member values) and in extensional form (giving a prototype). A set of prototypes can be given in the extensional forms; then statements are applied element-wise.

The asset query sublanguage allows finding asset instances using statements like

```
lookfor MyClass { c # y }
```

Operations can be combined by means of concatenation. The following sample statement updates all instances of `MyClass` with a value `x` of attribute `c` so that `c` becomes `y`:

```
modify lookfor MyClass { c=x } { c:=y }
```

Concatenation follows the implicit rules that (1) sets of sets of instances are flattened to sets of instances, that (2) there is no distinction between singleton sets and single instances, and that (3) projection can be applied to sets element-wise. For example, the statement

```
{lookfor MyClass { c = x },
 lookfor MyClass { c = y }}.c
```

retrieves the union of all instances of `MyClass` with a `c` value of `x` or `y` and projects it to the value(s) of the attribute `c` (this can result to `{}`, `x`, `y`, or `{x,y}`, depending on the existing asset instances).

C. On Open Content Modeling

Asset models are units of model reuse. Through the `import` statement classes can be imported and used for two reasons: for domain interrelation and for personalization.

The first way of reuse, domain interrelation, allows integrating definitions in order to use a domain as a related domain or subdomain. If studying art history, for example, one will want to reuse some work of historians.

One specific property of the ADL is its ability to redefine content classes in a specific context. By this means the ADL can be used for personalization and for the management of content revisions and content variants. Imports of classes for this reason are the second use of model reuse.

The redefinition of classes can include the addition of attributes, the removal of attributes, and changes to the inheritance hierarchy. For example, based on

```
model SomeModel
class SomeClass {
  concept characteristic c :T1
          characteristic d :T2 }
```

some user may define

```
model MyModel
from SomeModel import SomeClass
class MyBaseClass
class SomeClass refines MyBaseClass {
  concept
    characteristic c :T3 ; changed type
    characteristic d unused ; omitted
    characteristic e :T4 } ; new attribute
```

Note that class redefinition has neither subtype semantics nor does is create revisions of types. Instead, each model is checked for consistency separately. Therefore, regardless of class definitions being based on imports, changes like a modified class hierarchy and the omission of attributes (keyword `unused`) are sound when looking at one model alone. Relationships between models (for model personalization etc.) are handled by explicit inter-model relationships established by, e.g., initializations with default or computed values. For example, a class `C` like

```
model BaseModel
class C { concept characteristic i :int }
```

can be changed using the `origin` reference to the original class definition to become:

```
model DerivedModel
from BaseModel import C
class C { concept characteristic i :String
          := Integer.toString(origin.i) }
```

This way one can change the type of an attribute and have the new value computed, e.g., when passing an instance from a `BaseModel` context to one using `DerivedModel`.

This way, one can even change attribute kinds, e.g., lift a characteristic value

```
class Painting {
  concept characteristic painter :String }
```

to a relationship

```
class Painting {
  concept relationship painter :Painter
   :=lookfor Painter{name=origin.painter}}
```

D. A Model Compiler for Concept-oriented CMSs

Due to the openness and dynamics requirements CCMSs call for specific implementations. Both well-known extreme software development approaches, individual development and generic software, fail to meet these requirements: individual software is not dynamic since it needs interference of programmers when model changes occur. Generic software does not meet the openness constraint since it prescribes certain model constructs that cannot be overcome and require the user to translate concepts as expressed in the generic language. Therefore, automatic software generation in conjunction with a fine-grained architecture is necessary to allow dynamics of information systems.

There are different approaches to the problem of generating whole software systems which are composed of various parts that are produced by independent generators: (1) the generated software modules have to be adapted in order to be composed [7], (2) generic software modules are wrapped in a domain-specific way [8], (3) glue code to combine modules needs to be generated [9], or (4) the generators need to cooperate in order to create a consistent set of modules. For the fully automatic generation approach required for CCM we favor the latter approach for content management systems.

Writing coordinated generators is a complex task, mainly because setting up an infrastructure for them [10] is difficult. Therefore, our model compiler for content management systems is designed as a framework. In conjunction with a facility for code generation it constitutes a domain-independent meta-programming infrastructure [11].

An instance of the compiler framework is defined by providing a *parser*, one or more *dictionaries*, several *generators*, and a *configuration* of the framework [12].

A typical compiler is divided into frontend and backend [13] in order to decouple source language recognition from target language generation. To this end, a compiler frontend creates an intermediate representation of the input definitions. Such an intermediate representation

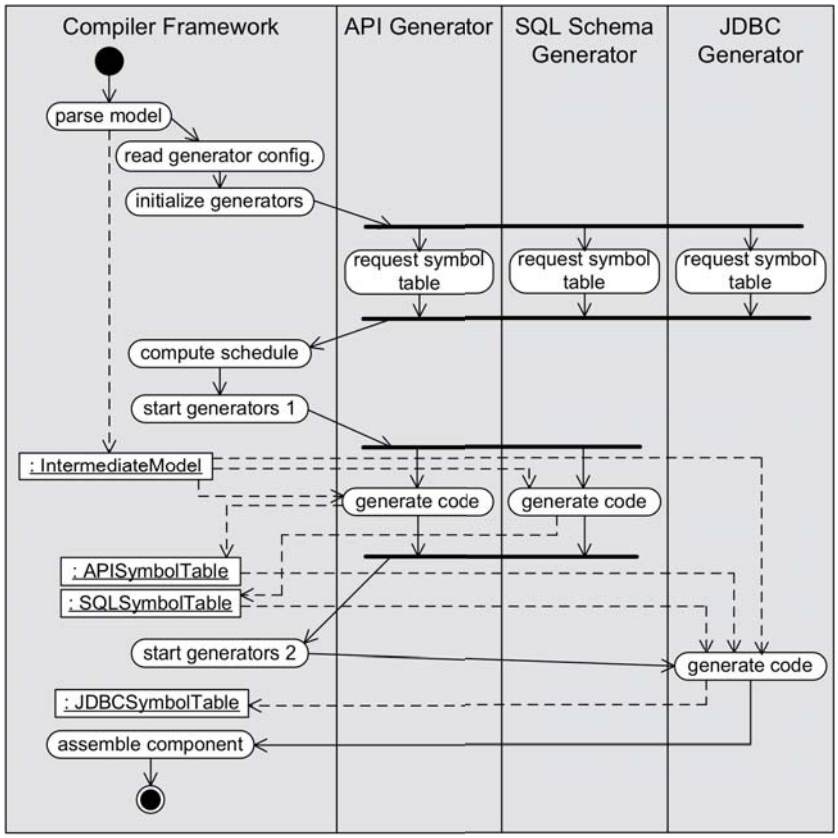


Figure 1. Activity diagram of a sample CCM compiler framework application.

forms the input of a compiler's backend that generates code in the target language. This allows compiler setups for multiple targets as well as – at least in theory – to process different source languages.

The model compiler for our conceptual language is built in an object-oriented fashion. The classical division into frontend and backend has been translated into a framework architecture that allows configuring compilers for the generation of dynamic content management systems. This framework addresses the need to generate multiple targets in conjunction.

A set of parsers is readily available for model compiler instances. The one most commonly used reads files containing asset language expression as defined in Section B. Other options are parsers for different syntactical forms, e.g., in XML, or parsers that adapt an internal model representation from modeling tools. For the purpose of user interface generation, an input language that is related to established presentation technology could be used (see Section V.B).

Alike a programming language compiler that creates an intermediate code representation the frontend in the compiler framework creates *intermediate model* representations in which asset class definitions are available as an object graph.

CCM model compilers have access to one or more dictionaries in which model definitions are stored. This way a compiler gets access to the models named in import statements. It furthermore registers information on the generated code in an own dictionary. This includes the names of implementation classes (e.g., fully qualified Java class names) that have been created for asset classes. Through the dictionaries compilers can create model interrelationships by accessing the information that has been stored by earlier compiler runs.

Code generators constitute the most important extension point of the model compiler framework. Each generator produces one module of a CCMS (see subsequent subsection) in one particular technology. The framework schedules the generators with respect to their dependencies.

There is a direct correspondence between generators and the modules of content management systems. For each implementation of one of the module kinds introduced below there is at least one generator. Often more than one generator contributes to the creation of a module. For example, client modules for database access are typically created by a pair of generators; one of them creates the database schema, the other one creates code to access the database as well as to store and retrieve asset instances.

Sets of generators are given in model compiler *configurations*. Generator instances out of the set of known generator implementations are chosen by means of selecting a configuration. In the context of user interface generation, for example, there are typically different configurations for different presentation technologies used for a CCMS.

Traditional compilers use *symbol tables* to store information about the language constructs recognized. Our model compiler for content management systems builds on the concept of symbol tables, but extends it significantly: these tables are not only used in the frontend of a compiler, but they are the means by which generators communicate during the generation process.

Symbol tables contain detailed information about the artifacts that were created by the respective generator. The aim of symbol tables is to make access to the artifact descriptions explicit for generators that rely on artifacts created by others (and most generators do). Without symbol tables, generators further down the chain would have to make assumptions about naming and would have to recover the corresponding pieces from the whole of the generated artifacts.

Each generator fills its symbol table during its execution and passes the symbol table back to the compiler framework afterwards. The framework in turn gives available symbol tables to further generators making them the essential means of generator communication.

A complete system is normally built from artifacts in several languages. Different meta-programming facilities are available to the generators that share a common intermediate model to create their output.

Figure 1. illustrates the cooperation of generators within the compiler framework. The main task of the frontend is to parse a CCM model definition and to create an intermediate model from it.

As part of the initialization of the generators in the backend, the framework determines the symbol tables each generators needs as input. Based on this information a schedule for generator execution is computed.

The compiler backend passes the CCM model (in the form of an intermediate model) and the required symbol table(s) to each generator. The example shows a setup with three generators. The first one, the API generator, is found in every setup. It creates the uniform module interface with respect to the CCM model. The current implementation creates Java interfaces.

The other two generators together create a client module (s.b.) for use with a relational database management system. One generator creates a relational schema out of the asset model, the other one a module implementation using JDBC to access the database according to the generated schema.

The JDBC generator will always be scheduled last since it requires information on both the schema (to create the proper "embedded" SQL statements) and the module API (in order to make the JDBC module implement it).

The final compilation step is the component assembly. A CCMS component is assembled from the generated modules and parameterizations of third party products when all contributing generators have finished their task. This includes two activities: actually building the modules and combining them in a component of a CCMS.

Modules are built from the generated artifacts. Each generated artifact needs a special final treatment: source code needs to be compiled, database schemata have to be deployed, etc.

E. An Architecture for Concept-oriented CMSs

A model-driven code generator – in contrast to programming language compilers – is in full charge of the architecture of the software it generates. This enables the CCM compiler to generate CCMSs in a form that allows incremental compilation. Consequently we have designed an architecture that allows CCMSs to evolve dynamically, thus meeting the dynamics requirement of our content management approach.

The creation of such an evolvable system can in some cases entail changes to its setup. The architecture of the system must therefore allow for flexible reconfiguration. A monolithic system is certainly not capable of such flexible change. Quite the contrary, we propose a modular system architecture that is built of many small modules.

Consequently, the most important concepts of the CCMS architecture are *components* and *modules* [4]. Conceptually, components are units of model reuse, while modules establish code reuse.

Components are logical units that implement one asset model. They are in turn implemented by modules that are each generated specifically for one functionality aspect – like persistence, distribution, transformation, etc. – in one component.

A component is implemented by a combination of modules, usually arranged in layers. Components as software artifacts themselves provide several services to their modules: resolution of identifiers, management of module lifecycles, and initialization of modules at system startup. Each module can use other modules and can also be used by several others. However, the setup of modules in a component always must be a directed acyclic graph.

All modules have a uniform interface and can therefore freely be composed in layers. The module interface reflects the capabilities of the asset language to create, modify, delete and query for asset instances. Each module can thus express its functionality in terms of calls to the module(s) on the underlying layer. This makes it possible to always combine modules in the way most appropriate to the task at hand.

Figure 2. Six kinds of modules of CCMSs.

Figure 2. illustrates the kinds of modules for the most frequently occurring tasks:

- Components are accessed via *server modules* using standard protocols.
- Asset instances (content, characteristic values, and relationships) are stored in third party systems, databases in most cases. *Client modules* perform the mapping of assets from asset models to schemata for such third party systems.

- A central building block of the architecture of generated content management systems is the mediator architecture [14]. Modules of two kinds implement it in our approach. The first are *mediation modules* that delegate requests to other modules based on the request (operation and assets involved).
- The other modules are *transformation modules*. By encapsulating mappings in such modules, rather than integrating this functionality into other modules, mappings can be added dynamically (compare [15]).
- *Hub modules* uniformly distribute calls to a larger number of underlying modules.
- By use of *distribution modules* components can reside at different physical locations and communicate by exchanging data, e.g., XML documents generated from the asset definitions (comparable to the approach of [16]).

These module kinds have been identified with respect to the requirements of content management systems. They provide basic services by the principle of *Separation of Concerns*.

The functionality of a content management system is implemented by a *component configuration* that composes selected modules. Important building blocks are typical module constellations, the perhaps most important one being an implementation of the mediator pattern [14] consisting of a *transformation* and a *mediation module*. Figure 3. illustrates it. Mediator pattern applications are discussed below.

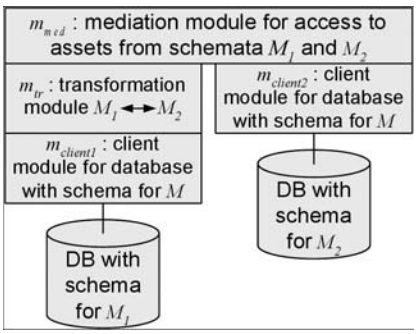


Figure 3. Architectural building block for evolution in CCMSs

For each kind of module used and for each supported implementation technology there needs to be one generator to equip the compiler framework with.

According to the two ways of combining asset models – model interrelation and personalization – openness and dynamics in CCMSs happen along two dimensions: (1) the *organization* and (2) the *application structure* [17]. Along the organization structure users can define their own views (by personalizing content and schema). Along the application structure, entity descriptions are shared and reused across domains.

In our approach the architecture of the generated systems allows changes along the organization structure by its ability to enable dynamic system evolution and personalization through open redefinition of assets and dynamic invocation of the model compiler [4].

Schema evolution leads to a mediator combination of client, transformation, and mediation modules as indicated in Figure 3. Evolution or personalization requires a mediation module that implements the desired personalization functionality (m_{med} in the figure). Typically this includes the delegation of requests in such way that new instances are created in the component for the new schema (M_2), modifications lead to the creation of a modified copy in that component while removing it from the component holding the outdated model (M_1), and search queries and deletion requests are posed on both components. Such a mediation module can be generated based on the input information, namely a base model and the changes applied to it in a derived model.

Personalization is quite similar, with the difference that modification of an asset leads to the creation of a copy that contains a reference to the personalized asset (instead of deleting the original), and deletion leads to the creation of a *null* asset hiding the original.

The association of models along the application structure is realized by component configurations. Figure 4. shows a configuration that combines two domains – regent and artist descriptions – into the new domain of political iconography. The component is accessed via mediation module m_{med1}. It distributes requests according to the type of the assets on which operations are invoked. If assets from one of the base domains *Regents* or *Artists* are affected, requests are delegated to the mediation module m_{med2}. This mediation module similarly delegates requests further to one of the components holding theses models. These components are accessed via distribution modules $m_{distrib1}$ and $m_{distrib2}$. In the example of Figure 4. the components consist of client modules $m_{client1}$ and $m_{client2}$ and the respective base systems only. Requests to the derived model *Political_Iconography* are forwarded by m_{med1} to the client module m_{client} that manages the users' assets from the political iconography.

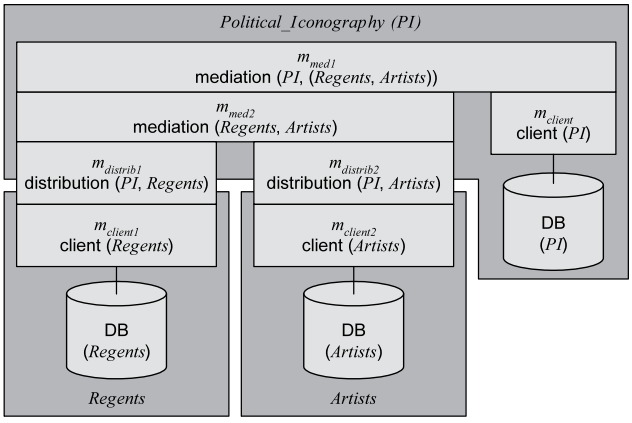


Figure 4. Sample CCMS components for domain interrelation.

As can be seen in Figure 4. the components for *Regents* and *Artists* are integrated into the overall CCMS for Political

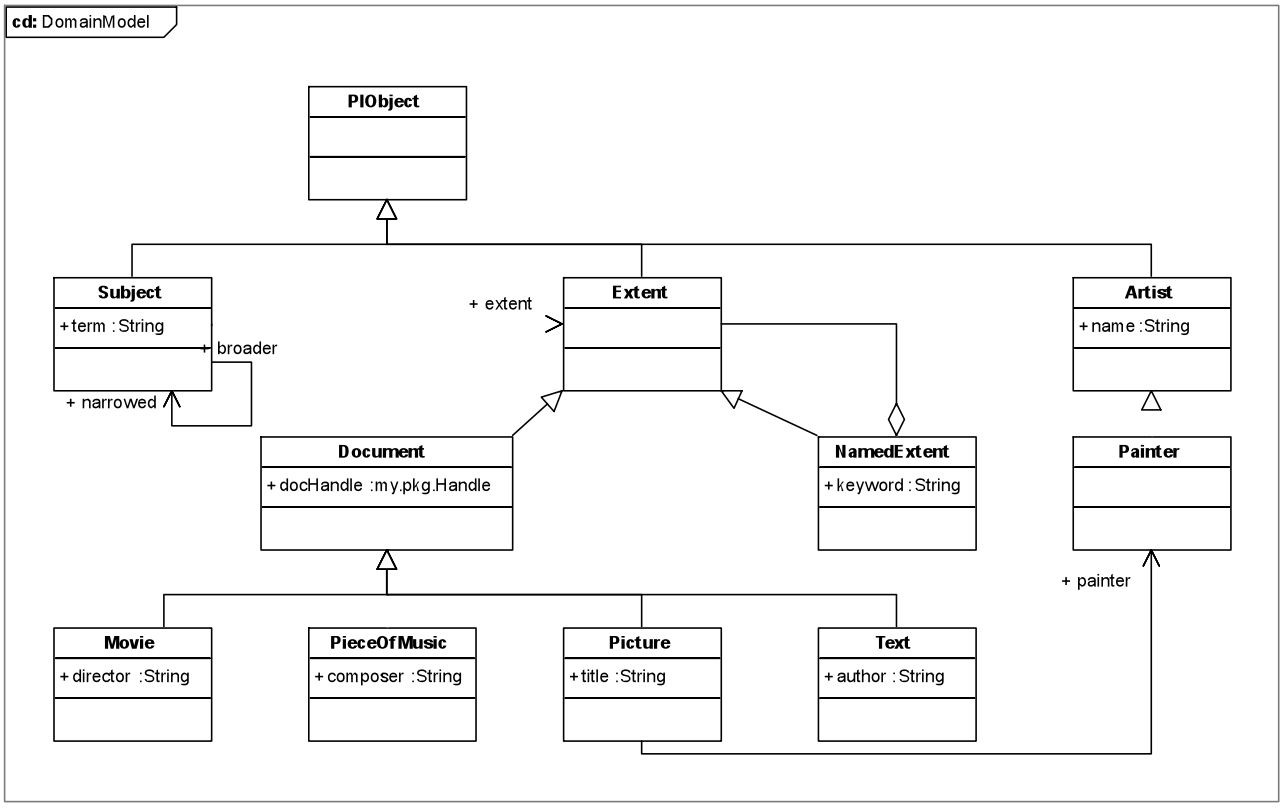


Figure 5. Sketch of a CCM model for the domain of political iconography.

Iconography without modification. This way the components remain unaffected, thus preserving their autonomy, i.e., to be maintained by experts from the respective domains.

III. CONCEPT-ORIENTED CONTENT MODELING

As a running example for the discourse on visualization led in the next section we introduce a tiny asset model in this section. It is a rather condensed extract from a model used in one actual project.

Figure 5. shows an overview in the form of a UML class diagram with attributes for asset characteristics and associations for asset relationships.

A. A Sample Structural Content Definition

The following code shows two simple sample classes:

```
class Document refines Extent {
  content docHandle :my.pkg.Handle }
class Picture refines Document {
  concept characteristic title :String
          relationship painter :Painter }
```

The content handle `docHandle` refers to document data, e.g., a digitized picture for a `Picture` instance. Let the Java class `Handle` be some class to handle references to such data.

The asset class `Picture` describes picture entities like paintings. In inherits the document handle, and defines a conceptual model consisting of a picture's title and a reference to a `Painter` asset.

B. Sample Content Classification by Relationships

Apart from the (structural) definition of the document descriptions it is necessary to define a hierarchy of (semantic) classifiers, here modeled as instances of class `Subject`. The base class `Extent` of `Document` defines the extent of subjects.

These are provided in the form of subject terms and corresponding relationships:

```
class Subject {
 content term :String
 concept
   relationship narrowed :Subject*
   relationship broader  :Subject
     = lookfor Subject {narrowed>={self}}
   relationship extent   :Extent* }
```

The content `term` is the subject term itself, and Java's standard String class is used for instances. The relationship `narrowed` points to more specific subject terms, and `extent` to the documents classified under the term at hand. The reverse of `narrowed`, `broader`, is modeled by a constraint that returns the broader terms based on the (persistent) relationship `narrowed`.

C. A Sample Content Evaluation Rule

The intended form of classification with the sample model given so far is the following: each document is classified under the most specific subject terms that apply. If

one wants to use the typical subsumption – documents also showing up under more general terms than those assigned directly – with the definitions made so far this has to be handled by, e.g., the visualization layer.

To make such evaluation rules part of the asset model, additional classes can be introduced for this purpose. For our example we would like to define a special `Subject` with a "deep" extent that takes extents from narrower terms into consideration:

```
class SubjectRec refines Subject {
 concept
  relationship extent :Extent* := {
   origin.extent,
   (create SubjectRec narrowed).extent }}
```

To further stress the importance of a rule-based level, please note that the transitive extent can be expressed by means of relationships (with the help of a productive rule to keep the relationships up-to-date):

```
class SubjectRec2 refines Subject {
  concept constraint deepExtent
            extent >= narrowed.extent
            onviolation modify self {
              extent += narrowed.extent }}
```

This way the recursive extent is materialized in each subject. Of course, the usual problems arise from this redundancy, e.g., updates of extents on picture removal.

IV. Open Dynamic Asset Representations

In this section we shed a light on the second typical task of CMSs, the rendering of representations of content.

The openness and dynamics properties of CCMSs require user interfaces (UIs) to follow model changes. This can only partially be achieved since suitable presentations require manual design [18]; there is no means to automatically produce a visualization that is guaranteed to meet the users' demand for adequateness and ergonomics.

Nevertheless, if visualizations are handled like content, then openness like that of content models can be achieved for them: visualizations can be defined in conjunction with domain models, e.g. group-wise, they can be passed between users that share the same domain models, and then they can be personalized group-wise or individually.

With this kind of user interface modeling presentations are not automatically generated from domain models, but users can define presentations on their own, using a language they are using anyhow. By the correspondence between domain and visualization models the CCM personalization capabilities are beneficial for user interface modeling. Not every user has to define a complete presentation. Usually, domain experts build their models reusing those of other domain experts (by means of personalization) or they use models of neighboring domains (by means of cooperation). Together with the reuse of domain models also accompanying presentation models can be reused.

To this end, we need to avoid "programming" of templates as found in typical CMSs. Instead, declarative definitions of visualization constructs are needed similar to the idea of *Model-Based User Interface Development Environments* [19]. Our aim is to express visualization using the ADL since domain experts already use it (compare [20]), and it allows direct links to domain models. It should be noted that the difference between content and a rendered document is in the eye of the beholder: view classes can be seen as normal classes from the domain of "views".

For the asset-based visualization descriptions we suggest models that follow the Model-View-Controller pattern. In this section the view part of the user interface models is presented. The subsequent section discusses the interrelationship between views and models, as well as the definition of controllers.

Three interrelated models for (1) abstract definitions of presentation components, (2) presentation technologies, and (3) component implementations are provided to CCMS users who can define asset visualizations with the help of these basic contributions for visualization specification.

The models apply to both pure presentations, like web pages, and interactive applications, e.g., for content editing.

Figure 6. gives an overview of the usage of the models presented in the remainder of this paper. The packages *Components* and *Technologies* represent two of the models discussed in this section (elsewhere called *platform model* [21]). The model of component implementations is omitted from the figure since it is not of interest to the domain expert using the models; it is exclusively used by the compiler. The package *Layout* sketches an application of the models (elsewhere called *presentation model* [21]). The relationships to the *DomainModel* are explained in the subsequent section.

A. Presentation Component Model

A very basic contribution for declarative UI descriptions is the *presentation component model*. This model enumerates abstract descriptions of visual components that are usually available in the supported visualization technologies. The following small model excerpt gives an impression of the class definitions, showing a base class of containers for other UI components and a text label class (to display some text):

```
model UIComponents
class UIComponent
class Container refines UIComponent {
  content children :UIComponent* }
class TextLabel refines UIComponent {
  content text :String }
```

Such an abstract component library is used to specify presentations for assets in a platform-independent way:

```
model MyPresentationModel
from UIComponents import ImageLabel,
                Panel, TextField, TextLabel
let picturePanel :Panel := create Panel {
  children := {
    create ImageLabel,
    create Panel {
      children := { create TextLabel,
                    create TextField }
    } } }
```

In this example a user defines a `Panel` consisting of an image, a text label, and a text field. It might be used to display pictures by showing the content (the picture data itself) and its title, where the text label is displaying "Title:" as a label to the text field, and the text field is holding the actual picture's title.

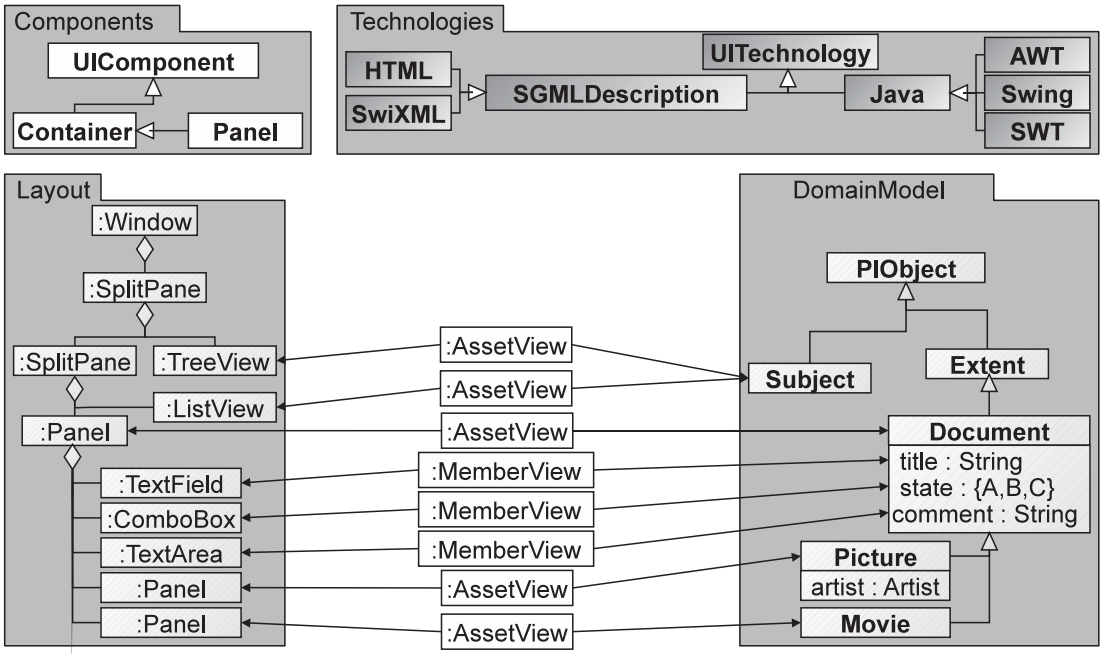


Figure 6. CCM models for user interface component implementations.

B. *Presentation Technologies Model*

The second basic contribution, the *technologies model*, is rather simple: it just enumerates the supported technologies by defining an asset class for each of them.

A snippet from a technologies model looks like this:

```
model UITechnologies
class UITechnology
class SGMLDescription refines UITechnology
class HTML  refines SGMLDescription
class Java  refines UITechnology
class AWT   refines Java
class Swing refines Java
class SWT   refines Java
```

The sole purpose of these classes is to be referenced in the component implementations model and being passed as a parameter to the presentation generator (see Section D).

C. *Presentation Component Implementations Model*

The third basic contribution is a model that contains implementations of components in certain technologies. Again a quite small excerpt shall present the basic idea:

```
model UIImplementations
from AssetMetaModel import AssetClass
from UIComponents import Panel,UIComponent
from UITechnologies
  import HTML, Swing, UITechnology
class UIImplementation {
  content prototype :java.lang.Object
  concept relationship component
             :AssetClass < UIComponent
          relationship technology
             :AssetClass < UITechnology }
create UIImplementation {
  prototype := my.HtmlUtils.element("div")
  component := Panel
  technology:= HTML }
create UIImplementation {
  prototype  := new javax.swing.JPanel()
  component  := Panel
  technology := Swing }
```

The model contains one class `UIImplementation` whose instances refer to prototypical implementations (in the current implementation given as Java objects) of abstract component definitions. The set of `UIImplementation` instances defines the pool of implementation artifacts that a UI generator (see Section D) can benefit from.

The link between abstract components and technologies is made by referencing the respective asset classes. The class `AssetClass` is imported from the ADL's metamodel (that is also available in ADL itself) for the required type constraints that furthermore restrict the referable classes to `UIComponent` and `UITechnology`, respectively.

The example sketches implementations of the abstract component `Panel` in HTML (here assuming a helper class to create HTML elements) and in Java Swing.

The definitions in `UIImplementations` are in fact written using more compact statements, but the necessary linguistic means have not been introduced in this paper.

D. *Presentation Generation Using Abstract Models*

User interface code is generated from the models presented to far, with one addition: links are established between presentation component instances and domain model entities in order to be able to create the demanded adaptive code. The links are based on the `AssetView` and `MemberView` assets as indicated in Figure 6. They will be discussed in Section V.A.

The actual rendering of assets is based on a rather simple algorithm: for each asset class `c` of the domain model to visualize and a technology (from the technologies model) `t`,

a UI generator first looks for the UI component(s) to use in the model relating content assets to UI component assets:

```
let v := (lookfor AssetView { type >= c }
         ).view
```

Then implementation prototypes for the UI components can be found in the component implementations model:

```
let p := (lookfor UIImplementation {
           component <= v.type
           technology = t }).prototype
```

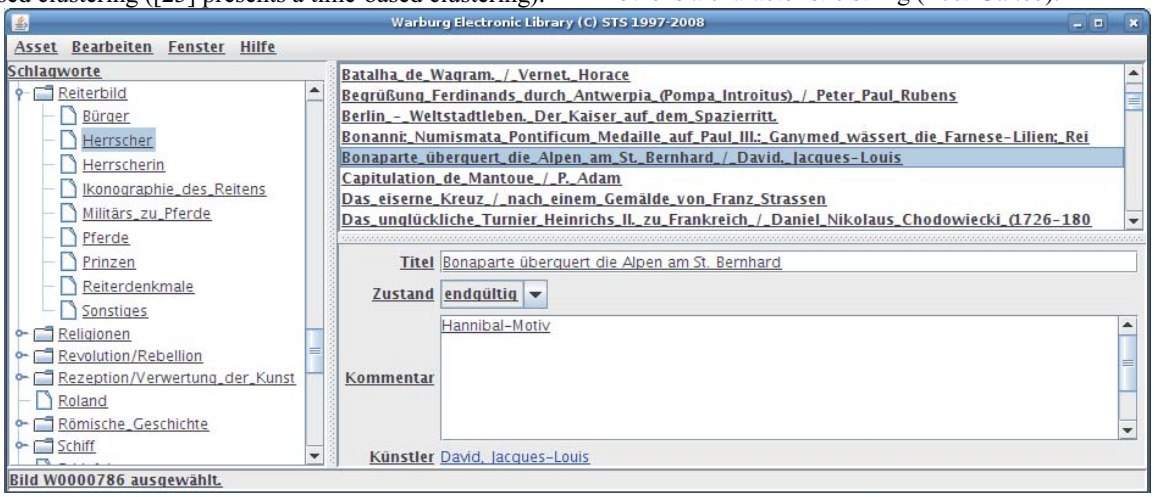
The prototypes are used to create fresh instances that are assembled to a UI as prescribed in the presentation component model. Assembly of the implementations usually has to be performed in a technology-specific way, so that there are specific generators for the supported technologies. These can be easily included in the asset compiler framework (see Section II).

The generated code does not create a static presentation. Instead, the user interface adapts to the asset bound to the presentation (see Section V.A). Such code forms an "adaptation engine" as proposed in [22] and establishes a type-based clustering ([23] presents a time-based clustering).

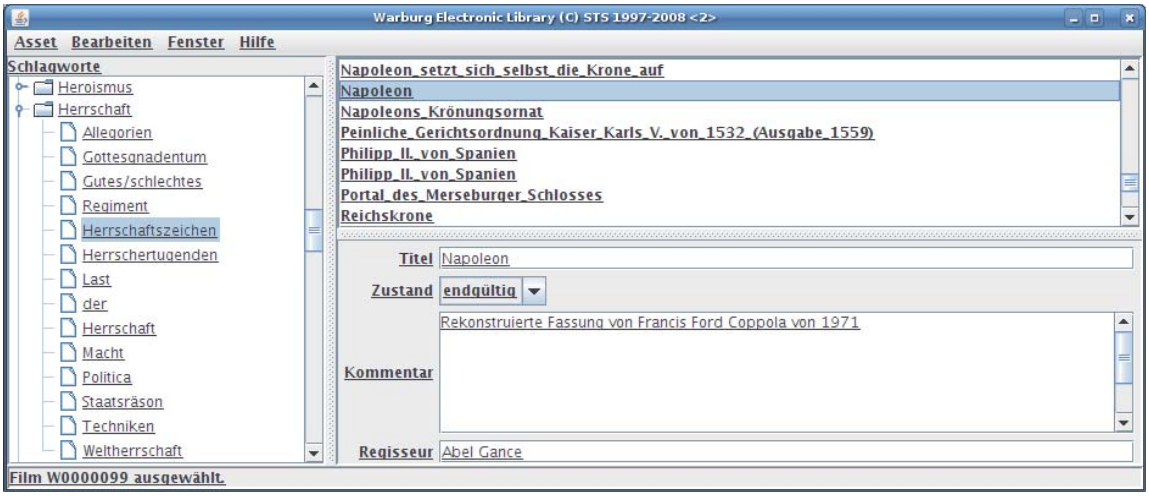
To this end, the code adjusts the presentation to the bound asset by selecting only those components that are defined for an asset type that matches the current asset's class. The selected components are added to the presentation, and child components that do not apply anymore (because they were added for a previously bound asset) are removed (equivalent to "generation at execution-time" in [24]).

How this adaptation is performed depends on the visualization technology. In Java (AWT, Swing, or SWT) applications, container components can dynamically be altered. Web pages need to be reloaded, or there could be JavaScript code to perform changes dynamically using AJAX (not yet implemented).

Figure 7. shows screenshots of a running Swing application. The screenshots show a GUI with a bound picture asset (a) and a movie asset (b). The panel in the lower right adapts to the bound asset. As can be seen, the artist (*Künstler*) of a picture is given as a relationship to a painter (*Jacques-Louis David*), while the director (*Regisseur*) of a movie is a characteristic string (*Abel Gance*).



(a) A CCM GUI showing a picture asset.



(b) A CCM GUI showing a movie asset.

Figure 7. Screenshots of a generated CCM visualization.

```
model ViewWithLinksToModel
from DomainModel import Picture, Subject
from UIComponents import horizontal, ImageLabel, Movie, Orientation, Panel,
                         TextField, TextLabel, UIComponent, vertical
class LabelAndField refines Panel {
  concept relationship label :UIComponent
          relationship field :UIComponent
          relationship children :UIComponent* = { label, field }
          relationship orientation :Orientation := horizontal }
class DocumentPanel refines Panel {
  concept relationship iconPanel :LabelAndField
          relationship children :UIComponent* := { iconPanel }
          relationship orientation :Orientation := vertical }
class MoviePanel refines DocumentPanel {
  concept relationship directorPanel :LabelAndField
          relationship children :UIComponent* := {super.children, directorPanel}}
class PicturePanel refines Panel {
  concept relationship titlePanel :LabelAndField
          relationship painterPanel :LabelAndField
          relationship children :UIComponent*
                                := { super.children,titlePanel,painterPanel } }
let classifierTree := create TreeView {
  nodeRenderer := create TextLabel }
let extentList := create ListView {
  itemRenderer := create TextLabel }
let moviePanel := create MoviePanel {
  iconPanel     := create LabelAndField {
                     label := create TextLabel { text := "Icon:" }
                     field := create ImageLabel
                   }
  directorPanel := create LabelAndField {
                     label := create TextLabel { text := "Director:" }
                     field := create TextField
                   } }
let picturePanel := create PicturePanel {
  iconPanel    := create LabelAndField {
                     label := create TextLabel { text := "Icon:" }
                     field := create ImageLabel
                   }
  titlePanel   := create LabelAndField {
                     label := create TextLabel { text := "Title:" }
                     field := create TextField
                   }
  painterPanel := create PicturePanel {
                     label := create TextLabel { text := "Painter:" }
                     field := create TextLabel
                   } }
```

Figure 8. Sample layout models for a user interface like the one shown in Figure 7.

V. Model Bindings and Controller Models

In the previous section the static layout of visualizations of assets of specific types has been presented, and it was already specified that the asset presentations should depend on the type of assets bound to the view. This section is concerned about the behavioral aspects of user interfaces. Following the model-view-controller pattern views are related to the two other interface components, models and controllers.

Bindings from the view layout to the domain model are covered by Section A. Section B presents and alternative form of defining layouts and model bindings.

Controllers typically serve one of two purposes: updating the view, e.g., by navigating between assets, and updating the model, e.g., by creating, modifying, or deleting one or

```
model ViewWithLinksToModel
from DomainModel import Movie, Picture, Subject
from AssetUI import AssetView, MemberView
create AssetView  {
  type := Subject           view := classifierTree }
create MemberView {
  member := Subject.term    view :=classifierTree.nodeRenderer }
create AssetView  {
  type := Extent*           view := extentList }
create MemberView {
  member := Document.name   view := extentList.itemRenderer }
create AssetView  {
  type := Picture           view := picturePanel }
create MemberView {
  member:=Picture.docHandle view := picturePanel.iconPanel.field }
create MemberView {
  member := Picture.title   view := picturePanel.titlePanel.field }
create MemberView {
  member := Picture.painter view := picturePanel.painterPanel.field }
create AssetView  {
  type := Movie             view := moviePanel }
create MemberView {
  member := Movie.docHandle view := moviePanel.iconPanel.field }
create MemberView {
  member := Movie.title     view := moviePanel.directorPanel.field }
```

Figure 9. CCM model interrelating views and model classes.

more assets. CCM models to define controllers for view updates are covered by Section C and such to define controllers for model updates in Section D. All kinds of controllers work on regular CCM assets.

A. Relating Content to Presentation Components

Users define the presentations they need on the basis of the abstract UI components model. They have to provide two kinds of definitions: an implementation-independent layout description as sketched in Section IV.A and links from content to the UI components that shall display that content.

In easy cases the link between content and a UI component can be made by referring to the content from the content compartment of a UI component. Additionally, the UI component is responsible for the access to the attributes of the asset to be visualized: the selection of members and the decision which relationships to follow (and to which depth). An example for a `Picture` instance `p` would be:

```
model ViewWithValues
from UIComponents import ImageLabel,
                Panel, TextLabel, vertical
create Panel {
  children := {
   create ImageLabel {image:=p.docHandle},
   create TextLabel  {text :=p.title} }
  orientation := vertical }
```

This way of linking content to UI components allows explicitly choosing the presentation for an asset instance, but it requires a complete definition of one instance per content type and desired visualization, without code reuse through classes. Compared to conventional implementations this is typical for simple manually programmed GUIs or such created with the help of interface design tools.

Some reuse can be achieved by defining UI classes for specific content (types) that are instantiated for a matching content instance. This is what typical template languages do.

A higher degree of reuse can be achieved by defining *rules* for the linkage of content to UIs.

The basis for such user-based visualization descriptions is a fourth basic model that defines class-based relationships:

```
model AssetUI
from AssetMetaModel
  import AssetClass, Member
from UIComponents import UIComponent
class AssetView {
  concept relationship type :AssetClass
          relationship view :UIComponent }
class MemberView {
  concept relationship member :Member
          relationship view :UIComponent }
```

The asset class `Member` is defined in the ADL's meta model like the metaclass `AssetClass` is.

Such a basic model can be used for definitions like the views shown in Figure 8. and the relationships shown in Figure 9. according to the graphical sketch in Figure 6.

The example shows a small excerpt of a model that defines a GUI like that from Figure 7. It consists of a tree showing the `Subject` hierarchy, the `Extent` list of one `Subject`, and a panel with the selected `Extent`.

Standard components are used for the tree and for the list in the example. These are configured with one component to render tree nodes and list items, respectively.

For the Extent assets one Panel per type is defined (the example of Figure 8. shows just excerpts of the panels for Movie and Picture instances). To be able to correctly set the horizontal orientation of, e.g., labels and text fields, and the vertical orientations of the components for the attributes, a helper class LabelAndField is defined. It allows to define orientations at the class level.

As sketched in Section IV.D, view components have to be related to domain model elements. Using the AssetUI model sketched above, the instantiations of AssetView and MemberView prescribe the rendering of all assets of a user-defined type. In the example of Figure 9. Picture (and subtypes) instances are defined to be rendered by a picture view, and values of their title attributes are rendered in a text field. (Expressions like Picture.title return the Member instance describing the named member.)

As can be seen in the example, a dedicated Panel class for Picture instances is defined. It is important to note that a relationship between *the class* Picture – not a Picture *instance* – and the instance picturePanel with its child components is established.

Whenever an asset is to be visualized, a suitable UI component can be found depending on its type as shown in Section IV.D. The component implementation instance is used in two ways: the first use is by a UI generator that uses it as a pattern for the generation of code that creates a component implementation at runtime. Examples are Java code that produces a rich Swing client or a JSP page that incorporates HTML fragments.

The second use is the running code itself that adapts a UI to a new or changed asset instance that is to be visualized by it. To this end, the information from the layout model and the links to the domain model are included in the generated code.

B. An Alternative Asset Language for Web Presentations

As indicated in Section II.D the CCM compiler framework allows using custom parsers for specific syntactic forms of model definitions. We currently investigate the use of HTML for layout definitions with embedded tags for the relationship to domain assets.

Specifying user interfaces by HTML with embedded tags allows using web design tools (as long as they leave the custom tags intact). Though this violates the idea of dynamics to some extent, it is important in projects where web designers create visualizations for users of a domain.

There are two custom tags that allow to express the AssetView and MemberView relationships. The semantics of the <assetview> tag is the following: if the asset currently to be displayed is of the type given by the type attribute, then the content of that tag is rendered; otherwise, it is excluded from the page.

The <memberview> tag is evaluated to the asset's member given by the name attribute. Following the example of the *JavaServer Pages Standard Tag Library* (*JSTL*) it optionally allows to define a variable. Then the tag is not expanded to the member's value, but instead the named variable is initialized with it. Later on the variable can be referenced by using the *Expression Language* (*EL*) of JavaServer Pages.

A page definition using this language might look like in the following example:

```
<html>
  …
  <ccmui:assetview type="Document"><table>
    <tr>
      <th>Icon</th>
      <td><ccmui:memberview
        var="icnsrc"
        name="docHandle"
        format="url"
        /><img src="${icnsrc}"></td>
    </tr>
    <ccmui:assetview type="Picture"><tr>
      <th>Title</th>
      <td><ccmui:memberview name="title"
        /></td>
    </tr></ccmui:assetview>
  </table></ccmui:assetview>
  …
</html>
```

In this example a table is rendered for Document instances. If the current asset is actually a Picture, then the table has two rows, one for the image (in this example, docHandle is supposed to always refer to an image file) and one for the title. For all other document instances (e.g., movies) the table contains only the row with the image content.

Of course, users can still alter such enriched HTML layouts since they are processed by a CCM compiler. But this requires users to have knowledge on HTML as well as on the custom tags.

C. View Controllers

There are interactive view elements that update other views, in particular by navigating from one asset to another. View updates can be formulated using the ADL by defining constraints on the view assets.

The following definitions establish synchronization between the subject tree and the extent list in the sample client shown in Figure 7.

```
class TreeListSynchronizer {
  concept
    relationship tree :TreeView
    relationship list :ListView
    constraint listInSyncWithTree
     (tree.selection=na and list.model=na)
      or (tree.selection # na
          and tree.selection.extent
              = list.model)
      onviolation modify list {
        model := tree.selection.extent } }
create TreeListSynchronizer {
  tree := classifierTree
  list := extentList }
```

In this example we use a constraint on the subject tree and the extent list. We define a class for the pair of them and create an instance so that the constraint is active. A TreeView has a selection relationship holding the currently selected tree node. If no node is selected (na as the

null value for no asset) then the list should be empty, meaning its model does not refer to an asset set. Otherwise, we require the model of the extent list to be equal to the extent of the selected subject.

When this constraint is violated the repair code in the `onviolation` clause assigns the list model according to the definition. Note that in case of an empty selection the expression in the `modify` statement results to `na`, and the list model is thus correctly cleared.

D. Controllers to Manipulate Models

Since both the domain model and the view model are formulated using the ADL, no specific technology is required to alter domain assets from within a user interface. The usual asset manipulation language commands can be used to modify assets from the domain model, and the CCM model compiler will create suitable target code from these commands.

To trigger commands on the domain model these can be wrapped in `Action` assets. `Action` is a predefined user interface class that has one relationship `perform`. This relationship can be defined with a constraint to form a kind of function. Instances are used at runtime by interactive components: these can be assigned an `Action` asset, and generated code will use the relationship `perform` to trigger the defined command that can, as a side-effect, modify assets from the domain model.

The following class definition gives an example for an action to store the modifications applied to a `Document` currently visible in the document panel. Let `docPanel` be a reference to the currently used document panel in the client, e.g., `picturePanel` in the case of a `Picture`:

```
class CommitAction refines Action {
  concept
   relationship perform :Asset*
     = modify docPanel.model {
         title := titlePanel.field.text
         …
       } }
```

When the `Action` from the example is related to an interactive UI component, e.g., a menu item, the `modify` statement on the `docPanel`'s model will be executed when interaction takes place (by dereferencing the `Actions`'s `perform` relationship). It updates the currently bound asset in that way that it assigns the updated values from the input fields to the asset's attributes.

An interactive UI component can be a standard component, e.g., a menu item or a button as well as a complex component [25]. E.g., a document panel as sketched above may trigger actions if any of the enclosed text fields has its value changed.

VI. Outlook

The first research direction that needs attention is a higher level of abstraction for the view component model. The initial design was targeted at fat clients and web pages. For these kinds of user interface approaches the presented view model is suitable. But new platforms arise, the most important one being mobile devices, but also interactive whiteboards, tables, etc. For some of these platforms there is no one-to-one mapping from logical view components to component implementations. Instead, some components are realized in a completely different way as they are on conventional graphical user interfaces.

To target such devices more abstract view models are required, and an additional processing step has to create a concrete view for a specific device. Only then the simple construction algorithm based on prototypes as presented in this paper can be applied.

The additional processing step can be realized by model-to-model transformations [26] that generate target asset models from source asset models. The CCM compiler framework can be used for such a model-driven software development approach. Generators that realize device-specific presentation patterns can process source models with more abstract view definitions and create more concrete view models for, e.g., device-specific layouts.

A second major research topic is that of UIs incorporating more than one technology. In practice, such hybrid definitions are regularly used, e.g., web presentations often use a mixture of layout descriptions and program code, like HMTL embedding Java that in turn embeds SQL, or the current trend to enrich web pages with flash animations and JavaScript code, eventually forming AJAX or Flex clients. Yet, a theoretic foundation for such hybrid language approaches is largely missing.

The aim of the CCM approach is to enable domain experts to create models on their own regardless of software development constraints. Currently they do so by using the ADL to formally define their information needs. But for many users the presentation level is the means to argue about models. One goal is to allow users to change presentations in order to express the demand for domain model changes or personalization, and to analyze the changes in order to derive the appropriate domain model changes. Though this should be undecidable for general changes, it might be tractable to recognize certain patterns. This approach would lead to some kind of agile user-centric design approach.

Acknowledgment

The author thanks Joachim W. Schmidt for countless discussions on the topic *CCM* and the seemingly never-ending energy still going into it. Furthermore we appreciate the cooperation with numerous project partners. In particular, we want to thank our colleagues from art history for sharing their insights into the description and management of multimedia artifacts with us. Finally, the author thanks his employer, the T-Systems Multimedia Solutions GmbH, for the opportunity to follow his scientific ambitions.

References

[1] Hans-Werner Sehring, "Adaptive Content Visualization in Concept-oriented Content Management Systems," Proceedings of the First International Conference on Creative Content Technologies, Athens, Greece, 2009.

[2] Gerd Kamp, "Multichannel publishing," OBJEKTspektrum, 2001.

[3] Gustavo Rossi, Daniel Schwabe, and Robson Guimarães, "Designing personalized web applications," Proc. World Wide Web 2001, ACM Press, 2001, pp. 275-284

[4] Hans-Werner Sehring and Joachim W. Schmidt, "Beyond Databases: An Asset Language for Conceptual Content Management," Proceedings of the 8th East European Conference on Advances in Databases and Information Systems, LNCS, vol. 3255, Springer-Verlag, 2004, pp. 99-112

[5] Ernst Cassirer, Language, Mythical Thought, and The Phenomenology of Knowledge. Vol. 1-3, The Philosophy of Symbolic Forms, Yale University Press, 1965.

[6] C.S. Peirce, Collected Papers of Charles Sanders Peirce. Harvard University Press, Cambridge, 1931.

[7] Johan Brichau, "Integrative Composition of Program Generators," PhD thesis, Vakgroep Informatica, Vrije Universiteit Brussel, 2005

[8] Gopal Gupta, "A Language-centric Approach to Software Engineering: Domain Specific Languages Meet Software Components," Electronic Proceedings of the CoLogNet Area Workshop Series on Component-based Software Development and Implementation Technology for Computational Logic Systems, 2002

[9] Uwe Assmann, "Meta-programming Composers In Second-Generation Component Systems," J. Bishop and N. Horspool, Systems Implementation 2000 – Working Conference IFIP WG 2.4, Chapman and Hall, 1998

[10] Yannis Smaragdakis and Don Batory, "Scoping Constructs for Program Generators," technical report, no. CS-TR-96-37, University of Texas at Austin, 1996

[11] Yannis Smaragdakis, Shan Shan Huang, and David Zook, "Program generators and the tools to make them," PEPM~'04: Proceedings of the 2004 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation, ACM Press, 2004, pp. 92-100

[12] Hans-Werner Sehring, Sebastian Bossung, and Joachim W. Schmidt, "Content is capricious: a case for dynamic system generation," Proceedings of the 10th East European Conference on Advances in Databases and Information Systems, LNCS, vol. 4152, Springer-Verlag, 2006, pp. 430-445

[13] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, "Compilers: Principles, Techniques, and Tools," Addison-Wesley, 1986

[14] G. Wiederhold, "Mediators in the architecture of future information systems," IEEE Comp., vol. 25, 1992, pp. 38-49

[15] Mira Mezini, Linda Seiter, and Karl Lieberherr, "Component integration with pluggable composite adapters," Software Architectures and Component Technology, Kluwer, 2000

[16] German Shegalov, Michael Gillmann, and Gerhard Weikum, "XML-enabled work-flow management for e-services across heterogeneous platforms," VLDB Journal, vol. 10, no. 1, 2001, pp. 91-103

[17] Hans-Werner Sehring, "Konzeptorientiertes Content Management: Modell, Systemarchitektur und Prototypen," dissertation (in German), Hamburg University of Science and Technology (TUHH), 2004

[18] Pedro J. Molina, "A Review to Model-Based User Interface Development Technology," Hallvard Trætteberg, Pedro J. Molina, and Nuno Jardim Nunes, "MBUI 2004, Making model-based user interface design practical: usable and open methods and tools, Proceedings of the First International Workshop on Making model-based user interface design practical: usable and open methods and tools, CEUR Workshop Proceedings, volume 103, 2004

[19] Paulo Pinheiro Da Silva, "User Interface Declarative Models and Development Environments: A Survey," Proceedings of DSV-IS2000, LNCS, vol. 1946, Springer-Verlag, 2000, pp. 207-226

[20] Jürgen Falb, Roman Popp, Thomas Röck, Helmut Jelinek, Edin Arnautovic, and Hermann Kaindl, "Fully-automatic generation of user interfaces for multiple devices from a high-level model based on communicative acts," HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences, IEEE Computer Society, 2007

[21] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta, "Applying Model-Based Techniques to the Development of UIs for Mobile Computers," Proceedings of the 6th international conference on Intelligent user interfaces, Santa Fe, New Mexico, United States, ACM, New York, NY, USA, 2001, pp. 69-76

[22] Steffen Lohmann, J. Wolfgang Kaltz, and Jürgen Ziegler, "Model-driven dynamic generation of context-adaptive web user interfaces," Models in Software Engineering, LNCS, vol. 4364, 2007, pp. 116-125

[23] Mittapally Kumara Swamy and Polepalli Krishna Reddy, "An Efficient Context-Based User Interface by Exploiting Temporality of Attributes," APSEC '09: Proceedings of the 2009 16th Asia-Pacific Software Engineering Conference, IEEE Computer Society, 2009, pp. 205-212

[24] Windson Viana and Rossana M. C. Andrade, "XMobile: A MB-UID environment for semi-automatic generation of adaptive applications for mobile devices," Journal of Systems and Software, vol. 81, no. 3, Elsevier Science Inc., 2008, pp. 382-394

[25] Sébastien Romitti, Charles Santoni, and Philippe François, "A design methodology and a prototyping tool dedicated to adaptive interface generation," Proceedings of the 3rd ERCIM Workshop, 1997

[26] Tom Mens, Krzysztof Czarnecki, and Pieter Van Gorp, "A Taxonomy of Model Transformations," Jean Bezivin and Reiko Heckel, Language Engineering for Model-Driven Software Development, Dagstuhl Seminar Proceedings, no. 04101, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005

A Practical Approach to Distributed Metascheduling

Janko Heilgeist*‡, Thomas Soddemann*, and Harald Richter†

*Fraunhofer SCAI, Sankt Augustin, Germany
Email: {janko.heilgeist, thomas.soddemann}@scai.fraunhofer.de

†Clausthal Technical University, Clausthal-Zellerfeld, Germany
Email: harald.richter@tu-clausthal.de

***Abstract*—The paper describes a metascheduler for high-performance computing (HPC) grids that is build upon a distributed architecture. It is modelled around cooperating peers represented by the local proxies deployed by each participating site. These proxies exchange job descriptions between themselves with the aim of improving user-, administration-, and grid-defined metrics. Relevant metrics can include, e.g., reduced job runtimes, improved resource utilization, and increased job turnover. The metascheduler uses peer-to-peer algorithms to discover under-utilized resources and unserviced jobs. A selection is made based on a simplified variant of the Analytic Hierarchy Process that we adapted to the special requirements imposed by the Grid. It enables geographically distributed stakeholders to participate in the decision and supports dynamic evaluation of the necessary utility values. An exemplary decision making process is presented, where user and resource provider jointly decide upon the resource where a job will be computed. Finally, we identify four intrinsic problems that obstruct the implementation of metaschedulers in general.**

***Keywords*-Grid computing; metascheduling; resource discovery; decision making**

I. Introduction

The problem of optimally scheduling jobs across loosely coupled distributed compute resources is still to be solved. Products such as Gridway [2] or Platform's LSF [3, 4] promise to provide out of the box solutions. On closer examination, most of these solutions still have problems ingrained in their design. A major drawback from our point of view is that despite the fact that resources are distributed, some of them work in a centralized fashion and resemble a staging queue in a classical batch system. Others are somewhat distributed, but intrusive as far as the interaction with a site's local batch scheduling system is concerned.

Unlike batch schedulers, a metascheduler supports the exchange of job descriptions across the boundaries of different sites. Such a migration can arise either directly from a user's explicit request for a remote resource or indirectly from a metascheduler's attempt to perform a grid-wide load-balancing. In the latter case, it is the metascheduler's responsibility to discover the best destination. Yet, existing batch schedulers provide no separate point of entry for metaschedulers. A metascheduler can therefore not control the underlying hardware but has to use the site's local batch schedulers — or, more commonly, a grid middleware. It has to extract the job description from a queue at the source site and submit it, customarily assuming the original user's identity, into a target queue. However, migration should be transparent to end users, who should, optimally, never notice that their computations were performed non-locally.

The architecture of a metascheduler can be designed to be either *centralized* or *distributed* [5]. A *centralized* metascheduler is controlled by a dedicated entity that is installed at a single site and has, typically, all the required information for a decision on whether, when and where to migrate a job. That is, it collects data on which sites participate in the grid, which hardware they provide, what the speed of their connection to the grid is, which load their hardware has to bear, and to which degree their queues are filled, etc.

On the other hand, in a *distributed* (or *decentralized*) architecture the metascheduler is split up into multiple independent instances that cooperate among each other. Each instance is separately deployed and represents its site in the grid, that is, it is responsible for all jobs entering and leaving its site via the grid. We call such an instance a *proxy* of the metascheduler. Naturally, a proxy has only limited information to act on compared to the centralized design, as it only gathers data locally and from its neighbors for performance reasons. It is therefore necessary to provide a proxy with sufficient additional input to perform its scheduling. This input can be obtained, e.g., by inter-proxy communication, by a shared pool of available jobs, or by overlapping the local job pools of adjacent sites [5, 6].

While the centralized concept of metascheduling is easier to design, implement, deploy, and maintain, its drawbacks nevertheless outweigh its benefits. We see three crucial

‡Present address: QAware GmbH, Aschauer Str. 32, 81549 München, Germany; Email: janko.heilgeist@qaware.de

Extended version of the paper originally presented at the International Conference on Advanced Engineering Computing and Applications in Sciences, 2009 [1].

problems that need to be addressed: First, a centralized metascheduler always represents a single point-of-failure. Problems with the metascheduler will have an immediate impact on the grid because access to remote resources is disrupted. Failures like a broken Internet link separate parts of a grid from the central scheduler. Furthermore, a successful attack on the scheduler compromises a grid.

Second, the centralized design suffers from reduced scalability when faced with increasing demand. The metascheduler represents a serious bottleneck as it is solely responsible for the migration of jobs in the grid. As the number of sites, users, and jobs grows, it will become more and more difficult to collect all the information that is required to determine the optimal schedule. The common countermeasure to deploy multiple backup servers running the metascheduler software prolongs the decline in performance but leads to additional costs and complexities.

Finally, the largest problem of the centralized architecture is of a political nature. Grids span across multiple administrative zones, companies and institutions, countries, or continents. Each site has its own local policies that a central metascheduler knows nothing about. This can simply be to prefer local users contending for the limited resources or can be as strict as national laws to prevent certain user-groups from accessing HPC resources. Additionally, the willingness of administrators to relinquish control over their hardware to an external entity is unpredictable.

All of the above problems do not emerge in a distributed design. Resilience is increased because a proxy failure affects mostly its home site. Link failures don't disable the grid, as separate sub-grids will continue to function independently. The overall performance scales with the size of the grid as the computation of the schedule is distributed across the grid nodes. And finally, political issues are mitigated by each site's ability to configure its proxy independently.

However, with only limited information to act on, distributed metaschedulers usually produce sub-optimal schedules. While it would be theoretically possible to achieve full information at every proxy through a complete data exchange between all sites, the cost of $O(n^2)$ is prohibitive and makes this idea infeasible. Determining good schedules is an algorithmic problem that is, as of now, best tackled by contenting oneself with approximations. Overall, we favor a distributed approach to metascheduling over the centralized design.

In the remaining sections of this paper, we will describe a distributed architecture that is currently being implemented for the Distributed European Infrastructure for Supercomputing Applications (DEISA 2, [7]). There, its decentralized design will be fully exploited in an international grid environment of autonomous sites.

In Section II, we present the architecture of the metascheduler and describe the general interaction between a proxy and its peers. The P2P algorithms required for this

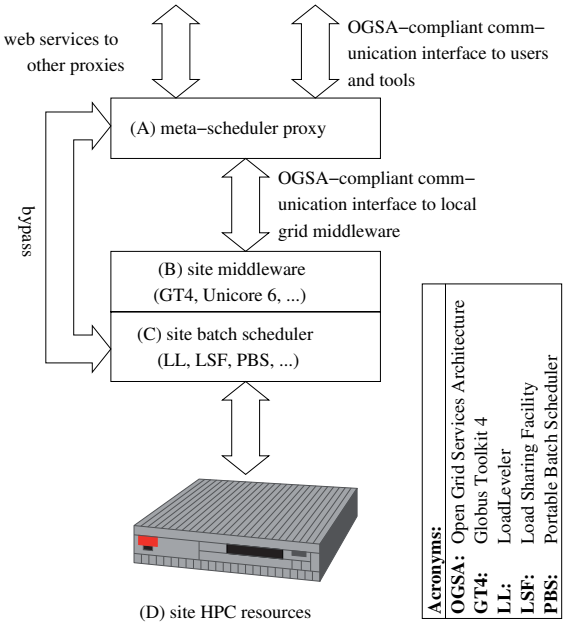


Figure 1: Architecture of the distributed metascheduler at the site level. Relevant references are OGSA [8], GT4 [9, 10], UNICORE 6 [11], LL [12], LSF [3, 4], PBS [13].

communication are portrayed in Section III and the idea of situation-based selection is introduced. Afterwards, the implementation of the metascheduler is explained in Section IV, where we will also present the details of the decision making algorithm. A full example of decision making is shown in Section V, where a decision must be made, which of a given set of resources is selected to execute a job. Then, an overview of related work on methods for resource discovery is given in Section VI. Finally, we finish this paper in Section VII with a conclusion and an outlook.

II. Architecture

The architecture uses cooperating peers rather than a centralized master. Grid administrators install a local proxy software, whose block diagram is displayed in Figure 1. The figure shows the metascheduler proxy (A), the grid middleware (B), and the local batch scheduler (C). The proxy will have to interact with users, remote proxies, grid middleware, and local batch schedulers. Towards the user, the proxy provides web services that are compatible with the standards of the Open Grid Services Architecture (OGSA, [8]). Thereby, users and administrators can continue to use existing tools and client software to monitor jobs and hardware, oblivious to the fact that they communicate with a proxy.

Among themselves, the proxies will communicate using custom-designed but open web services. However, the communication will be mostly restricted to the search for available computing resources and the exchange of scheduling

information. For the actual migration of the job descriptions, the proxies rely on the grid middleware. The interface between proxy and middleware is compatible to the OGSA standard. Thus, the proxy can use any OGSA-supporting middleware. Popular examples include, e.g., Globus Toolkit (GT4, [9, 10]) and UNICORE 6 [11].

The actual metascheduling work of the proxies is performed in three stages by 1.) deciding that a job is to be migrated, 2.) discovering available resources, and 3.) deciding to which target site the job is to be migrated. In the first stage, the proxy determines which job to migrate away from the current site. Jobs may be selected for migration, e.g., because they require resources not available locally, access remote data, or reduce utilization of the resource by fragmenting the schedule. Which criteria are considered is usually determined by the site operator.

Afterwards, *resource discovery* is carried out in an *active* as well as *passive* fashion. In the first case, a proxy actively searches for available resources, while in the second case it only listens for messages sent by other proxies announcing underutilized resources. Having received a collection of offers from its peers, a proxy re-enters the *decision making* phase with the goal to select the best offer to accept. Now, the criteria considered include, e.g., the bandwidth available at a remote site, the cost incurred by using a resource, or other potential benefits offered to the user.

The remote side in this scenario influences the sequence of events twice: first, when it receives a request for computation time, it opts to either provide an offer itself or to ignore the request. Second, if it has disseminated an announcement about available resources, it chooses between all the candidates that took an interest in this offer. The criteria that control the decision in this case could be selected to, e.g., prefer jobs that best utilize the vacant resource, are immediately available for execution, or belong to a site that should be recompensed for previously accepting jobs itself.

The preceeding paragraphs exhibited the fact that scheduling is based on local pieces of information and policies. While the examples focused mostly on criteria that resource providers find profitable, the decision making process can be easily extended to include measures that benefit users or the grid community. Currently, we restrict ourselves to the criteria *queue size*, *average waiting time*, and *waiting time*. They cover an acceptable range of interests insofar as their optimization is designed to result in improved utilization of resources, increased fairness, and greater customer satisfaction. Furthermore, these values can be simply determined and should be deducible from the information provided by any existing local batch scheduler. The real accomplishment will be to find a weighting of the criteria that considers the advantages of all parties involved in a grid environment.

In addition to the illustrated processes, there are other details to be accounted for in the migration of a job description. An offer obtained by a site will usually be valid only for a limited time. An offering site may 1.) reserve the offered timeslot until acknowledged or canceled by the receiving site, 2.) reserve the offered timeslot for a limited time, or 3.) don't provide any guarantees on the period of availability at all. Each of these scenarios requires a receiving site to react accordingly, by either canceling unused offers in a timely fashion, acknowledging offers within their restricted lifetime, or re-issuing a request if previous offers are withdrawn.

In an unstructured grid of independent peers, all these scenarios may occur and have to be supported by a meta-scheduler proxy. For this reason, our decision making algorithm ranks the offers in order of decreasing preference. It iterates through this list and tries to acknowledge the offers successively. The first offer that is available will be accepted and the remaining offers explicitly canceled. Thus, the algorithm will always return the best possible resource for the job regardless of the types of offers received.

III. Resource Discovery with P2P-Algorithms

Over the last decade, peer-to-peer (P2P) networks have changed the way a search algorithm's usability is evaluated. With sizes of hundreds of thousands of simultaneous peers, they have rendered most traditional algorithms obsolete. Instead, a whole new class of *distributed hashtable (DHT) based* search algorithms has been created such as CAN [14], Chord [15], Pastry [16], and Tapestry [17]. The combination of a particular network overlay structure with an efficient routing algorithm makes them especially suited to deal with large distributed networks.

Unfortunately, their approach of hashing a search request is not particular useful in a grid environment. Here, most search requests are multi-criteria queries, e.g., "10 nodes with 32 CPUs each for 5 hours", and composed of ranged criteria, e.g., "10–15 nodes". Hashing such a request results in loss of valuable information. Recent research tackles this problem by mapping these extended queries to the routing layers of, e.g., CAN or Chord. We describe three examples of such algorithms in Section VI. An excellent overview and a taxonomy for grid-enabled DHT-based algorithms is given in [18].

DHT-based algorithms derive their scalability, speed, and fault-tolerance from the fact that queries are distributed evenly amongst the participants of a network. This spread is a direct result of the mathematical properties of the used cryptographic hashing functions such as SHA-1. However, in a grid environment the necessary difference between query values is not always guaranteed. It is typical to encounter limited types of different hardware, common software stacks, and restrictions by resource providers on the requestable numbers of nodes and runtimes of jobs. All of these constraints reduce the possible values an attribute can adopt. As a result, the routing of queries focuses on a few grid nodes and a DHT-based algorithm's benefits are

seriously diminished. Considering the cost and complexity and their fairly insubstantial benefits for grids of the targeted size, we decided against using DHT-based algorithms.

A. Forwarding-based Algorithms

Instead, we employ traditional *forwarding-based* search algorithms. As their name suggests, these algorithms work by forwarding a request from one peer to another peer. Starting at an initial site, a request recursively spreads through the grid until some stop criterion is met. Each receiving peer tries to satisfy the request locally or, else, forwards it to its direct neighbors. Resource offers take the reverse path of a request back to the initial site.

In most instances, different algorithms vary in the way a request is routed, a stop criterion is chosen, or additional information is collected to augment the routing. Two exemplary algorithms that are representative of the class of forwarding-based algorithms are the Breadth-First-Search (BFS) and k-RandomWalks.

Breadth-First-Search: The BFS is the simplest of the forwarding-based algorithms without all the bells and whistles other variants attach to it. A request is endowed with an integral "time-to-live" (TTL) that is chosen by the initial site. Each peer that receives a request decrements the request's TTL by one. If it hasn't reached zero yet, the peer forwards a copy of the request to all of its neighbors indifferently. Afterwards, the peer tries to generate offers matching the request and returns them to the initial site.

Basic details will usually differ between implementations of this algorithm. In general, peers will check whether they have already seen a request before handling and forwarding it a second time. In this spirit, they will not forward a request to the neighbor they originally received it from either. Further, an implementation will usually see to it that each peer aggregates all offers from its neighbors. By bundling the replies at each peer, the number of return messages is kept down. But even with these enhancements, the number of requests grows exponentially with the depth of a search. Also, because the requests continue to be forwarded even if a satisfying offer was discovered, there is no chance to abort a search early.

Still, the BFS exhaustively examines the grid up to the search depth. If there is a matching resource on any of the peers it can reach in TTL hops, then the BFS is guaranteed to discover it. In an unstructured distributed grid, the BFS is the only algorithm with this kind of promise. Nevertheless, the algorithm is too expensive to be employed on its own.

k-RandomWalks: The k-RandomWalks algorithm is a variant of BFS that reduces the number of messages disseminated in the grid. The initial peer issues its request to a maximum of k neighbors. Each subsequent peer will forward the request only to a single random neighbor of its own. Therefore, the number of messages is bound by $k \times$ TTL, and any of the two parameters can be adjusted independently.

The BFS' major drawback — its huge cost in terms of messages — is somewhat reduced by k-RandomWalks. By modifying the parameters k and TTL the character of the algorithm can be changed. Increasing or decreasing TTL directly influences the distance that can lie between a requesting site and the potential offers it receives. Increasing or decreasing k controls the thoroughness with which the grid is searched for results.

However, regardless of the values of the parameters the algorithm will never yield the quality of results produced by the BFS. Instead, it will often find no results even though a matching resource is nearby. For this reason, the k-RandomWalks is rarely employed by itself, too.

B. Situation-dependent algorithm selection

The balance between proper results and incurred costs is crucial when employing forwarding-based algorithms. Naturally, there are more advanced algorithms in this class than the cited examples BFS and k-RandomWalks. Variants such as, e.g., Adaptive Probabilistic Search (APS, [19]) and Distributed Resource Location Protocol (DRLP, [20]) learn from previously performed searches. They collect data about which neighbor returned promising results and adjust their routing correspondingly. Nevertheless, the costs associated with these variants can not be justified by their results in general.

Instead of designing yet another forwarding-based algorithm, we chose to examine the existing methods more closely. It emerged that most algorithms are not to be rejected out of hand. Rather, it is the situation that determines the suitability of a method. No algorithm is always perfectly applicable. But given the right circumstances and an intelligently selected technique, the results can be acceptable. Therefore, it is advantageous to investigate the situations in which a search algorithm is to be invoked.

Coming back to our metascheduler, we discovered two main cases in which a P2P algorithm is necessary: I.) *resource requests*, that is, active inquiries into available resources matching specific requirements, and II.) *resource announcements*, that is, the dissemination of vacant resources that other peers passively listen for. We further identified two sub cases each: I.a.) *necessary* and I.b.) *optional* requests and, on the other hand, announcements of II.a.) *immediate* and II.b.) *future* resource vacancy. Of these four cases, each makes other demands on an appropriate search algorithm.

The key difference between the main cases lies in the way a particular message needs to be distributed. In I.) the system seeks a resource that matches detailed requirements. Thus, it is beneficial to employ algorithms that, e.g., learn from previously performed searches and can route a request specifically to sites where such a resource is known to exist. A similar design could theoretically be used with resource announcements in II.) to find jobs that can take advantage of a vacant resource. However, jobs are transient

compared to HPC resources. Although the probability to find a compatible job is higher on sites with a similar resource, other sites are not to be neglected.

The difference between a situation's sub-classes lies in the urgency, by which a metascheduler requires its results. In the cases I.a.) and II.a.) a higher priority is placed on the search than in the corresponding counterparts I.b.) and II.b.). Due to the higher priority, it is especially important to find actual results and preferably in a speedy manner. Extra cost incurred by an expensive search algorithm is acceptable due to the urgency of the request. For instance, it is more important to obtain jobs for an immediately available resource than for a resource available in the future. In the latter case, a sequence of cheap searches with a reduced chance of success may still yield a result in time.

Each class of situations requires another type of algorithm. It will always lead to an inferior performance if a single technique is chosen. Therefore, we opted to implement various forwarding-based algorithms and dynamically select the particular method to apply in a situation. Compared to DHT-based algorithms, the forwarding-based algorithms are mostly easy to implement and make no demands on a specific network overlay structure. With a basic framework in place, the simplicity of implementing these techniques allows us to support several methods simultaneously.

Each time a P2P search algorithm is about to be invoked, the circumstances are analyzed and the situation is categorized into one of the four classes I.a, I.b, II.a, and II.b. Then the appropriate algorithm is determined and the search initiated. Which algorithm is deemed appropriate can be configured by an administrator of a site. While it is necessary that every proxy supports all employed algorithms, each proxy can be differently configured. However, not every combination will actually lead to improvements. Due to the limited amount of space, we refer the interested reader to [21] for feasible configurations.

IV. Software Design

A huge part of the development time of any complex software system is spent dealing with secondary issues such as handling threads, writing web service stubs and skeletons, and managing database access. Therefore, we decided to build the metascheduler as an enterprise application archive (EAR) that is deployable in any application server conforming to the JavaEE 5 standard [22]. The environment provided by a JavaEE application server supports the programmer by taking over many of the more arduous tasks.

Moreover, an administrator benefits from an easy installation, configuration, and maintenance of the final application. Still, minor adjustments are generally required before an EAR can be deployed in a container since the JavaEE specification grants compliant implementations some leeway. Our development team uses the Apache Geronimo [23] application server because it offers a free, open, and complete implementation of the JavaEE 5 standard. In addition, we plan to actively support Red Hat's JBoss [24] and SUN's Glassfish [25] in the fourth quarter of 2009, too.

The metascheduler is structured into three main modules: 1.) resource discovery, 2.) decision making, and 3.) resource management. Whereas the first two constitute the core modules of the scheduler, the latter embodies the interface to grid middleware or batch scheduler. Additional auxiliary components provide the glue binding the three main modules together. They are secondary to the logical modular design however, and we will neglect them for now.

The *resource discovery* module is chiefly responsible for the discovery of remote resources that match certain specified requirements. This task includes the active search for resources as well as the dissemination of vacant resources to other sites. The module incorporates the various forwarding-based P2P algorithms referred to in Section III-A and exposes them as web services to the remote peers. In cooperation with the decision making module, it implements the situation-based selection of one of these algorithms.

In the *decision making* module, we have concentrated the logic that steers the metascheduler. A variant of the Analytic Hierarchy Process (AHP) [26, 27] is used throughout the system to choose between alternative solutions. Its hierarchical design allows us to consolidate the opinions of several parties into an overall decision. The selection of the target site of a migration can therefore incorporate different viewpoints such as the interests of job owner, resource provider, and grid community. In contrast to the standard AHP, the utilities of an alternative are determined dynamically and with minimal human interaction.

Finally, the *resource management* module serves as an interface between the metascheduler and the lower layers of a site's scheduling stack. It provides a set of abstract methods that allow the reservation and management of timeslots of the underlying hardware resource. Different grid middleware will generally require different implementations of this module. But with an OGSA-compatible default implementation, the metascheduler is adequately equipped to handle a majority of the existing installations on a grid.

A. Resource Discovery

The implementation of a P2P algorithm's web services follows the "contract first" design approach. First, the abstract web services are unambiguously described in the Web Service Description Language (WSDL, [28]). Then, JAX-WS-compliant (Java API for XML Web Services, [29]) tools are used to generate an interface from this description. The interface contains the definitions of the web service methods. Implementing these methods and deploying the code into an application server results in a usable web service.

Each P2P search algorithm supported by a metascheduler proxy is made available as a separate web service under its own URL. Thus, a proxy is characterized by a bundle of

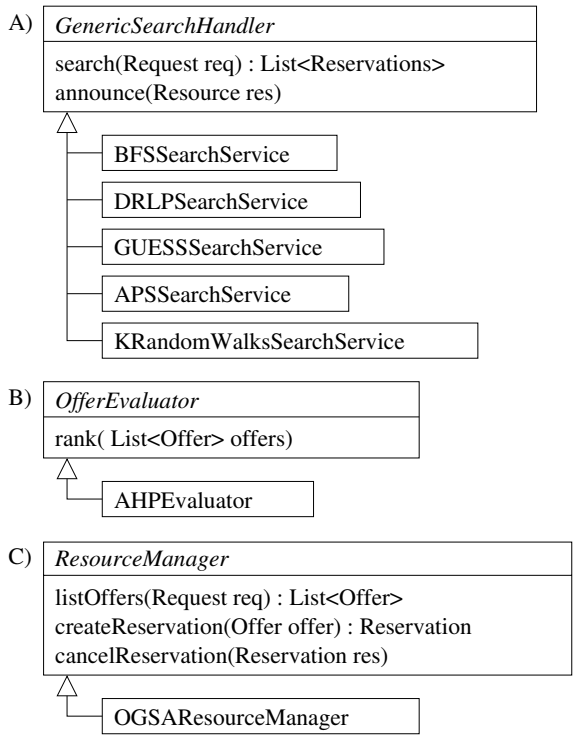


Figure 2: Software modules of the distributed metascheduler.

different service URLs. It is therefore reasonable to provide a specialized "get-to-know-service" that serves as a central point of entry for remote peers. It can be seen as some kind of directory that a remote peer can query to identify the additional services offered by a proxy. We call this service the *gatekeeper service*. Note, that this is not a central directory service but a local web service provided by a proxy to supply information about itself in agreement with the distributed nature of the design.

The information supplied by a gatekeeper currently contains a user-friendly name and a list of all locally available web services. The name is used exclusively as a convenience for users, while proxies identify themselves by their gatekeeper URLs instead. It can be specified by an administrator to label a managed resource, e.g., "IBM Power6 system at RZG". In the list of services, each available web service is identified by its fully-qualified name (FQN) and mapped to its respective local URL. We established the provided information as scarce and restricted to the absolute minimum. It is easily extendable to include, e.g., a free text description of a resource, usage policies, service level agreements, detailed costs, etc.

Search algorithms are defined via a generic abstract web service displayed in part A) of Figure 2. An actual algorithm needs to implement only two methods: `search` and `announce`. The `search` method takes a resource request as an argument and returns a list of matching reservations. Where these reservations actually come from, e.g., multiple reservations from a queried proxy itself or additional reservations from remote sites, depends on the particular algorithm considered. Naturally, reservations will have to contain the appropriate pieces of information that a querying site requires to positively identify the source of an offer.

As of now, a resource request incorporates a requirements definition in the format of the Job Submission Description Language (JSDL, [30]) plus a data block each for search algorithm and decision making algorithm. These blocks are used to map a received request to the appropriate algorithm implementation and, additionally, used by the particular algorithms to attach custom data to a request. Regarding the search algorithms, such a piece of custom data can, e.g., include timeout or original source of a request, routing directives, etc.

The `announce` method takes a structure similar to the request as an argument but has no return value. An argument's JSDL component is used to describe the offered resource including the address of the offering site. A return value is not required, because a site that wishes to apply for the resource, is required to submit its request in a separate step. Thus, an announcement can be handled as a "fire and forget" message. By removing the reply message for this type of search, an announcement's costs is kept down.

B. Decision Making

The module in charge of *decision making* is a simplified implementation of the Analytic Hierarchy Process (AHP) [26, 27, 31] displayed in part B) of Figure 2. It considers various facets of an item to sort a given set of similar items in order of preference. The module is applied in several stages of the metascheduling process to rank available timeslots, received offers, migration candidates, etc. By making the algorithm a replaceable module, the different sites could, in theory, employ different decision making modules without interfering with each other.

The AHP is a hierarchical multi-criteria decision making algorithm originally developed for the domain of economics. Its decision is based on a tree of criteria that have been chosen as relevant to the decision at hand. Figure 3 shows an exemplary AHP tree with two levels that combines several criteria relevant to a migration decision. Inner nodes of a tree define meta-criteria that are fully described by their child nodes. Leaf nodes represent criteria where the utility of an option can be explicitly determined. Each criterion in the tree has a weight assigned by a human. These weights determine the importance of a criterion with regard to its corresponding parent node.

The ranking of alternatives starts at the leaves of a tree. Here, each alternative item is evaluated and has its utility computed. In the original AHP, an item's utility is determined based on pairwise comparisons provided by a human

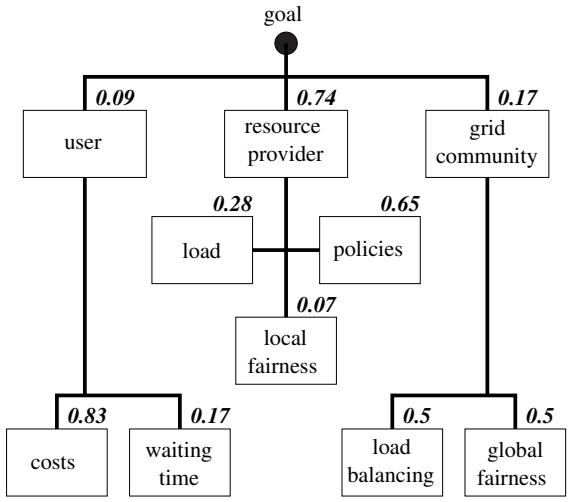


Figure 3: Exemplary AHP hierarchy with multiple parties. The weights of each node's children add up to 1.0.

decision maker. For this purpose, each pair of items has to be judged and the preference for an item has to be expressed on a scale of nine values. These values represent notions from "equally preferred" to "extremely preferred" defined by Saaty [26]. A pairwise comparison matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ is the result of this rating process. Finally, the normalized principal Eigenvector of A is determined and its kth entry is used as the utility of the kth alternative.

Utilities with regard to inner nodes of a tree are computed recursively. At every node, the utility of an alternative is determined as the weighted sum of corresponding values at the child nodes. That is, for each inner node the utilities at its immediate child nodes are multiplied by the corresponding child's weight. These products are then added and result in an alternative's utility with respect to the parent node. As a by-product of using normalized utilities at the leaves and using normalized weights, the process also produces normalized utilities at each inner node. The obtained values are then used at the root node to rank alternatives from best to worst.

Usually, consent in an AHP context is reached by personal interaction: stakeholders come together to debate pairwise comparisons. In a grid, involved parties are geographically distributed, yet situations where a decision has to be reached are abundant. Because debate and direct interaction are infeasible, we exploit the hierarchical structure of a criteria tree to account for distinct opinions. Each stakeholder is allowed to build its own tree from a common set of criteria and assign its own weights. These separate trees are brought together under a common root to construct the final tree.

At this point, the persistent normalization protects an automated decision making process from malicious users. As utilities at each subtree's root are normalized, the expressed opinions are initially of equal strength. In the next step, every subtree is connected to the root of the final hierarchy with its own weight. These *root weights* exclusively control the share of participation given to a particular stakeholder. In a grid environment, it is reasonable to assume that resource providers will reserve the right to define these values to themselves. As root weights can be defined on a per resource basis, different providers will not interfere with each other. Additionally, a provider can encourage participation — and even advertise the possibility thereof — without loss of control.

In our metascheduler, three stakeholders participate in a decision: 1.) the owner of a job, 2.) the provider of a resource, and 3.) the grid community. The lifetime of a subtree will vary according to the party that defined it. A user's tree is optionally created as part of the job submission process and permanently attached to a job description. It exists only as long as a job remains in the grid. A provider's tree is specific to a particular resource and part of a proxy's configuration. It will generally be constant for extended periods of time and will be used in every local decision making process. Finally, a grid community's tree is defined at the grid level and assumed to be part of the initial agreement to found a grid. It is configured at the proxy level, too, but identical across all participating sites. Typically, this tree will be the most static of the three subtrees. The root weights are configured in the same XML document that sets provider and grid community subtrees.

Furthermore, we extended the original AHP to dynamically compute utilities at the leaves of a tree. Such an extension is necessary, before AHP can be employed in a grid environment. Here, alternatives are defined by measurements or predictions such as waiting time (in minutes), cost (in dollars), and utilization (in percent). Again, AHP is clearly focused on human interpretation of the alternatives. Saaty gives several examples in [27], where a human decision maker's preference may differ significantly from the values suggested by a standard scale. We tried to accommodate this view and simultaneously make the algorithm feasible for a grid environment. Hence, we made the computation of utilities configurable based on an underlying criterion.

We provide two evaluation methods in the stock installation of a metascheduler: a direct mapping module and a hyperbolic tangent [32] based module. The direct mapping module allows a provider to map ranges of input values to specific utilities. It supports open ranges, ranges bounded above or below, and exact values. The resulting utility function is obviously non-continuous. It might be tempting to use this method for continuous criteria to express, e.g., that only waiting times of up to 5 hours are acceptable. However, this is generally a bad idea, because it leads to situations where the priorities do not reflect a stakeholder's intent. A job with a predicted waiting time of 4:59h is in most cases only marginally more preferred than a job with

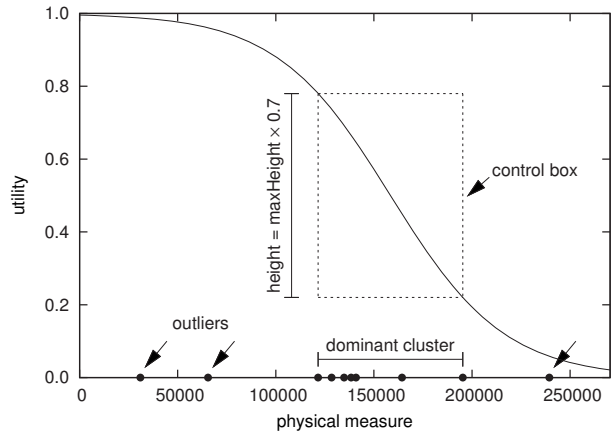


Figure 4: Continuous utility function for minimization criteria based on the hyperbolic tangent. The control box is determined automatically from the distribution of the measures.

5:01h waiting time. Thus, this method should only be used for discretely valued criteria such as user IDs, group IDs, queue names, etc.

In contrast, the evaluation method based on a hyperbolic tangent is designed for continuous criteria. It reflects the idea, that distinct utilities should be computed for the range of input values where the majority of the alternatives are located. Above average quality should yield higher utilities but eventually be bounded. An analogous reasoning is true for below average quality. We chose the hyperbolic tangent as the underlying utility function because it converges to 0 (resp. 1) for small (resp. large) input values. The notion of a *control box* is used to scale and shift the hyperbolic tangent with the aim of focusing its transition from 0 to 1 on average alternatives (see Figure 4).

Upon receiving a set of alternatives, the evaluation module first starts to determine the dominant cluster with regard to a given criterion. It iteratively aggregates the closest input values until a cluster exceeds 51% of all values. Horizontal position and width of a control box are set to reflect the position and width of this dominant cluster. Vertically, the box is centered around the midpoint 0.5 of the available utilities. The height of a control box is at most 0.8 and scaled with the ratio of elements in the dominant cluster. Determining a control box is independent of whether the current criterion is about maximizing or minimizing measures.

Finally, the hyperbolic tangent is scaled and shifted to match a control box, i.e., the curve runs through the lower left and upper right corners of a control box for maximizing criteria. Accordingly, it runs through the upper left and lower right corners for minimizing criteria. The resulting function is used to map input values to raw utility values, which are then normalized to bring them in a form suitable for use in AHP.

Direct mapping and hyperbolic tangent based evaluation represent two methods designed for non-continuous and continuous criteria, respectively. In Section V, we present a full example that uses both mapping techniques. However, the metascheduler is easily extendable with additional evaluation methods.

C. Resource Manager

The interface between metascheduler and locally installed grid middleware or batch scheduler is represented by the resource manager module displayed in part C) of Figure 2. It provides the means to inquire for a set of offers matching a request's JSDL description and, optionally, to fix such an offer into a reservation. The definitions of offer and reservation are deliberately kept as broad as possible to account for different systems, which a metascheduler is required to interact with.

Conceptually, an offer represents a timeslot on some resource that is endowed with additional metadata. The contents of the structure must allow the resource manager to identify the offered resource and assign a job correctly to it. The attached pieces of metadata constitute a mapping of arbitrary criteria to corresponding measures taken or predicted by a resource manager. An offer satisfies the preconditions put forth by the decision making module. Thus, a set of offers can be ranked in order of preference if some weighting of the criteria is supplied externally.

As mentioned previously, the metascheduler currently supports the criteria queue size, average waiting time, and waiting time. These points of comparison were selected because every existing batch scheduler should be able to supply them easily. However, the metascheduler is not restricted to any fixed set but can be configured to understand arbitrary criteria. Thus, the set can be freely extended as long as a resource manager knows how to obtain the related values.

A reservation is just a wrapper around an offer that optionally assigns an owner to it. Whether this assignment is for a limited period of time, until the reservation is explicitly canceled, or not authoritative at all is up to the particular resource manager or its site. The main reason for a separation of offer and reservation was to distinguish between the mere availability of a timeslot and its dedication to a particular party.

V. Decision Making — An example

Two unique features distinguish the described metascheduler from existing solutions. First, its concurrent use of multiple forwarding-based search algorithms that selects the best algorithm for a given situation. Second, its generic decision making algorithm that supports multiple stakeholders for each scheduling decision.

The same basic principles of decision making will eventually be used in both search module and generic DM module. Forwarding-based search algorithms are then plain implementations of existing techniques that generally require no customization. For this reason, the following example will focus on the new concepts in decision making. We will show a selection process that is common to scheduling: which resource from a given set of alternatives will be used to execute a compute job. In the process, we will pay particular regard to the observance of individual interests in a joint decision.

A user has used the metascheduler to submit a job that requires 9,216 CPU-hours to run to completion. It is assumed, that the job is moldable and will scale perfectly to any number of CPUs, i.e., it can run in 3 days on 128 CPUs or in 18 hours on 512 CPUs. Such an assumption is rather naive and more realistic models for the speedup of parallel programs exist. For instance, Cirne and Berman [33] presented a generator that models sets of moldable jobs according to Downey [34]. However, an explanation of these concepts is beyond the scope of this example.

As part of the submission process, the user selected three criteria that he wishes the metascheduler to optimize. His goals are to minimize the *cost* of computing his job, to minimize the *waiting time* until his computation will start, and to minimize the overall *completion time* until the results are returned to him. Then, he used the method established by AHP to judge the three criteria with regard to their relative importance on a verbal scale. The user stated that he preferred lowering the cost strongly over lowering the waiting time and, moderately over lowering the completion time. Likewise, he preferred lowering the completion time moderately over lowering the waiting time. From these judgments, normalized weights were derived for cost, waiting time, and completion time as, respectively,

$$w_{\text{cost}} = 0.637, \quad w_{\text{wait}} = 0.105, \quad w_{\text{comp}} = 0.258.$$

Note that weights derived through the AHP will always add up to 1.0.

The metascheduler issued a search on behalf of the user's job to acquire offers for a matching slot on an HPC resource. Eventually, the search returned five offers from three different resources. For the sake of simplicity, we are working with several assumptions about these resources and offers. First, the discovered resources were filtered such that they fulfill all necessary requirements with regard to software, disk space, memory, etc. Second, the user is authorized to use any of the three HPC resources. And third, all resources belong to the same resource provider. This third assumption is most restrictive, but allows us to keep the number of parties participating in the decision down to the user and his provider.

Like the user, the resource provider selected two criteria that the metascheduler is supposed to optimize. The first goal is to maximize the average load of the resources, and, therefore, to favor resources with a low *utilization* during scheduling. Then, explicit *queue priorities* were manually defined for queues and job classes. On resource A, medium-sized jobs that used at most 64 CPUs are favored over large jobs that used up to 512 CPUs in parallel. This decision was made to increase the number of users served simultaneously and, therefore, maximize the perceived fairness. Also, guests on resource C are penalized by a lower priority with regard to regular users for political reasons. The second goal is to stay true to these priorities and consider them in the scheduling process. Finally, both criteria were judged by an administrator who stated that better utilization was moderately preferred over adherence to the queue policies. The resulting normalized weights are

$$w_{\text{util}} = 0.750, \quad w_{\text{prio}} = 0.250$$

for *utilization* and *queue priorities*, respectively.

Furthermore, the root weights that govern the share of participation were established in cooperation with a user representative. Instead of using the AHP, the values were obtained by external means and set to

$$w_{\text{user}} = 0.16, \quad w_{\text{prov}} = 0.84,$$

for *users* and *provider*. In other words, a share of 16% in the overall decision is granted to users.

Table I shows the relevant characteristics of the three resources that the metascheduler found. For each resource the cost in dollars per CPU-hour was chosen consistent with the values calculated by Walker [35]. All remaining properties were defined based on on our experience with HPC resources. Table II lists five offers and their characteristics with regard to the decision making criteria. An offer's cost is determined by multiplying the job size of 9,216 CPU-hours with the corresponding resource's cost. Waiting times were picked randomly to provide some variance in the quality of each offer. Finally, completion time is derived from the waiting time and the job's runtime using the maximum number of CPUs allowed for each queue. Utilization and queue priority associated with an offer can be obtained from Tab. I.

Now, the next step is to evaluate all offers with regard to all criteria specified by the participating decision makers. For cost, waiting time, completion time, and utilization, the style of automatic prioritization illustrated in Section IV-B is used. Queue priorities, however, already represent priorities themselves. They are an example for a direct mapping technique where discrete values — in this case the queue names — are mapped to static utility values. On resource C, however, the mapping can be implemented purely numerical by detecting user or guest status based on the user ID. Still, the resulting utilities are not normalized yet. Therefore, they are divided by the sum of all five queue priorities to fit into the generic hierarchical prioritization scheme.

Resource	Cost ($/CPU/hour)	Utilization	Queues	Queue Priority	Policy
A	0.07	0.95	jumbo	1.0	max. 512 CPUs
			medium	1.2	max. 64 CPUs
B	0.045	0.90		1.0	max. 128 CPUs
C	0.10	0.70	users	1.0	max. 64 CPUs
			guests	0.6	max. 64 CPUs

Table I: Characteristics of the available HPC resources.

Offer-ID	Resource	Queue	Cost ($)	Waiting time (hours)	Completion time (hours)
offer-1	A	jumbo	645.12	36	54
offer-2	A	medium	645.12	16	160
offer-3	B		414.72	20	92
offer-4	C	users	921.60	4	148
offer-5	C	guests	921.60	8	152

Table II: Characteristics of the resource offers.

The four utility mappings with automatically derived hyperbolic tangents are shown in Figure 5. It can be seen, that the algorithm adjusts itself to distribution and magnitude of the underlying measures regardless of the actual criterion. For instance, most of the waiting times lay in the range between four and 20 hours with a negative outlier at 36 hours. Here, the algorithm maps the average offers to (unnormalized) utilities between 0.18 and 0.82, while it singles out "offer-1" and assigns an utility that marks it as unacceptable. On the other hand, three out of five completion times fall between 148 and 160 hours with two positive outliers at 54 and 92 hours. Again, the mapping assigns values in the middle spectrum of utility to the cluster, but it awards "offer-1" and "offer-3" with utilities close to the maximum. Also note, how the algorithm accommodates the height of the control box to the share of options in the dominant cluster. Figure 6 shows the normalized utilities for all offers and criteria in comparison.

In a final aggregation step, the singular utilities are combined for each set of elements in the criterion hierarchy until the root is reached. To illustrate, let's consider "offer-1", which is characterized by five independent utilities:

$$u^{(1)}_{\text{cost}} = 0.198, \quad u^{(1)}_{\text{wait}} = 0.005, \quad u^{(1)}_{\text{comp}} = 0.279,$$
$$u^{(1)}_{\text{util}} = 0.080, \quad u^{(1)}_{\text{prio}} = 0.208.$$

A weighted sum of these values is calculated to determine the utilities from the point of view of the user and the resource provider, respectively. The user's utility for "offer-1" is, therefore,

$$u^{(1)}_{\text{user}} = w_{\text{cost}} u^{(1)}_{\text{cost}} + w_{\text{wait}} u^{(1)}_{\text{wait}} + w_{\text{comp}} u^{(1)}_{\text{comp}} = 0.198$$

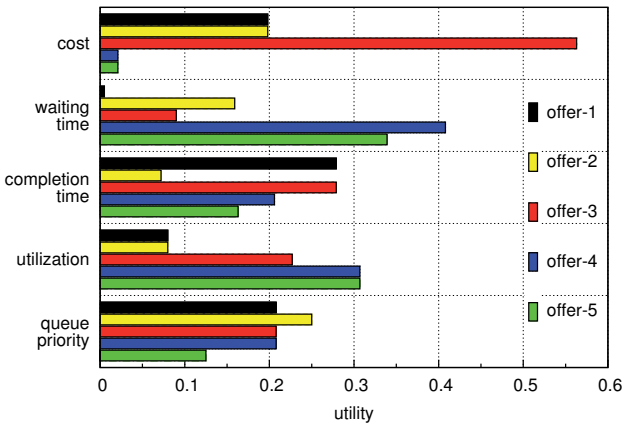


Figure 6: Utilities of each resource offer with regard to the leaf criteria.

and the provider's utility

$$u^{(1)}_{\text{prov}} = w_{\text{util}} u^{(1)}_{\text{util}} + w_{\text{prio}} u^{(1)}_{\text{prio}} = 0.112.$$

Finally, these utilities are combined with the root weights to obtain the final overall utility of the first offer.

Intermediate utilities, i.e., utilities with regard to user, provider and the overall goal, are shown in Figure 7. It is noticeable, that the overall utilities are closely related to the provider utilities. The reason for this similarity is, of course, the large share of the root weights granted to the resource provider. Nevertheless, the user's strong preference for the third offer is not ignored. Despite his minor power, the third option is judged as marginally more preferable than the fourth option in the final prioritization. Using the

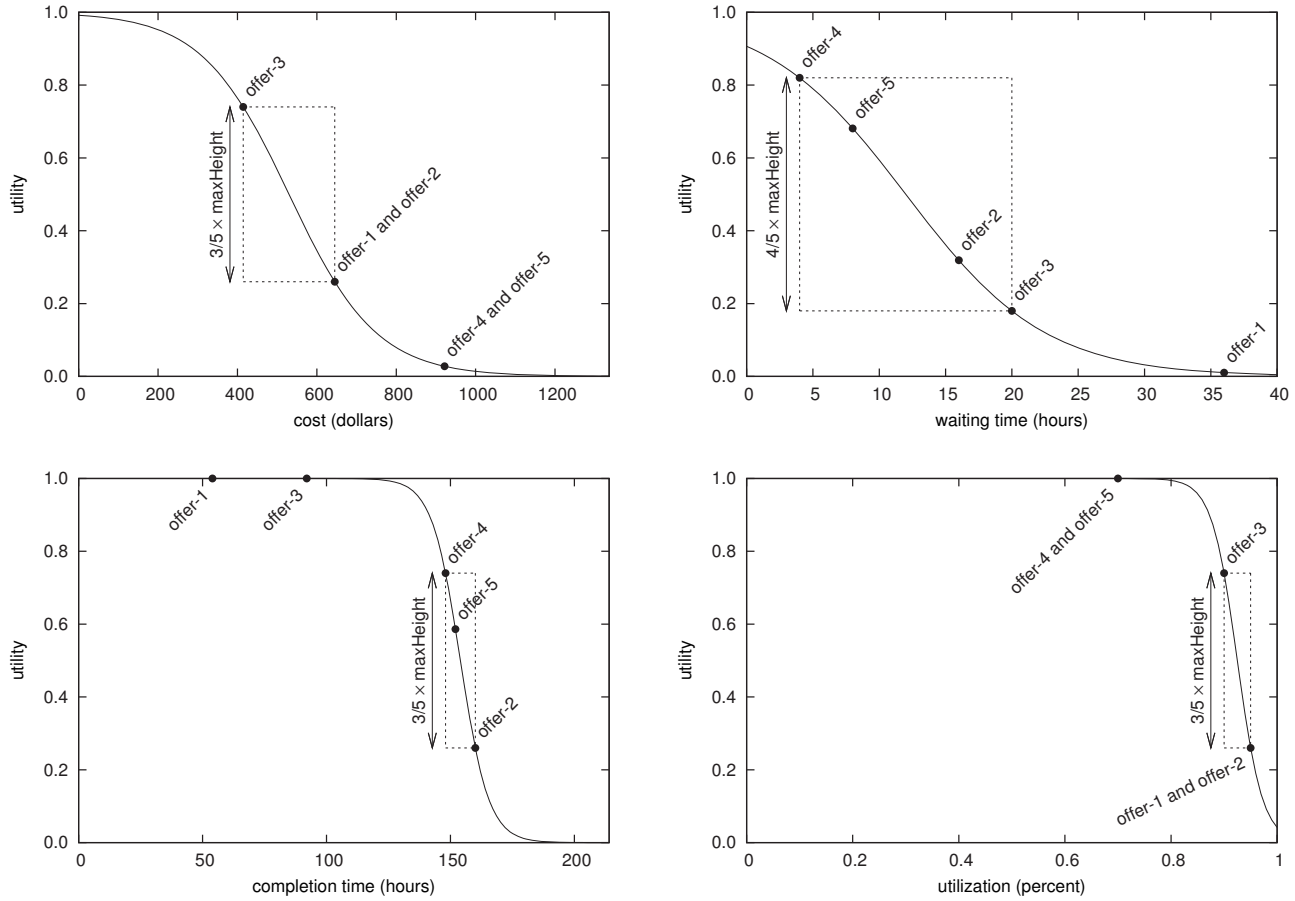


Figure 5: Automatic utility mappings derived for the criteria cost, waiting time, completion time, and utilization.

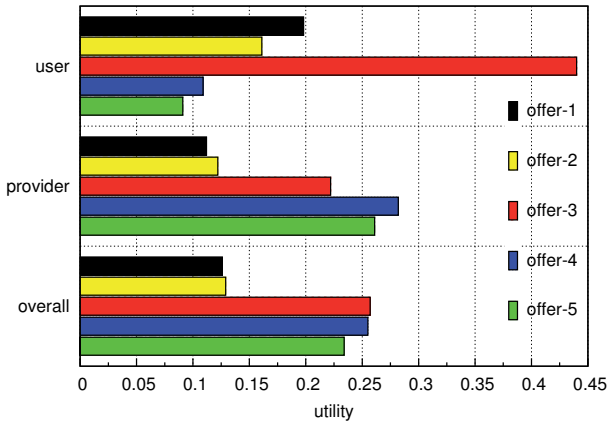


Figure 7: Per-party and overall utilities of each resource offer.

overall utilities as a guideline, the metascheduler would try to acknowledge the offers in decreasing order of preference

$$\text{offer-3} > \text{offer-4} > \text{offer-5} > \text{offer-2} > \text{offer-1}.$$

The first offer that would actually be accepted by both sides of the bargain is assigned to the job; remaining offers are subsequently canceled.

VI. Related Work

Current grid scheduling solutions focus on centralized resource discovery approaches. UNICORE 6 [11] is a grid middleware that is widely deployed, e.g., in the DEISA 2 grid [7] and in parts of the German D-Grid [36]. It performs resource discovery via a single registry called *Common Information Provider* (CIS). The CIS collects static and dynamic information on a grid's resources, which is provided by *CIS Information Providers* (CIP) monitoring each resource. A CIP publishes an Atom-feed that is periodically polled by the CIS by means of a web service protocol. The migration of job steps in a workflow is performed by so-called *Service Orchestrators*. They employ different

brokering strategies, which in turn use the information from the CIS to reach their decisions.

A similar approach is taken by the second major grid middleware Globus Toolkit 4 [9, 10]. The resource management is contained in the *Monitoring and Discovery Service* (MDS), which consists of *Aggregator Framework*, *Index*, and *Trigger*. Index is the counter-part to UNICORE's CIS. There are one or more index servers per virtual organization (VO) that store all the information about a VO's resources. The Aggregator Framework is used locally to monitor a resource by periodically spawning shell scripts. Obtained data is then pushed into the central index.

Finally, the Trigger component can be configured to perform actions whenever a predefined event occurs. While the Aggregator Framework is the primary way of updating an index, Trigger can also be used to poll information into the registry. Globus Toolkit 4 has no support for automatic migration of jobs by itself. It requires the help of additional software such as Gridway metascheduler [2] to provide this functionality. The Gridway information manager accesses the MDS index via its web service interface to discover remote resources. Based on this data it reaches its scheduling decisions. Another centralized scheduling approach that resides on top of GT4 is described in [37].

Distributed designs for resource discovery have previously been produced in theory. Here, the focus has been on the transfer of grid principles to the domain of P2P networks. Examples for such algorithms are MAAN [38], Squid [39], and QuadTree [40]. We sketch these three methods because they represent interesting and unique approaches to the problem of mapping ranged criteria and multi-criteria queries to the underlying DHT-based substrates. Many more grid-enabled DHT-based algorithms are described in [18].

Multi-Attribute Addressable Network (MAAN) uses two separate procedures to augment the underlying Chord algorithm. First, multi-criteria requests are resolved by maintaining distinct hashing functions and querying separately for each involved attribute. Second, MAAN employs a *locality preserving hashing function*, that is, a function that maintains the order relation between numerical values for their hashed values. Each query for a ranged criterion is then represented by the actual sought-after value and the minimum and maximum allowed values. MAAN forwards a request from the node responsible for the minimum value via its successors in the Chord ring to the node corresponding to the maximum value. Whatever results have been found at this point are returned to the querying node.

The Squid [39] algorithm is based on a Chord routing layer, too. It interprets the d criteria or attributes relevant to a grid-style query as spanning a d-dimensional space. This space is mapped to a 1-dimensional space by means of *space filling curves* (SFC). The resulting scalars are conventionally hashed and mapped to grid nodes using the Chord routing layer. Each multi-criteria query corresponds to a point or, in case of ranged criteria, a sub-region of the d-dimensional space. It is mapped by the SFC to distinct clusters of scalars. Each cluster corresponds to individual nodes in the grid, which have to be queried separately. The results obtained by several of these traditional DHT-queries are then merged to receive the final results of a grid-enabled DHT-query.

Another method that gets by with a single Chord DHT-layer is the QuadTree algorithm [40]. It recursively subdivides the d-dimensional attribute space with the help of quad-trees. Each block is identified by its centroid and mapped to a Chord node. A ranged multi-criteria query intersects with one or more blocks. To execute a query, those relevant blocks are determined and the corresponding grid nodes are contacted. The combined results of these distinct queries form the final response. The performance of the QuadTree algorithm is further improved by a cache. Here, each node stores the addresses of its immediate children in the tree to reduce the number of lookups performed.

With regard to decision making, the scheduling heuristic of Maui Cluster Scheduler [41] bears certain similarity to the hierarchic approach we propose in this article. However, Maui's hierarchy is static and weights are the only mechanism by which an administrator can modify the criteria. A direct modification of the tree is infeasible, since aggregation of the individual per-criteria priorities is not generic. Instead, criteria derive priorities from their child criteria in unique ways, e.g., applying minima or maxima along the way. Furthermore, the definition of weights is restricted to the administration of a resource, and, in contrast to our design, users can not participate in a scheduling decision. Maui, though, supports fairshare criteria that favor a fair distribution of the resource amongst the users. Thus, user interests are considered in the scheduling at least indirectly.

VII. Conclusion and Future Work

We have presented our implementation of a hybrid algorithm for a distributed metascheduler that efficiently links available resources and matching jobs. It supports the exchange of jobs between resources and, thereby, achieves improved resource utilization and shorter turn over times. The metascheduler is currently being implemented and will be deployed as part of the DEISA2 grid in fall of 2009.

The design of the metascheduler fully supports the JavaEE specification's security framework. Accordingly, the metascheduler can be extended to integrate existing security infrastructure including virtual organizations, Community Authorization Service [42], and Shibboleth [43]. Towards middleware or local batch schedulers it acts transparently, hence, letting these layers provide their own security arrangements.

Finally, we provided an example for a decision making process that is common to scheduling. The example showed how the prioritization algorithm uses clustering to reward or penalize options of particular merit or demerit, respectively.

Furthermore, it illustrated how the opinion of a job's owner can be regarded in a decision without loss of control for the resource provider.

During the design and implementation of the scheduler, we gained intense experience with input queues and local resource management systems (LRMS) of HPC sites. In cooperation with the authors of [37], we have isolated four intrinsic problems that have to be solved to make metascheduling a fully working feature in the future.

First, the grid scheduler generally has only the role of a "power user" from the perspective of the LMRS and has to compete with other users, e.g., other grid schedulers in the same grid. As a consequence, an optimal schedule is not possible for any individual metascheduler.

Second, grid schedulers have no control over a site's policy, and, therefore, over the prioritization of the jobs waiting in queues. Thus, no control exists over the respective local resource management system for the metascheduler.

Third, every site uses a custom configuration of queues, and processors can be shared among queues or dedicated exclusively. Usually, the information from the LRMS does neither allow one to reliably determine the number of free processors nor does it allow one to determine the total number of processors. The only statistics commonly available are the number of running and waiting jobs. Some sites do not even provide this information because of nondisclosure. To conclude, local schedulers currently do not provide sufficient information for a good schedule.

Finally, resources are highly utilized and waiting times at clusters can last up to hours. Therefore, queue waiting time considerably exceeds the actual execution time for small jobs. Since small jobs constitute the majority of all jobs, input queue waiting time is the dominant factor. However, input queue waiting time and input queue length have shown to be not continuously differentiable functions over time. Instead, they can vary within minutes by a factor of thousand or more. They behave more like fractals than continuous functions. This makes predictions of future queue waiting times and queue lengths a delicate task. However, scheduling always relies on such predictions. Future work will focus on finding solutions to these obstacles.

References

[1] J. Heilgeist, T. Soddemann, and H. Richter, "Design and implementation of a distributed metascheduler," in *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP '09. Third International Conference on*. IEEE Computer Society, Oct. 2009, pp. 63–72.

[2] "Gridway," http://www.gridway.org, The Globus Alliance, [accessed: 2010-06-19].

[3] "Platform Computing," http://www.platform.com, [accessed: 2010-06-19].

[4] S. Zhou, "LSF: Load sharing in large-scale heterogenous distributed systems," in *Proc. Workshop on Cluster Computing*, 1992.

[5] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of job-scheduling strategies for grid computing," in *GRID '00: Proc. 1st IEEE/ACM Intl. Workshop on Grid Computing*. Springer-Verlag, 2000, pp. 191–202.

[6] Q. Wang, X. Gui, S. Zheng, and Y. Liu, "De-centralized job scheduling on computational grids using distributed backfilling: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 14, pp. 1829–1838, 2006.

[7] "Distributed European Infrastructure for Supercomputing Applications (DEISA 2)," http://www.deisa.eu, [accessed: 2010-06-19].

[8] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw *et al.*, "The Open Grid Services Architecture (OGSA)," http://www.ogf.org/documents/GFD.80.pdf, Open Grid Forum, July 2006.

[9] I. Foster, "Globus toolkit version 4: Software for service-oriented systems." in *IFIP Intl. Conf. on Network and Parallel Computing*, ser. Lecture Notes in Computer Science, H. Jin, D. Reed, and W. Jiang, Eds., vol. 3779. Springer-Verlag, 2005, pp. 2–13.

[10] "Globus Toolkit," http://www.globus.org, The Globus Alliance, [accessed: 2010-06-19].

[11] "UNICORE," http://www.unicore.eu, Jülich Supercomputing Centre, [accessed: 2010-06-19].

[12] *IBM Load Leveler: User's Guide*, IBM Corp., Sept. 1993.

[13] R. Henderson and D. Tweten, "Portable Batch System: External reference specification," NASA Ames Research Center, Tech. Rep., 1996.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proc. 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2001, pp. 161–172.

[15] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proc. 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2001, pp. 149–160.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proc. IFIP/ACM Intl. Conf. on Distributed Systems Platforms*. Springer, 2001, pp. 329–350.

[17] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE J. Sel. Area Comm.*, vol. 22, no. 1, pp. 41–53, 2004.

[18] R. Ranja, A. Harwood, and R. Buyya, "Peer-to-peer based resource discovery in global grids: A tutorial," *IEEE Commun. Surveys Tuts*, vol. 10, no. 2, pp. 6–33, 2008.

[19] D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search for peer-to-peer networks," in *P2P '03: Proc. of the 3rd Intl. Conf. on Peer-to-Peer Computing*. IEEE Computer Society, 2003, p. 102.

[20] D. Menascé and L. Kanchanapalli, "Probabilistic scalable P2P resource location services," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 2, pp. 48–58, 2002.

[21] J. Heilgeist, T. Soddemann, and H. Richter, "Algorithms for job and resource discovery for the metascheduler of the DEISA grid," in *Advanced Engineering Computing and Applications in Sciences, 2007. ADVCOMP 2007. International Conference on*. IEEE Computer Society, Nov. 2007, pp. 60–66.

[22] "Jsr-000244: Java EE 5.0 specification," http://jcp.org/en/jsr/detail?id=244, Sun Microsystems, Inc., [accessed: 2010-06-19].

[23] "Apache Geronimo," http://geronimo.apache.org, The Apache Software Foundation, [accessed: 2010-06-19].

[24] "JBoss Enterprise Middleware," http://www.jboss.org, Red Hat Middleware, LLC., [accessed: 2010-06-19].

[25] "Sun GlassFish," https://glassfish.dev.java.net, Sun Microsystems, Inc., [accessed: 2010-06-19].

[26] T. Saaty, *Multicriteria Decison Making: The Analytic Hierarchy Process*, 1988, revised and published by the author; Original version published by McGraw-Hill, New York, 1980.

[27] ——, "How to make a decision: The Analytic Hierarchy Process," *Eur. J. Oper. Res.*, vol. 48, no. 1, pp. 9–26, 1990.

[28] "Web Service Description Language (WSDL)," http://www.w3.org/TR/wsdl, World Wide Web Consortium (W3C), [accessed: 2010-06-19].

[29] "The Java API for XML-based Web Services (JAX-WS)," http://jcp.org/en/jsr/detail?id=224, Sun Microsystems, Inc., [accessed: 2010-06-19].

[30] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job Submission Description Language (JSDL)," http://www.ogf.org/documents/GFD.136.pdf, Open Grid Forum, July 2008.

[31] T. Saaty, *Fundamentals of the Analytic Hierarchy Process*. RWS Publications, 2000.

[32] M. Abramowitz and I. A. Stegun, Eds., *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing*. Dover, 1972, ch. 4.5 Hyperbolic Functions, pp. 83–86.

[33] W. Cirne and F. Berman, "A model for moldable supercomputer jobs," in *IPDPS '01: Proceedings of the 15th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2001, p. 10059b.

[34] A. B. Downey, "A model for speedup of parallel programs," Computer Science Division, University of California, Berkeley, Technical Report UCB/CSD-97-933, Jan. 1997.

[35] E. Walker, "The real cost of a CPU hour," *Computer*, vol. 42, no. 4, pp. 35–41, Apr. 2009.

[36] "D-Grid Initiative," http://www.d-grid.de, [accessed: 2010-06-19].

[37] D. Sommerfeld and H. Richter, "A two-tier approach to efficient workflow scheduling in MediGRID," in *Grid-Technologie in Göttingen - Beiträge zum Grid-Ressourcen-Zentrum GoeGrid*, U. Schwardmann, Ed. Göttingen, Germany: GWDG, 2009, vol. 74, pp. 39–51. [Online]. Available: http://www.gwdg.de/forschung/publikationen/gwdg-berichte/gwdg-bericht-74.pdf

[38] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," *4th Intl. Workshop on Grid Computing*, p. 184, 2003.

[39] C. Schmidt and M. Parashar, "Flexible information discovery in decentralized distributed systems," in *HPDC '03: Proc. of the 12th IEEE Intl. Symp. on High Performance Distributed Computing*. IEEE Computer Society, 2003, p. 226.

[40] E. Tanin, A. Harwood, and H. Samet, "Using a distributed quadtree index in peer-to-peer networks," *The VLDB Journal*, vol. 16, no. 2, pp. 165–178, 2007.

[41] D. B. Jackson, Q. Snell, and M. J. Clement, *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Berlin/Heidelberg, Germany: Springer, 2001, vol. 2221/2001, ch. Core algorithms of the Maui scheduler, pp. 87–102.

[42] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A community authorization service for group collaboration," in *Proc. of the 3rd Intl. Workshop on Policies for Distributed Systems and Networks*, 2002, pp. 50–59.

[43] "Shibboleth," http://shibboleth.internet2.edu, Internet2 Middleware Initiative, [accessed: 2010-06-19].

Accelerating Cellular Automata Evolution on Graphics Processing Units

Luděk Žaloudek, Lukáš Sekanina, Václav Šimek
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
izaloude@fit.vutbr.cz, sekanina@fit.vutbr.cz, simekv@fit.vutbr.cz

***Abstract*—As design of cellular automata rules using conventional methods is a difficult task, evolutionary algorithms are often utilized in this area. However, in that case, high computational demands need to be met. This problem may be partially solved by parallelization. Since parallel supercomputers and server clusters are expensive and often overburdened, this paper proposes the evolution of cellular automata rules on small and inexpensive graphic processing units. The main objective of this paper is to demonstrate that evolution of cellular automata rules can be accelerated significantly using graphics processing units. Several methods of speeding-up the evolution of cellular automata rules are proposed, evaluated and compared, some with very good results. Also a comparison is made between mid-end and high-end graphics accelerator card based on the results of evolution speedup. The proposed methods are evaluated using two benchmark problems.**

Keywords–cellular automata; parallel computing; GPU; CUDA; genetic alghorithm

I. INTRODUCTION

The recent development of the SIMD-oriented general computation on Graphics Processing Units (GPUs) has motivated the research on new approaches to the acceleration of various computational models. Among others, accelerators of cellular automata (CA) and evolutionary algorithms (EA) have been proposed because of inherently parallel nature of these bio-inspired computing systems [1, 2, 6, 8].

Since their conception in 1950s [21], cellular automata have found many applications. These include physical systems modeling, road traffic simulation, random number generation, artificial life simulation, cracking of encryption standards [5, 7, 18, 20], etc. CA utilize several key features which make them unique computational models. Among these features are massive parallelism, locality of cell interactions, simplicity of basic building blocks and complex emergent behavior on a global level.

Because of inherent complexity, design of cellular automata is a difficult task for a human engineer. For example, Langton's self-replicating CA loops are based on identical cells; each of them has more than 280 transition rules [13]. In order to increase the efficiency when designing CA rules, evolutionary algorithms have been introduced to the field [14, 19]. By means of EA, the space of possible solutions to the problems of CA design may be explored efficiently. For example, Sipper has developed so-called "Cellular programming approach" [19] which allowed the CA rules to be evolved using a parallel cellular EA.

CA may be evolved either directly in hardware (such as FPGA) or in software, using simulators. This paper deals with the evolution of CA rules in a software CA simulator. However, design by EA is very computationally demanding. Not only is it necessary to simulate the CA which may consist of thousands of cells, but whole populations of CA have to be simulated and each CA may have many possible initial configurations, which need to be evaluated in order to determine the quality of a candidate solution.

One of possible ways to accelerate the CA simulation and, therefore, the execution of an EA is parallelization. Not surprisingly, there are some problems: Desktop CPUs with more than 6 cores are still not available (June 2010) and hi-end servers, supercomputers or computing clusters are highly expensive or overburdened if accessible. Since our interest lies with very large CA, we need a processing power capable of effectively accommodating hundreds of threads in order to justify the parallelization effort and the increased cost. However, with modern GPUs one can obtain computing power of supercomputers for a price of a hi-end PC.

The goal of the paper is to propose a GPU accelerator for evolutionary CA design. We will, in fact, propose and compare several architectures with the aim to identify the most efficient one. This paper extends our previous work [1] in the following aspects: (i) Models of CA and EA computation are presented in a greater detail. (ii) More experiments have been performed to evaluate the proposed architectures. (iii) We included another platform (9600 GT) for comparison. (iv) We have not investigated the speedup factor only; we have also measured the efficiency of the evolutionary algorithm using a simple benchmark problem.

The rest of the paper is organized as follows. Section II introduces one-dimensional cellular automata. Relevant evolutionary algorithms are to be briefly surveyed in Section III. Section IV describes the basic concept of general computing on GPUs. Section V proposes several methods how to utilize the parallel computing power of modern GPUs in CA rule evolution, whereas the benchmark problems – CA counter and majority – is defined in Section VI. Section VII describes in detail the experiments for the evaluation of the methods described in sections V and VI. Results of the conducted experiments are summarized in Section VIIII. While Section IX discusses obtained results, Section X

concludes this paper and proposes several possibilities of further development.

II. 1D CELLULAR AUTOMATON

A cellular automaton is an *n*-dimensional grid of identical cells, each working as a finite state automaton [3]. In its synchronous version, the state of the cells is periodically updated using a local transition function. If all the cells use the same local transition function, the automaton is known as uniform; otherwise, it is non-uniform. The next state of each cell is a function of its current state and the states of its neighboring cells. In case of 1D CA ($n = 1$), the neighborhood is defined using radius of r. In theory, the cellular automaton model supposes that the number of cells is infinite. However, in the case of practical applications the number of cells is finite. Then, it is necessary to define the boundary conditions, i.e. the setting of the boundary cells. Boundary conditions for the cells on the edges of the CA are usually either cyclic or constant (i.e., the states are taken from the opposite edge of CA or from the last cell). The state of the CA in the beginning of the run is called the initial configuration.

Experiments described in this paper deal only with 1D CA for simplicity and clarity purposes. The binary one-dimensional non-uniform CA of finite size may be described formally [17] as a 7-tuple $\mathbf{A} = (Q, N, R, z, b_1, b_2, c_0)$, where:

$Q = \{0, 1\}$ is a binary set of states,
N denotes a neighborhood ($N \subseteq \mathbf{Z}$),
z denotes the number of cells,
b_1 and b_2 are boundary values,
c_0 is an initial configuration, and
a mapping $R : S \rightarrow (Q^N \rightarrow Q)$ assigns to each cell of the grid $S = \{1, 2, \ldots, z\}$ a local transition function $\delta_1, \ldots, \delta_z$, where $\delta_i : Q^N \rightarrow Q, i \in S$.

A configuration of **A** is a mapping $c \in Q^S$ which assigns a state to each cell **A**. If only a single neighborhood $N = \{-1, 0, 1\}$ (i.e., $r = 1$) is considered, then the global transition function $G : Q^S \rightarrow Q^S$ is defined as:

$$G(c(i)) = \begin{cases} \delta_i(c(i-1), c(i), c(i+1)) & i = 2 \ldots z-1, \\ \delta_1(b_1, c(1), c(2)) & i = 1, \\ \delta_z(c(z-1), c(z), b_2) & i = z, \end{cases}$$

where c_i denotes the CA configuration in a step i. G is used to define a sequence of configurations $c_0, c_1, c_2, \ldots$ such that $c_j = G(c_{j-i})$, for $j \geq 1$. This sequence represents the *computation* of **A**.

Consider a uniform version of **A**, with the nearest neighbors neighborhood (i.e., $|N| = 3$) and cyclic boundary conditions. Each such cellular automaton is defined by a mapping $\{0, 1\}^N \rightarrow \{0, 1\}$ uniquely. Hence there are 2^8 such cellular automata, each of which is uniquely specified by the following (transition) rule

$000 \rightarrow a_0$
$001 \rightarrow a_1$
$010 \rightarrow a_2$
$011 \rightarrow a_3$
$100 \rightarrow a_4$
$101 \rightarrow a_5$
$110 \rightarrow a_6$
$111 \rightarrow a_7$.

We can speak of the cellular automaton with rule i, where i is an integer ($0 \leq i < 256$) with the binary representation $a_7a_6a_5a_4a_3a_2a_1a_0$. For example, Figure 1 shows the behavior of the cellular automaton with rule 150 that starts its computation from the initial configuration $c_0 = \ldots00100\ldots$ (the black square represents logic 1).

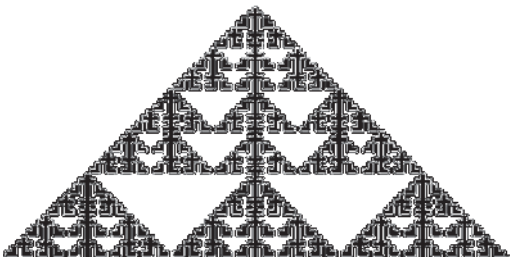


Figure 1. Development of 1D CA with rule 150.

The properties of cellular automata have been investigated by means of analytic as well as experimental methods. In general, the objectives are either to (i) find a method for the design of cellular automaton rules for a given application or (2) predict the global behavior of a given cellular automaton if the rules and the initial configuration are known. Because of the inherent complexity of cellular automaton, evolutionary design of CA rules has been adopted [19].

III. EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are stochastic search methods. They are inspired by Darwin's theory of biological evolution. Instead of working with one solution at a time (as random search, hill climbing and other search techniques do), these algorithms operate with the population of candidate solutions (candidate CA rules in our case). Every new population is formed by genetically inspired operators such as crossover (a part of CA rules is taken from one parent, the rest from another one) and mutation (inversion of some bits of the CA rule) and through a selection pressure, which guides the evolution towards better areas of the search space. The EAs receive this guidance by evaluating every candidate solution to define its fitness value. The fitness value calculated by the fitness function indicates how well the solution fulfills the problem objective.

The most common form of EA is a genetic algorithm (GA) which has the following form:

1. create randomly initialized population of individual solutions
2. evaluate the population – assign the fitness value to each individual
3. select the best individuals based on their fitness value
4. apply genetic operators (crossover and mutation) on the selected individuals and create a new population
5. if termination criteria are met (fitness, number of generations), finish, otherwise continue with 2

In our case, a candidate solution will be encoded as a finite size binary string composed of substrings that define the transition function for every cell.

It has been shown that EA may generate innovative results in many fields. However, the scalability of representation and scalability of fitness calculation were identified as major problems of the evolutionary approach [22]. In this work, the scalability problem is approached using parallelization of the CA rules evolution.

IV. NVIDIA GPUs AND CUDA

Although there are other universal computation-capable graphic accelerators with GPU programming interfaces such as ATI Stream from AMD, the most notable is nVIDIA with their Computer Unified Device Architecture (CUDA).

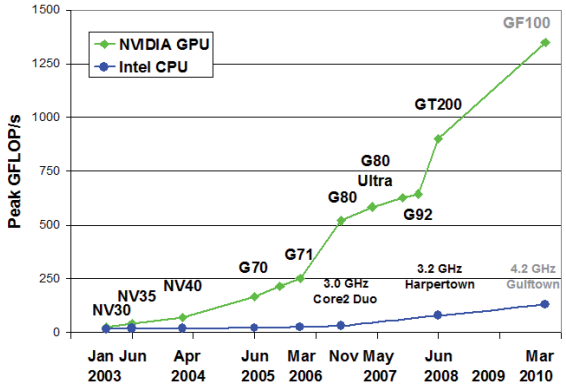


Figure 2. Performance of nVIDIA GPUs compared with Intel CPUs [15]. Supposed performance [16] is provided for the last chips as there are no official data yet (June 2010).

nVIDIA graphic accelerators contain GPUs with manycore streaming multiprocessors (MP) capable of outperforming general-purpose CPUs in some tasks (Figure 2). Each of these accelerators has from 4 to 30 multiprocessors with 8 scalar processor cores, two special units for transcendentals, a multithreaded instruction unit and on-chip shared memory (Figure 3). The multiprocessor creates, manages and executes concurrent threads in hardware with zero scheduling overhead [15].

Up until recently, direct programming for this hardware was not possible and indirect practices using OpenGL or DirectX had to be employed [11]. That changes with CUDA, which is a direct GPU programming interface.

CUDA works as a C language extension providing abstractions of thread groups, shared memories and barrier synchronization. This renders fine-grained data and thread level parallelism.

The code is separated into two classes: Host code which is executed on the CPU and device code which is executed on the GPU. Memory is differentiated in similar way, although it is possible to access device memory from host and vice versa trough the CUDA runtime library.

There are several types of device memory: Constant, shared, global, local and texture. Constant, texture and shared memory space is cached (4 clock cycles latency) but limited opposed to local and global memory (400 to 600 clock cycles latency) [15]. Effective usage of fast cached memory is key to high performance of the parallel application.

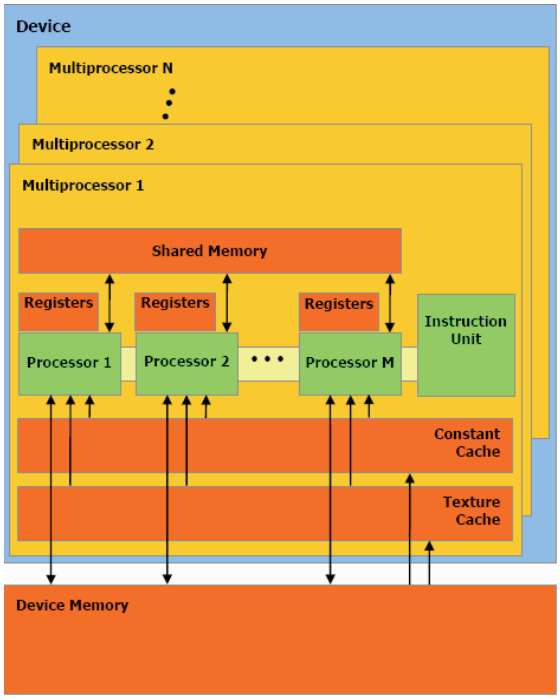


Figure 3. nVIDIA GPU structure [15]

Notable is the thread hierarchy used in CUDA programs: Threads may be arranged into blocks, where each block runs on one multiprocessor. It is possible to have more blocks than multiprocessors and more threads per block than cores. Shared memory may be accessible only within the block and thread synchronization is possible also only within the block. This is a possible drawback in some applications.

In recent years, CA have been implemented in GPUs, for example in [8]. As mentioned before, previous implementations of CA used Open GL or similar "shading" languages which brought several disadvantages: General purpose programming with Open GL or DirectX is overly complicated due to their specialization to computer graphics and also it does not enable direct control over the GPU's parameters, possibly rendering the computations ineffective.

V. PROPOSED METHODS OF PARALLELIZATION

In order to parallelize a GA, a computational complexity of its components must be considered. In the case of evolutionary design of CA rules, the evaluation of candidate rules (fitness) is surely the most demanding part. CA consisting of possibly thousands of cells must be simulated for a pre-specified number of simulation steps. Furthermore, many possible initial CA configurations have to be evaluated.

When determining the quality of CA, e.g. in the majority task [19] (a benchmark task for a 2-state CA which

determines whether the initial configuration contains more 1s then 0s by filling all the cells with the prevalent state after a number of steps), a 1D CA with 64 cells has 2^{64} possible initial configurations. This problem is dealt with by evaluating only several thousand randomly generated training vectors and measuring the success rate.

In GA, each generation has a population of possibly hundreds of individuals which further multiplies the number of calls to the fitness function.

Three possible approaches to parallelization will be proposed in following subsections.

A. *The Level of Cells*

First approach executes parallel lookup of transition rules in cells. There are as many threads as there are cells within the CA. The cell states are kept in the shared memory so the threads have to be synchronized after each step in order to guarantee the proper sequence of simulation steps.

The algorithm follows:

```
/*host part*/
Generate the initial CA configuration
Load the configuration into device
shared memory;
Load the CA transition rules into
device shared memory;

/*device part*/
For each thread do
  Repeat for S steps
    Compute transition function for
    one cell and update the cell;
    Synchronize the shared memory;

/*host part*/
Load the final CA configuration into
host memory;
```

This limits the algorithm only to one MP due to lack of synchronization between blocks. Graphical representation of the algorithm is shown in Figure 4.

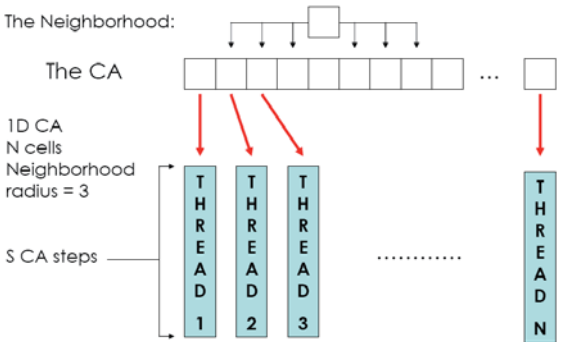


Figure 4. Simulation of CA with parallelization on the level of cells. Here, *S* denotes the number of CA simulation steps

B. *The Level of Training Vectors*

Second approach utilizes parallelization on the level of training vectors. There are as many threads as there are training vectors. Because there is no dependency between two same automata running two different simulations, it is possible to use more parallel blocks (i.e. multiprocessors) than one.

The algorithm ensues:

```
/*host part*/
Generate V initial configurations;
Load the CA configurations into device
global memory;
Load the CA transition rules into
device shared memory;

/*device part*/
For each thread do
  Load one CA configuration into
  registers/local memory;
  Simulate the CA for S steps;
  Calculate the fitness function;
  Update the fitness result into
  device global memory;

/*host part*/
Load the fitness results from device
global memory into host memory;
Calculate fitness for the individual;
```

Graphical representation of the main part of the algorithm is depicted in Figure 5.

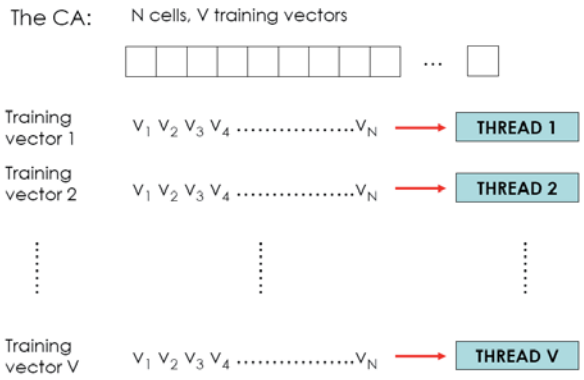


Figure 5. Parallelization of CA simulation on the level of training vectors. Here *V* denotes the number of training vectors per individual CA and *S* denotes the number of CA simulation steps

C. *The Level of Individual Solutions*

The last approach is to evaluate one individual per thread. There are as many threads as there are individuals within the GA population.

However, this approach is the most memory consuming, because we need to hold not only multiple CA configurations but also multiple transition rules.

The algorithm follows:

```
/*host part*/
Load the CA transition rules into
device global memory;

/*device part*/
For each thread do
  Load the CA transition rules into
  local memory;
  Generate training vectors;
  For each training vector do
    Simulate the CA for S steps;
    Update the fitness into device
    global shared memory;

/*host part*/
Load the fitness results from device
global memory into host memory;
```

Graphical representation of the algorithm is shown in Figure 6.

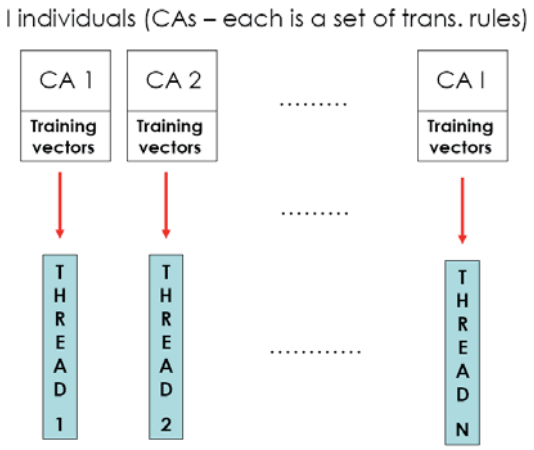


Figure 6. Parallelization of CA simulation on the level of individuals. I denotes the number of individuals and threads.

D. Problems Without Training Vectors

Previous subsections proposed parallelization approaches for problems which require evaluation with sets of training vectors. However, there are also problems which do not require such measures because the initial configuration of the CA is known. Such problems include applications as counters or random number generators, which were not mentioned in [1].

VI. Benchmark Problems

A. 4-bit Counter

A counter is a device (in this case implemented by means of CA), which is able to generate a certain sequence of numbers or in this case a certain sequence of CA configurations. For example a 4-bit counter seeded by the value of 5 has to generate a sequence $seq^{(5)}$ = 5-6-7-8-9-10-11-12-13-14-15-0-1-2-3-4. The sequence is encoded in four cells, each with 2 possible states (0, 1).

The goal will be to evolve a simple 1D non-uniform CA which will generate the desired sequence. The CA will have a simple neighborhood with radius r = 1. The non-uniform CA was selected because non-uniform cellular automata enable us to perform more complex tasks than with uniform CA of the same number of cells [19]. Generating a certain sequence is not a simple task in this context.

Since training vectors aren't used, the proposed approach will be different from the scenarios assumed in Section V. The evaluation of the candidate solution works in the following way: The CA is simulated for 16 steps (because we have 16 numbers in the sequence) and in each step, the configuration is compared with a desired number. So for example, the second step configuration is compared with the value 6, the third step configuration with the value 7 etc. In the end, all 16 configurations should correspond with the 16 desired numbers. Each match is awarded with a point to the fitness value, so the maximum fitness is 16 and minimum is 1 (the initial configuration is counted automatically).

It is needed to keep as many rules as there are cells, because a non-uniform CA is used. A non-uniform CA does not need to have necessarily the same number of rules as the number of cells (some rules may apply for more than one cell) but in this case, the encoding is simple and keeping references to rules and interpreting them may prove unnecessary and too complex. Also there are no training vectors, so it is not possible to parallelize a single individual with the same set of transition rules which are placed in the shared memory within the same block. So either the cell parallelization approach has to be used (Section V.A) or an approach similar to the parallelization on the level of individuals mentioned in Section V.C. The latter approach seems more promising, so we will use it. Fitting more transition rules into the GPU memory several times could prove to be challenging but the CA has a simple neighborhood and 2 possible states, so in this case, it is not a serious problem.

B. Majority

The second benchmark problem is the majority task defined in Section V. A 2-state CA computing the majority task has to determine whether the initial configuration contains more 1s then 0s by filling all the cells with the prevalent state after a number of steps. Usually, the number of steps equals double the number of cells in the automaton and the quality of the solution is determined by randomly generating certain number of training vectors and measuring the proportion of successful CA runs [19].

VII. The Experiments

A. The Evolution of CA Rules for the Majority Problem

Several experiments were conducted in order to evaluate the speedup of proposed parallelization methods. Two

different computers were used: (i) A laptop with Intel Core 2 Duo processor at 1.83 GHz with 667 MHz FSB and low-end nVIDIA GeForce 8600M GS graphic accelerator with 4 multiprocessors; (ii) A workstation with Intel Core 2 Duo processor at 3.33 GHz with 1333 MHz FSB and hi-end nVIDIA GeForce FX 285 (iia) graphic accelerator with 30 multiprocessors or a mid-end nVIDIA GeForce 9600 GT (iib) graphic accelerator with 8 multiprocessors, however (iib) was used only in later experiments described in Subsection B.

The programs were compiled with MS Visual C++ 9.0 compiler (serial versions) and with CUDA 2.3 SDK (parallel versions).

Each parallelization approach was tested on both computers and the execution times were compared with those of serial versions of the programs. This means the serial versions were run on a single CPU core without the use of GPU. The whole program execution time was measured.

The 1D 2-state uniform CA with 64 cells, r=3 (7 cell neighborhood) and cyclic edge conditions was used for the experiments. Each simulation lasted 128 synchronization steps. The CA rule is represented as string of 128 integers.

First approach (Section V.A) was tested both only on a number of CA simulations without the GA (and without fitness evaluation!) and with the GA, which was a standard algorithm with 10 generations, crossover rate of 70% and mutation rate of 1%. The crossover was one-point and the mutation probability is meant for a single gene (i.e. bit). Two-step tournament selection was used. Population size was 100 and 240 training vectors were used. Note that 10 generations are not sufficient to find a good solution, however we are interested in the speedup analysis only.

The rest of the experiments (based on Section V.B and V.C) were executed with the same GA. The size of the population and the number of training vectors varied for the last experiments (240 vectors and 100 individuals for the training vector approach and vice versa) but the number of fitness evaluations was proportionally the same. The number 240 was conveniently selected because the number of processing cores within the GTX 285 accelerator.

Additionally, several more experiments were conducted focusing on the scalability of the proposed algorithms. The same parameters were used except for the number of training vectors or individuals within the population. Sizes of the problems were increased up to hundredfold relative to the original problem sizes.

The fitness function of the GA consists of V simulation runs of the individual CA each for S simulation steps, where V is the number of training vectors and S is equivalent to double the width of the CA. At the end of each simulation run, the success of the run is evaluated based on the majority task (selected for demonstration purposes): All cells of the CA should be in the state prevalent at the initial configuration.

B. Further Experiments With Majority

Based on the results of previous experiments (see Section V), the best approach was selected and more detailed results were obtained. The focus of these experiments was to determine best block-size setting policy and to make a more detailed comparison between two accelerators on the same machine (iia and iib).

Different problem sizes (namely 240, 480, 1200, 2400, 4800. 9600 and 24000 training vectors) and block sizes were evaluated (8, 10, 15, 20, 40, 80, 120, 160, 240 and 320 threads per block). Not all the results could be obtained for some tasks because the number of threads per block must be an integer.

C. The Evolution of Binary Counters

The other proposed approach mentioned in Section VI was also evaluated. The 4-bit counter design was selected as a benchmark since is not complicated in terms of search space, so the best solution is well known due to experiments with brute force search [17]. The problem has best solution with fitness 10 which means that the CA can approximate only 10 numbers from the sequence.

The initial configuration of 5 was chosen deliberately, because the CA has the best results with this. For example, the $seq^{(0)}$ has best fitness of 8 and $seq^{(2)}$ has best fitness of 9. Best result for $seq^{(5)}$ generates the sequence 5-**2**-7-**4**-9-**14**-11-12-13-**6**-15-0-1-2-**7**-4. The incorrect numbers are in bold [16].

The experiment was designed as follows: The CA was 1D non-uniform 4-cell with r=1 (3-cell neighborhood) and static boundary conditions. Each CA simulation lasted 16 steps.

The evolution of the desired CA was performed with the standard GA. The population was set to 32 individuals, crossover rate of 70% and mutation rate of 18%. The crossover was one-point and the mutation probability is meant for a single gene (i.e. bit). Two-step tournament selection was used. The GA parameters match the experiment conducted in [17] to maintain the best possible quality of solutions and comparability to previous result.

The termination was set to the achievement of the maximum fitness and several hundred runs were conducted with the serial version and several dozen with the parallel version. The machine for the parallel version was the same workstation as in previous experiments (ii), fitted either with GeForce 9600 GT or with GeForce GTX 285. The serial experiments were conducted on several dozen of blade servers comparable with the workstation (Intel Xeon 2.8-3.2 GHz processors with 1 GHz FSB). The goal was to measure the average generation needed to find the best solution (fitness 10). Also, the time was measured for the serial and for the parallel versions.

VIII. RESULTS

The objective of this paper was to evaluate the performance of several parallel algorithms incorporated into an EA. The majority problem was selected only to conveniently pose as the EA's goal in the first series of experiments and so we are not interested in the quality of evolved solutions. The only relevant information is the achieved speedup compared to serial implementations.

The results for the laptop are shown in Table I while the results for the workstation are shown in Table II. The values in the tables are averages of 10 runs.

Fitness evaluation means the simulation of one CA with one training vector (e.g. 100 individuals with 240 training vectors each and over 10 generations means 24000 fitness evaluations).

TABLE I. RESULTS FOR LAPTOP WITH 8600M GS

Approach	Fitness evaluations	Serial time	Parallel time	Speedup
Simulation of CA only	50000	347.39	0.56	621.68
Parallelization of GA				
CA cells	240000	1787.23	2597.60	0.69
Training vectors	240000	1787.74	251.20	**7.12**
Individuals	240000	1784.26	821.04	2.17

TABLE II. RESULTS FOR WORKSTATION WITH GTX 285

Approach	Fitness evaluations	Serial time	Parallel time	Speedup
Simulation of CA only	50000	187.33	0.38	489.75
Parallelization of GA				
CA cells	240000	981.34	1597.60	0.61
Training vectors	240000	980.38	72.18	**13.58**
Individuals	240000	980.95	105.21	9.32

The scaling properties of the three proposed algorithms are summarized in Table III. Only the best results from both testing computers are shown. The result for 24000k fitness evaluations with the cell level parallelization is not shown because the computation did not terminate before 10 hours reserved for parallel computation.

TABLE III. SCALING PROPERTIES OF PROPOSED PARALLEL ALGORITHMS

Fitness evaluations	240k	2400k	24000k
Approach	Speedup		
CA cells	0.69	0.69	N/A
Training vectors	14.36	127.16	**417.36**
Individuals	9.32	28.41	192.88

After this, several extremely long runs (more than 240k fitness evaluations) were computed with the training vector approach and the best results approached a speedup of 420.

The other series of experiments shows results of comparison between serial version, workstation with 9600 GT (iib) and workstation with GTX 285 (iia). Note that the original contribution [1] included result with GTX 280. In this article the accelerator was replaced with its more modern version GTX 285. The difference between GTX 280 and 285 is in higher core and memory operating frequencies, other characteristics remain the same. More details on the accelerators may be found in [4].

As mentioned in Section V.B, several variations of the task were evaluated. Table IV. shows the best results for different problem sizes (number of training vectors) and different thread-per-block setup. The results are average speedup over several runs with respect to serial run described in Section VII. The best results for each accelerator are highlighted.

TABLE IV. SPEEDUP FOR 9600 GT AND GTX 285

Block size	Problem size [training vectors]						
	240	600	1200	2400	4800	9600	24000
	9600 GT						
8	13.95	26.14	22.66	26.72	26.77	28.09	28.39
10	14.14	27.08	33.02	33.28	33.29	35.29	34.99
15	**14.36**	**27.86**	34.30	44.65	52.33	52.51	52.64
20	13.99	27.42	59.38	61.26	61.18	61.42	64.13
40	13.97	27.27	62.54	92.90	97.37	98.59	97.96
80	13.84	27.50	**64.47**	106.06	108.02	110.00	109.64
120	13.65	27.09	60.92	97.18	98.68	118.61	121.03
160	N/A	26.71	N/A	**108.56**	101.22	**120.32**	**126.34**
240	12.69	25.15	61.00	83.97	98.44	123.19	121.02
320	N/A	N/A	N/A	N/A	**109.68**	110.49	111.10
	GTX 285						
8	**13.58**	27.00	64.21	64.00	83.28	97.20	94.61
10	13.57	27.02	65.27	117.72	118.85	122.58	120.79
15	13.58	**27.04**	**66.08**	126.68	124.73	161.32	170.53
20	13.22	26.68	65.27	125.01	215.24	218.03	218.67
40	13.25	26.49	65.45	125.99	222.77	286.31	272.57
80	13.14	26.32	64.82	**127.16**	232.85	350.96	329.73
120	12.96	25.92	64.21	125.87	222.62	351.56	387.20
160	N/A	25.55	N/A	124.10	**236.04**	**370.48**	**417.36**
240	12.08	24.10	59.75	117.43	224.61	302.47	387.37
320	N/A	N/A	N/A	N/A	208.42	371.04	377.95

Figure 7 shows best results for different accelerators and problem sizes only for the optimal block size settings. The results are average execution time obtained from several runs. Figure 7 indicates the difference in performance growing with the problem size.

The last series of experiments measured the speed of evolution designing the solution for the 4-bit counter problem.

The average generation needed to achieve the maximum fitness was 14509 and the average time to achieve it was 5.25 seconds.

With the 9600 GT accelerator (iib), the average generation was 14591 and the time needed 8.5 seconds.

For the GTX 285 accelerator (iia), the average results were 14374 generations and 8.77 seconds.

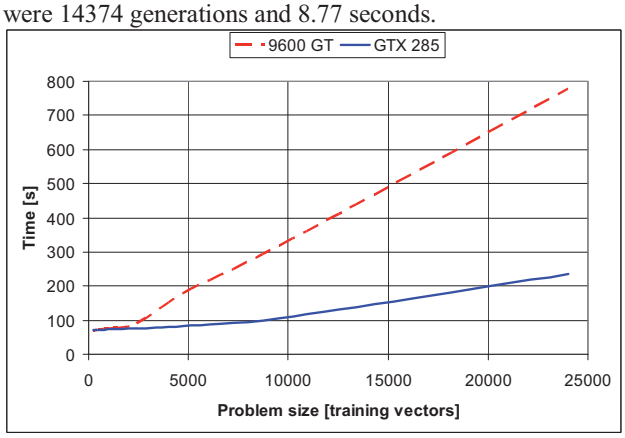


Figure 7. Execution time for different problem sizes and best block size settings for GeForce 9600 GT and GTX 280

It can be seen that no actual improvement of the GA was achieved. However, this was to be anticipated. Moreover, not even a speedup of computation was achieved but on the contrary, the result was slower than serial computation. The reason for this is too small population of the used GA. The overhead required to start the parallel computation on the GPU is larger than the speedup. The result is summarized in Table V.

TABLE V. SPEEDUP OF EVOLUTION WITH POPULATION OF 32

GPU	Avg. generation	Time [s]	Speedup
Serial	14509	5.25	N/A
9600 GT	14591	8.50	0.62
GTX 285	14374	8.77	0.60

In order to prove that accelerating the GA has any sense, another experiment was conducted. This time the population was increased tenfold to 320 individuals. An unsolvable fitness condition was set and the computation was terminated at generation 15000. This had to be done in order to actually measure something, because otherwise the GA would end too soon. This time the experiment proved the conclusion of the previous one: The new problem was large enough, that the parallelization could pay off with a speedup of 38.66 for (iib) and 38.11 for (iia). Results may be found in Table VI.

TABLE VI. SPEEDUP OF EVOLUTION WITH POPULATION OF 320

GPU	Avg. generation	Time [s]	Speedup
Serial	15000	54.12	N/A
9600 GT	15000	1.40	38.66
GTX 285	15000	1.42	38.11

IX. DISCUSSION

A. *Speedup vs. Cost*

As can be seen in Tables I and II, the parallelization of the CA simulation on the level of CA cells shows massive speedup. More surprisingly, those results were obtained with only one multiprocessor in the GPU due to the inter-block synchronization problem mentioned in Section IV. The possible explanation is the effective use of the device shared memory which is much faster than ordinary memory. The results from the laptop GPU are better, because they were compared with much slower processor in the laptop opposed to the hi-end processor in the workstation.

However, when the simulator was inserted into a GA, the speedup declined due to more memory accesses and fitness evaluation. The one-block approach utilizing only one multiprocessor shows its weaknesses and the overall result is even worse than the serial approach.

The results for the parallel GA (the approach from Section V.B) with threads executing individual training vectors are the best of the experiments. The maximum speedup for the workstation is 417.36 and the speedup for the laptop is only 31.34 for the largest problem. As opposed to the cell parallelization approach from Section V.A, this algorithm has to upload large quantities of data to and from the device memory but it has more processor cores and the data don't conflict with each other.

As seen in Table III, there was no speedup drop with larger problems (2400 and 24000 training vectors) and the performance was even better than for the smaller problem (240 training vectors).

The individual-per-thread approach (Section V.C) showed smaller speedup than training vector-per-thread approach. The lower performance is probably caused by more memory transfers (several sets of CA rules per block opposed to only one set of CA rules per block).

There are also some problems with graphic accelerator cards used as primary display adapters due to graphic driver timeouts caused by long thread execution times so this approach may not be suitable for this reason. This problem may be solved by using second graphics adapter as the primary display adapter at the expense of increased cost in hardware.

The scaling capabilities of the last approach are also good as seen in Table III. The conclusion for the experiments is that parallelization on the level of evaluation of training vectors is the most effective due to utilization of all multiprocessors in the GPU and quick and small parallel kernel (code for the GPU - as opposed to parallelization on the level of individuals where the kernel lasts longer).

The comparison of graphic accelerators showed partly interesting results. A GeForce 9600 GT with 8 multiprocessors showed a nice speedup not falling so far behind the GTX 285 with 30 multiprocessors in some cases. The mid-end card was even faster in some of the smallest tasks. This could be influenced by slightly larger scheduling overhead with the more complex GPU. This opposes the manufacturer's claim about the zero scheduling overhead [15]. No other explanation seems to fit and since precise

details about GPU hardware are obfuscated by the manufacturers, we may only guess the real reason.

Of course, when the problem size increased sufficiently, GTX 285 manifested its higher number of processors which could be used to full capacity and the speedup was significantly higher. The interesting part is, that 9600 GT costs about 80 Euro and the GTX 285 about 370 Euro (Jan 2010, retail price for the Czech Republic). The low-end card shows no promise for scientific computation for two main reasons: It is mounted in a laptop and the memory bus is too slow.

B. *Other Tasks for GPU*

It seems that the mid-end card is the most cost-effective for small and medium-sized tasks. However, the results may be different with other parallelization tasks. It is important to note the two laws for parallel computation. First is the Amdahl's law [12] which states:

$$S(P) = P / (1 + \alpha\ (P - 1)),$$

where S(P) is the speedup, P is the number of processors and α is the sequential part of the task. The equation clearly shows that a fixed task speedup is severely limited by the sequential part, no matter how many processors are used.

The second law is attributed to Gustafson [12]:

$$S(P) = P - \alpha\ (P - 1)$$

and it may be applied to a class of problems, where the non - sequential part of the task is limited by the number of processors and when a new processor is added, we may assign it the same amount of work as to the other processors.

This class of problems includes evolutionary design of CA rules which need to be evaluated by high number of training vectors. More training vectors means more accurate results and more fine-grained fitness function which may contribute to better evolution results.

There is also another class of evolutionary design problems which is limited by the first law. This class includes CA design, which needs to evaluate a small fixed number of steps or possibilities and is not suitable for parallelization. These problems may be also limited by the number of individuals within the GA population, as was the case with the 4-bit counter evolution, where the best population size is 32 [17]. Some of these problems may drift to the area where the task size could be expanded and so Gustafson's law may be applied. Thankfully, many CA design tasks by means of EA fall in this category like our example with the majority benchmark. The 4-bit counter is only a simple demonstration and evolution of larger non-uniform CA will probably need larger populations.

The last conclusion obtained from the results is the fact that block size settings greatly impacts the GPU's performance. Table IV. shows that the optimal settings changes with the size of the problem. The reason for the need to tailor the settings for a certain task is the way the GPU dispatches threads within the blocks.

The GPU programming model uses several levels of data- and thread-parallelism which enable the simultaneous execution of data-independent threads [15]. This concept works in concert with the hardware which offers scheduling consistent with multi-threading concept. This enables the threads within one block to be scheduled as the need arises.

The threads are executed on groups called "warps" which are 16 threads wide and which work in a SIMD concept. That means that each thread within the warp has to execute the same instruction, of course on different data. If the blocks are not large enough, significant parts of the warp may be wasted. Also, the warps are scheduled depending on their availability. If one warp waits for a memory access, several more warps may be executed, even from different blocks. Figure 8 illustrates simultaneous execution of warps on three different multiprocessors.

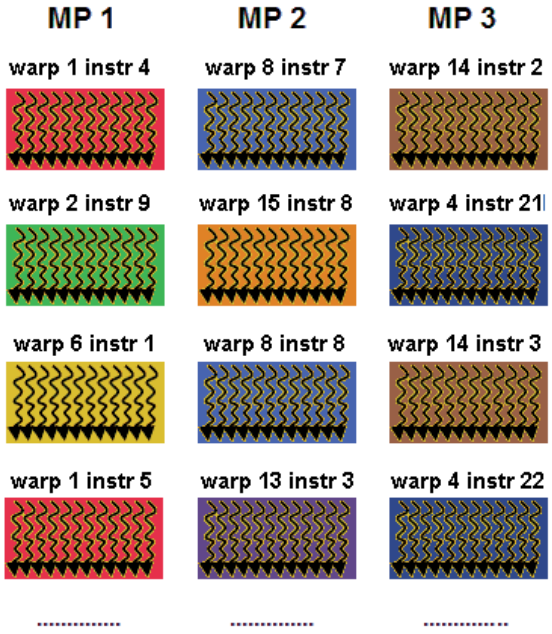


Figure 8. Warps executing on three different multiprocessors (MP): Each warp contains 16 threads and each is executing the same instruction. Warps are scheduled according to readiness for execution.

X. CONCLUSION

The experiments have shown that evolution on GPUs has several limitations. The most significant one is the fact, that the amount of device shared memory and registers is limited thus restricting the size of evolved CA, the number of training vectors or the size of the EA population.

Dealing with this problem may be the objective of further research and development. Possible solutions include partitioning the computation and serializing the parts in order to save memory. Another solution may be using device global memory instead of shared memory and local memory instead of registers. However, both of these approaches

would result in slower performance. Future accelerators may possess larger shared memory [16].

Another possibility for development is testing combinations of the three proposed approaches. E.g. it would be possible to evaluate several individuals of the GA population each in one block while running parallel lookup of cell transition rules in that block.

Further improvement of methods mentioned in this paper could lead into fast parallel version of Sipper's Cellular programming approach [19].

Cellular programming is a methodology developed to design non-uniform CA systems capable of computing complex tasks such as synchronization, majority or sorting. Speeding up the evolution of CA systems may prove to be appropriate step in perfecting such systems via simulation and implementing them in hardware.

The most recent language contribution to the field of GPGPU (General Purpose GPU computing) is OpenCL (Open Computing Language). OpenCL is a C language extension similar in use to CUDA but there is one significant improvement: It supports many different GPU or CPU architectures, being almost universal. OpenCL uses abstractions independent of manufacturers which can be automatically transformed into efficient code for any supported architecture by means of compilation. With Open CL, it is possible to run the same program on nVIDIA and ATI. The language and its tools started to become available only recently, so this article deals only with CUDA. Future plans include transferring current and prospective work to this new language.

Generally the future of GPGPU looks bright. Manufacturers like nVIDIA and ATI compete with each other in GPU performance pushing the development further. Right now more advanced GPU are being planned and developed. Example of this new generation may be nVIDIA Fermi which will include 512 cores (64 MPs) and such technologies as simultaneous kernel execution, shared cache memory for the entire GPU or ECC (error checking and correction) [16].

Moreover with the introduction of OpenCL the programming of competitor's hardware will be unified, further improving the programmer's experience. The architecture is also designed as scalable from the start enabling to connect several GPUs together or to migrate old programs to newer versions just by adjusting the problem size and block settings. More cores and higher operating frequency means more computing power for problems which can be enlarged in accordance with Gustafson's law.

ACKNOWLEDGMENT

This work was partially supported by the grant **Natural Computing on Unconventional Platforms** GP103/10/1517, the FIT grant FIT-10-S-1 and the research plan **Security-Oriented Research in Information Technology**, MSM0021630528.

REFERENCES

[1] Žaloudek, L., Sekanina, L., Šimek, V.: "GPU Accelerators for Evolvable Cellular Automata", Computation World: Future Computing, Service Computation, Adaptive, Content, Cognitive, Patterns, Athens, GR, IEEE, 2009, pp. 533-537.

[2] Chitty, D.M.: "A data parallel approach to genetic programming using programmable graphics hardware", GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, Volume 2., London, ACM Press, 2007, pp. 1566–1573.

[3] Codd, E., Cellular Automata, Academic Press, 1968.

[4] CUDA-Enabled GPU products - NVIDIA URL: <http://www.nvidia.com/object/cuda_learn_products.html> [cit. 29.1.2010]

[5] Durbeck, L. and Macias, N., "The Cell Matrix: An Architecture for Nanocomputing", Nanotechnology 12, IOP Publishing 2001, pp. 217–230.

[6] Fok, K.L., Wong, T.T., Wong, M.L.: "Evolutionary computing on consumer graphics hardware", IEEE Intelligent Systems, Vol. 22, No. 2, IEEE, 2007, pp. 69–78.

[7] Gardner, M., "Mathematical games: The fantastic combinations of John Conway's new solitaire game 'Life'", Scientific American 223, Oct 1970, pp. 120-123.

[8] Gobron, S., Devillard F., Heit B., "Retina simulation using cellular automaton and GPU programming", Machine Vision and Applications Journal 66, Springer, 2007, pp. 331–342.

[9] Harding, S.: "Evolution of Image Filters on Graphics Processor Units Using Cartesian Genetic Programming", 2008 IEEE World Congress on Computational Intelligence, Hong Kong: IEEE CIS, 2008, pp. 1921–1928.

[10] Harding, S. and Banzhaf, W.: "Fast genetic programming on GPUs", Proceedings of the 10th European Conference on Genetic Programming, LNCS 4445, Springer, 2007, pp. 90–101.

[11] Harris, M.: "Mapping computational concepts to GPUs", ACM SIGGRAPH 2005, ACM, New York, NY, 2005.

[12] Henessy, J. and Patterson, D.:Computer Architecture A Quantitative Approach, The Morgan Kaufmann Series in Computer Architecture and Design, Morgan Kaufmann Publishers, 2003.

[13] Langton, C.G., "Self-Reproduction in Cellular Automata", Physica D: Nonlinear Phenomena 10(1-2), Elsevier, 1984, pp. 135-144.

[14] Lohn, J.D., and Reggia, J.A., "Automatic discovery of self-replicating structures in cellular automata", IEEE Transactions on Evolutionary Computation, vol.1, no. 3, IEEE CS, 1997, pp. 165-18.

[15] nVIDA CUDA Programming Guide, Version 3.0 URL: <http://developer.nvidia.com/object/ cuda_3_0_downloads.html.> [cit. 14.6.2010]

[16] Next Generation CUDA Architecture, Code Named Fermi URL: <http://www.nvidia.com/object/fermi_architecture.html> [cit. 31.1.2010]

[17] Sekanina, L.: Evolvable Components: From Theory to Hardware Implementations, Natural Computing Series, Springer-Verlag, Berlin Heidelberg, DE, 2004.

[18] Šimek, V., Dvořák, R., Zbořil, F., V., Kunovský, J. "Towards Accelerated Computation of Atmospheric Equations using CUDA", Proceedings of Eleventh International Conference on Computer Modelling and Simulation, Cambridge, GB, IEEE CS, 2009, pp. 449-454.

[19] Sipper, M., Evolution of Parallel Cellular Machines: The Cellular Programming Approach, Springer Verlag, Heidelberg, 1997.

[20] Tomassini, M., Sipper, M., Perrenoud, M., "On the generation of high-quality random numbers by two-dimensional cellular automata," Computers, IEEE Transactions on , vol.49, no.10, Oct 2000, pp.1146-1151.

[21] Wolfram, S.: A New Kind of Science, Wolfram Media Inc., Champaign, IL, 2002.

[22] Yao, X. and Higuchi, T., "Promises and Challenges of Evolvable Hardware", IEEE Transactions on Systems, Man, and Cybernetics 29(1), IEEE, 1999, pp. 87–97.

Service-Oriented Integration Using a Model-Driven Approach

Philip Hoyer, Michael Gebhart, Ingo Pansa, Aleksander Dikanski, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology
Zirkel 2, 76131 Karlsruhe, Germany
{ hoyer | gebhart | pansa | a.dikanski | abeck } @ kit.edu

Abstract — **Provision of processes supported by Information Technology (IT) spreading around several different units of one organization requires the integration of existing distributed legacy applications. Typically the part of the application's functionality used in a process is offered through proprietary interfaces, complicating the integration. A possible solution to this issue is to construct standards-based, service-oriented interfaces offering only the required functionality. Existing approaches within this field mostly focus on the technical issues of the integration using Web services and hardly consider the integration from the perspective of the IT-supported processes. In this article, we introduce a development approach for modeling an IT-supported process which is enhanced by the automatic generation of necessary Web service artifacts. Our approach is exemplified by a scenario at the Karlsruhe Institute of Technology (KIT) that implements a process to visualize the study progress of a student.**

Keywords—model-driven development; service-oriented integration; Web services; Unified Modeling Language

I. INTRODUCTION

Due to fast changing markets and emerging requirements, an organization's Information Technology (IT) needs to be flexible in order to quickly provide new functionality. Usually this flexibility is required at the high level of rapidly changing business processes, which necessitates the implementation of IT-supported business processes. Several applications already exist and are in practical use within today's organizations, providing basic functionality and data. Often these existing legacy applications can hardly be replaced or enhanced with new functionality due to high costs or the associated high complexity. If new functionality has to be added, it should reuse existing functionality in order to reduce costs. Thus, the realization of new functionality requires the integration of these existing applications. Using existing applications in an integration scenario is complicated by the proprietary interfaces these applications provide. Additionally not all of the functionality of an application might be used in a new IT-supported process. To overcome these issues, a service-oriented architecture (SOA) is a widely accepted approach for process-oriented integration scenarios, but still the development of standardized interfaces is carried out by hand, leading to high development costs as well as to long and error-prone development cycles.

In a typical top-down integration approach, in which the needed functionality of the legacy applications is determined by the IT-supported process to be implemented, the process has to be formally modeled beforehand. It describes the flow of actions that have to be performed for the new functionality. Each action represents a functionality provided by one of the existing applications. The workflow that integrates the existing applications can be formalized using Activity Diagrams of the Unified Modeling Language (UML) [2]. In a next step, the existing applications have to be made accessible to reuse existing functionality. For this purpose, adapters have to be developed to perform the integration on a technical level. They enable the access to existing functionality in a convenient way. Figure 1 illustrates this approach of developing new functionality by integrating existing functionality. As technology to realize the adapters, Web services can be used as they represent a standardized technology that is platform and programming language neutral. Web services are de facto standard in the context of service-oriented architectures. To provide the functionality using Web services, the interface description and the data schemas have to be developed using the Web Service Description Language (WSDL) [3] and XML Schema (XSD) [4]. Providing the functionality as a Web service enables a convenient usage of it within different contexts. To decouple the Web service from the existing applications, it is desirable to abstract from existing data types within the existing application and to create the data schema used within the Web service from a more conceptual view.

Existing integration approaches mostly focus on solutions on technical levels. That means that they focus on the development of Web service adapters including technical aspects such as the usage of various Web service standards. The conceptual issues such as the modeling of an IT-supported process that can be effectively realized are subject of further investigation. This requires the analysis of the existing applications to avoid unnecessary data transformations when realizing the workflow that integrates the existing applications. For example, the data types available within the existing application can provide an indication about how to design an appropriate IT-supported process.

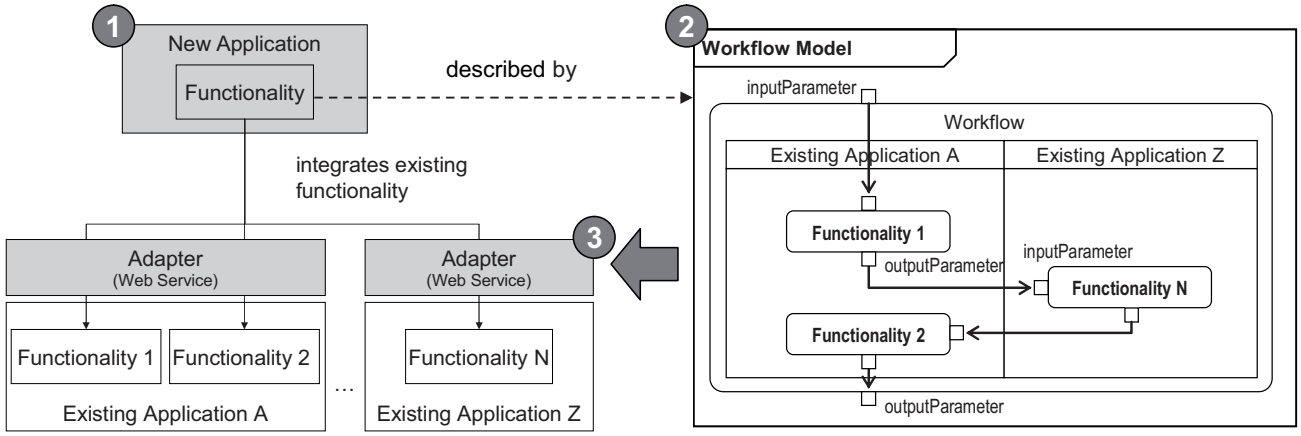


Figure 1. Integration of Existing Functionality

In this article we propose a development approach that supports the developer in creating the IT-supported process and helps to speed up the workflow that integrates the existing applications by automatically generating Web service adapters for existing applications and a Web service for the newly developed functionality. The presented approach in this paper extends our model-driven development method described in [1]. To get a better understanding of the IT-supported process, we start with mock-ups and sketches of graphical user interfaces (GUI). They enable the identification of required functionality. As a complement a domain model is created, allowing the derivation of conceptual data types that are independent of existing applications. Afterwards, an analysis of existing applications is performed that provide the required functionality identified with the help of the GUI sketches. This analysis enables the creation of the process with a reduced set of data transformations. The process is formalized using Activity Diagrams of the Unified Modeling Language (UML). In a next step, the Web services and data schemas are automatically derived from the formalized process and the domain model using an enhanced version of our model-driven development method [1]. The enhancement lies in the usage of the domain model which enables the decoupling of the Web service providing the new functionality from existing applications by creating data types that are independent from existing applications.

The derivation of the Web service artifacts is a two step process, starting with a platform-independent model representing the service interfaces and data schemas on a conceptual level and the final platform-specific realization using Web services. The approach targets a service-oriented integration in which functionality is provided as a service. The approach itself is model-driven, which means that the required Web service interface descriptions using WSDL and the data schemas using XML are generated automatically from the workflow that integrates the existing applications and the domain model. To realize our approach, existing work in the context of developing Web service adapters is reused. Also, existing guidelines how to derive data schemas from a domain model are applied.

Our approach is exemplified by a university scenario at the Karlsruhe Institute of Technology (KIT). In this scenario, the goal is to provide a new feature that allows students to gain insight into their current study progress, by combining and visualizing their data from disparate sources. The required functionality and data is shared across several existing applications, so that an integration workflow is necessary, which accesses these applications and combines the collected data. In a first step, the requirements are analyzed. The domain model for the study progress scenario and several GUI sketches are created. Based on these requirements, the required applications are identified and the process is created and formalized. Afterwards, the process and the domain model are transformed into a description of the required services. In a last step, the necessary Web service interfaces using WSDL and data schemas using XML Schema are created.

The rest of the article is structured as follows: Section 2 introduces the background and gives general information about integration issues and their solutions. In Section 3 the concept of our approach of a service-oriented integration using a model-driven approach is described. The practical implications and issues of our approach are exemplified by the aforementioned case study of a study progress visualization in Section 4. Section 5 presents the most relevant related work in the context of integrating existing functionality, modeling workflow with the UML and transformation into Web services. Section 6 concludes this article, discusses the achieved results, and presents an outlook as well as suggestions for future research work.

II. BACKGROUND

The necessity for building integrative solutions comes along with the evolution of the way Information Systems are used. We thereby define an Information System (IS) as all the components and algorithm that are necessary for enabling IT-based computation of information. In the very beginning of supporting business activities through IT, the typical architecture of an IS was structured using two logical layers – a layer representing the client side and a layer representing the centralized business logic and data stores –

typically on a single tier or host. The availability of today's high performance networks connecting datacenters all over the world has lead to a multi-layer and multi-tier structuring of distributed business logic and distributed data stores. Figure 2 shows a comparison of these two approaches (based on [20]). It is obvious that while in the late 1960's one centralized datastore and computation logic served multiple clients, we are faced not only with distributed computation logic today but also distributed data stores that might be connected with many-to-many relationships amongst together.

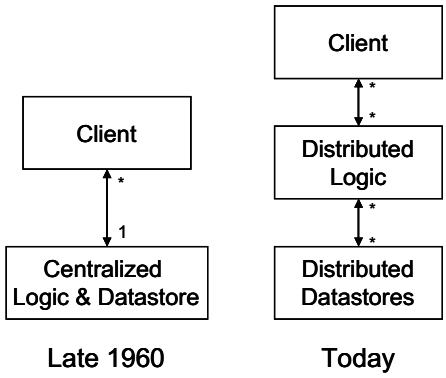


Figure 2. Evolution of Information Systems

Although separating the business logic from the data stores and enabling access to these components in a distribution way has many advantages, a logically holistic access to the information these systems contain is desired. Furthermore, not only access to information is needed but also an evolution of functionality is desired. Therefore reasons for aiming for integrative solutions in order to capture a holistic access to information are manifold: increasing the efficiency of business-related tasks, extending existing applications with new functionality, reusing existing applications, saving of software investments, avoid the costs of introducing a new software system. According to [21] the term "application integration" describes a strategic approach to couple different existing applications for the purpose of simplifying business-related tasks. The benefit of integration within these scenarios is more than just a conciliation of existing functionality. Rather, the idea is to leverage synergetic effects and thus to increase efficiency of IT-supported business activities.

For the purpose of constructing software-based solution logic, a clear distinction of the necessary development steps is necessary. There exists several different classification schema for describing levels and approaches for application integration [20],[21]. A common agreement to a basic classification scheme comprises four categories: information-oriented application integration (IOAI), service-oriented application integration (SOAI), business process-oriented application integration (BPOAI) and user interface-oriented application integration, which is also referred to as portal-oriented application integration (POAI). For a better understanding of the conceptual contribution of our work, at least some basic knowledge of the fields of application of each of the single possibilities is necessary, which is why we briefly describe each single category and highlight the main points of interest.

A. Information-Oriented Application Integration

Information-oriented integration is a simple approach for integrating several existing systems by considering the extraction of information of a source system and deciding how to convict this information into one or many different target systems. Almost any information system that is used today in a typical business scenario follows an n-tier architecture based on a database or a data store component enabling the information-based integration to easily be introduced. This is often the only possibility if changes to the business logic of an existing information system cannot be performed.

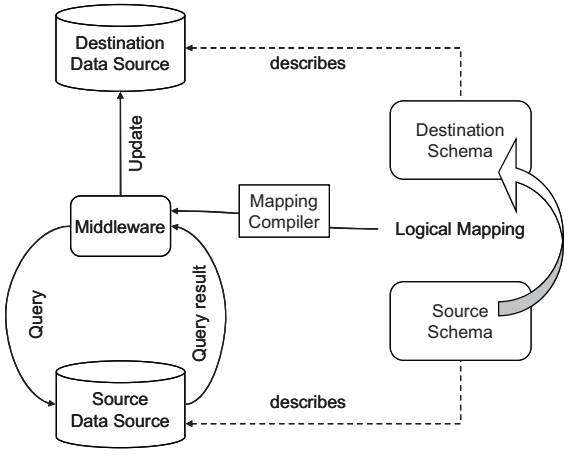


Figure 3. Concepts in Information-Oriented Application Integration

Figure 3 outlines the principles of this approach. The different data sources are described using different schema descriptions. These descriptions have to be logically mapped to each other, which can either be done at runtime or in less frequent cases, at design time.

Although the advantages of this approach are its simplicity and often fast-to-develop solution, this approach bares a couple of disadvantages. Often it is not clear in advance to what extends the desired solution needs to be based on integrated data stores. This leads to a couple of single integrative island with a total amount A where $A = 1/2 * \nu * (\nu+1)$ connections for ν different data stores ending in solutions that are hard to maintain or to evolve.

Further, this approach is not aligned with requirements derivable from the overall business processes, making changes at the business level hard to be propagated to the supporting IT level.

B. Service-Oriented Application Integration

Often it is not sufficient to only consider the information that existing applications operate on but also to enable access to functional capabilities the existing applications offer. By focusing the functionality in means of interface-oriented

semantics, one can think of a service a distinct application offers, leading to a new aspect of layering the possibilities of integration: service-oriented application integration (SOAI).

Simply speaking, SOAI allows applications to share common business logic [21]. The idea is to identify objects of reuse within an organization and enable access to these reusable objects through standardized service interface. Although the concept of reuse can hardly be assumed in advance, in the last years a couple of technologies and platforms to realize this vision have surfaced, namely Web service technology with its standardized interface descriptions and access methods using WSDL [3], XML [4] and SOAP [22].

Service-orientation is rather a design decision than a concrete solution to all the integration issues. Many questions are still open, for instance, it often is not clear how to design services in order to fulfill certain quality attributes such as loose coupling.

C. Business Process-Oriented Application Integration

While IOAI occurs at the level of data exchange [21], the concept of business process-oriented application integration (BPOAI) can be seen as an advancement of the IOAI and SOAI by focusing not only the data level of each single application but considering the overall process that each of the single applications participate with. Therefore, BPOAI takes the flow of information on a more abstract level, enabling an admission to the integration issue on a level that is more independent of the concrete data schemas.

Having the existing applications organized using service-oriented interfaces can be seen as a requirement for an efficient support of the business processes. Typical implementations of service-oriented integration scenarios are based on Web services, thus enabling the descriptions of interfaces and exchanged data schemas using XML. Such a standardized approach enables the usage of models for the descriptions of the processes and is proposed to lead to a more formal approach to solve integration issues, as models can easily be reused. Therefore, great efforts have been made to introduce modeling languages that are based on formal meta models in order to support direct transformations from the modeled business process to a concrete architecture supporting these modeled processes. Examples of these languages include the UML Activity Diagrams or the Business Process Modeling Notation (BPMN) [23] with its upcoming release 2.0 but also approaches that are based on mathematical formalism such as Petri nets [24] or event-driven process chains [25].

D. User Interface-Oriented Application Integration

Focusing the user interface for solving integration issues aims at enabling a single point of access to a multitude of existing user interfaces the existing applications have. While the first three approaches (IOAI, SOAI, and BPOAI) consider the exchange of information for the purpose of automating parts of business processes, the application integration on the level of user interfaces takes the human factor into account. Still many steps of business processes cannot be fully automated today, thus the need for enabling access to information or operations distributed applications provide are still a requirement. One of the biggest problems with user interface-based integration is the fact that almost any graphical user interface (GUI) of traditional business applications is tightly coupled with monolithic frameworks making it all but impossible and infeasible to make the functionality a GUI offers available to external software artifacts.

In the recent yeas, the concept of portals evolved for Web-based applications. User interface-oriented application integration therefore is often called portal-oriented application integration (POAI). A portal thereby is a Web browser-based approach for constructing a distributed architecture consisting of a portal server, a framework for generating and operating pluggable parts (portlets) of the user interface and a couple of connectors to access the existing applications.

E. Discussion

We currently observe a shift from simple information-based approaches to service-oriented approaches because of the several advantages of SOAI and because of the disadvantages of the other approaches pointed out in the previous descriptions. Although IOAI can be the solution to choose in small scenarios, SOAI proposes to have a better acceptance due to the enablement of business process-oriented aspects. A clear separation of IOAI and SOAI can be given by arguing that IOAI solely focuses the exchange of information, while SOAI considers not only the information but also the methods that operate on the information. As a major goal of the integration project we applied our development approach on was to create a solution that is accepted by a wide range of students having different skills in using Web-based information systems, the integration of the existing applications should lead to a single user interface that can be extended with further functionality on demand. Figure 4 relates the different aspects of integration of our approach, showing the flow of information across all elements of the architecture to be integrated.

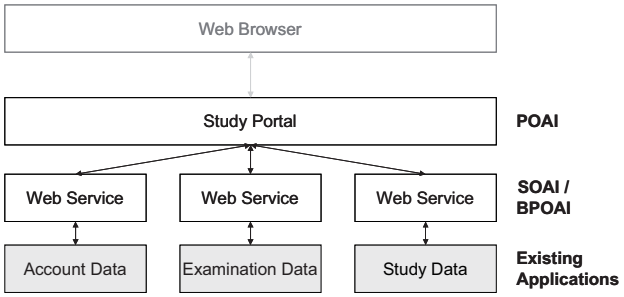


Figure 4. Different Aspects of Integration in our Approach

Our development approach therefore focuses the construction of service-interfaces for supporting both a flexible realization of business processes and usage of portal technology. For the purpose of simplifying the development method, we equivalently use the term “business process” and “workflow” as we focus on the technical representation of a business process which we consider to be a workflow [36].

This is not a constraint in our opinion as we argue that a technical representation of a business process can be found and the relevant automatable steps can be identified. For increasing the quality of the engineered solution, we use models and model transformations to omit unnecessary manual steps in code generation. Figure 5 outlines the steps in our development process that are based on model transformations and shows which of the development steps have to be performed using manual model transitions.

III. SERVICE-ORIENTED INTEGRATION USING A MODEL-DRIVEN APPROACH

In this chapter we describe our model-driven software development approach for a service-oriented integration of legacy applications using the Unified Modeling Language (UML). The overall goal of the development process is to develop a new high-level service by integrating legacy applications via service adapters (low-level services) and specify a workflow to compose those low-level services in such a way that they create the high-level service. Since existing applications used in the composition might be substituted in the future by newer applications, which provide new or better functionalities, the final solution must consider the adaption to a changing IT-landscape with reasonable time and effort.

The proposed development process contains six steps but does not cover the whole software lifecycle. For example, the test and deployment step is omitted as it does not significantly differ from other software development processes. Furthermore our solution should be adapted to the requirements of the integration project, adding new steps or leaving out steps described here.

Our development process starts with the definition of the requirements (Figure 5: A). Due to our experience we suggest the development of a prototype of the graphical user interface, but other methods can also be used. After capturing the requirements the next step is to develop a domain model (Figure 5: B). It contains the entities derived from the requirements. The domain model is of conceptual nature and independent from concrete applications. Having completed the domain model, the development process continues with the design of the workflow according to the defined requirements (Figure 5: C). In this step we regard the available legacy applications but also use the entities from the domain model created in the previous step. This allows us to avoid dealing with data transformation issues, since in the legacy systems the entities might be represented as different data types. Afterwards, model-driven transformation techniques are applied, generating formal interface descriptions and executable workflow definitions by transforming the workflow model and the domain model into a service model (Figure 5: D). All created and generated models so far are independent from a concrete technology. A final transformation step generates the necessary Web service artifacts – WSDL for service interfaces, XML Schema for data types and BPEL for workflow definition from the service model (Figure 5: E). Note that we use Web service technologies for the integration solution since it is most common, yet the transformations can be rewritten to generate artifacts based on other technologies than Web services using the same models (as suggested by the concept of Model-Driven Architecture). At last the Web service adapters are implemented based on the generated WSDL interfaces, which is still done by manual work (Figure 5: F). The adapter can either accesses a database used by the legacy application or uses a native interface provided by the legacy application.

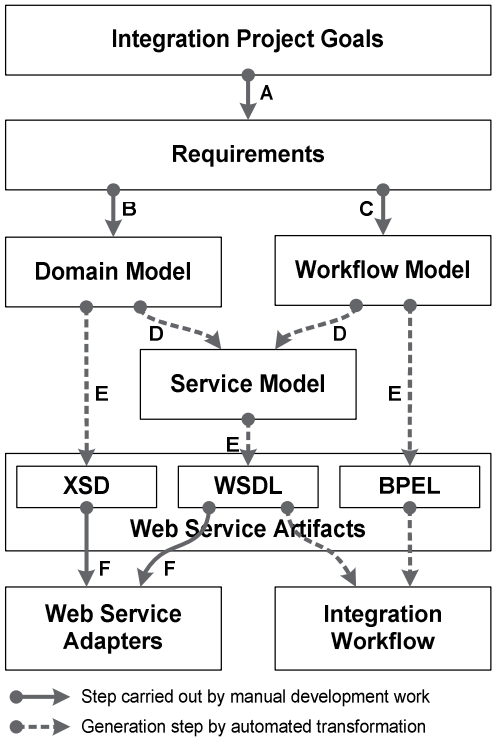


Figure 5. Model-Driven Development Process Overview

In the following the development process is described in detail.

A. Capturing the Requirements

The requirements needed for designing the integration solution can be captured using manifold techniques. All techniques for requirement analysis have in common that there is a close collaboration between the customer and the architect or similar roles, since only the customer knows what he expects from the final software solution, but cannot express it in an unambiguously and well-formed way.

Some traditional techniques for requirement elicitation are introspection, questionnaires, interviews or brainstorming [11]. Representation-based techniques use descriptions of scenarios or use cases. Our development approach does not prescribe a concrete technique but rather allows the developer to choose one that fits best to the project. Since we propose an approach for integration scenarios, an important part after the requirement elicitation is to analyze existing applications and systems, which are required to fulfill the functional requirements.

A common approach that works well in our experience is the prototyping of the graphical user interface, since it gives the customer a "look-and-feel" of what the final solution might look like. The requirements can than be deduced from that prototype.

B. Creating the Domain Model

Based on the defined requirements, the next step is to create a domain model. A domain model is a static model that contains relevant entities of that domain and their relations [26]. It does not contain dynamic issues, like sequences or information flows. Consequently we use UML class diagrams so that each entity in the domain model is formalized as an UML *Class* with typed *Properties.* Furthermore we use *Associations* and *Generalizations* (note that all UML meta classes are written in *italic*) to specify relations between entities. All *Properties* should be typed as primitive data types (like string, integers, boolean values etc.). Relations between entities are formalized by modeling *Associations* between the corresponding *Classes*. *Packages* can be used to classify entities into logical groups.

In contrast to the approach described in [1] this approach propose to design the entities rather abstract by understanding the domain the integration project is settled in and not derive them from the existing legacy applications. Nouns and noun phrases in the requirements specified in the previous step can help to identify relevant entities. Although this seems to be a rather complex and time-consuming task than deriving the entities directly from the legacy applications, the effort put into the domain model will pay off later when new or changing requirements have to be implemented or new applications substituting the legacy applications have to be integrated into the existing solution. We also recommend using or building upon existing standardized domain models, such as ebXML in the domain of electronic business [35], complementing our use of open standards for describing the service interfaces and data types. Using such standardized domain models simplifies the integration of functionality of third party services as well as it provides the ability to offer the service to third parties.

With the domain model, our integration solution use data types that are independent from the concrete legacy applications. The workflow only operates with data objects from the domain model and therefore does not have to cope with transformations of different data types representing the same entity that otherwise would be necessary to implement in the workflow. At runtime the data transformations between the legacy applications and the workflow are carried out in the service adapters. Hence the service adapter transforms native data objects from the legacy application into data objects as defined in the domain model and vice versa. The legacy applications can be replaced by new applications at the cost of rewriting the relevant service adapters, but the workflow does not have to be adapted, since the interface of the service adapter remains the same.

C. Designing the Workflow Model

After the completion of the domain model, the next step is to design the workflow model in a bottom-up way. During the execution of the workflow, the legacy applications are invoked and provide required data or execute actions. In contrast to the domain model, the workflow model is a dynamic model. The workflow model makes use of the entities defined in the domain model though. In the model the workflow itself is represented by an UML *Activity* (c.f. Figure 8: Source model, *Activity* "Wf").

Many workflows require some initial data transferred from the invoking application before the workflow can be executed. Also after the completion of most workflows some data is returned to the application that has called the workflow. To specify those data objects, the activity can have *ActivityParameterNodes* attached to it (Figure 8: "wfIn", "wfOut"). An *ActivityParameterNode* always has a reference to a *Parameter.* The *Parameter* is either typed with a primitive type or a concrete *Class* from the domain model. Furthermore the Parameter requires a direction type that indicates if the value of the Parameter is passed into the workflow or from the workflow.

To represent the legacy applications in the workflow model that are invoked at runtime *ActivityPartitions* are used (Figure 8: "AppX"). *ActivityPartitions* are usually used to group *Actions* in an *Activity* that share some common characteristics, e.g., belonging to the same organization unit. The *Activity* representing the workflow must contain at least one *ActivityPartition*, because otherwise there would be no legacy application to call.

To call a legacy application, *CallOperationActions* are modeled (Figure 8: "OpX"). *CallOperationActions* are more specialized *Actions*, which have a reference to an *Operation*. As a minor restriction, it is not possible to invoke more than one application within one invocation. Therefore, each *CallOperationAction* must be contained in exactly one *ActivityPartition*. However, since one application can be invoked in many ways to retrieve different data sets, an *ActivityPartition* can contain several different *CallOperationActions*.

The activity diagram is refined by specifying the type of data sent to or retrieved from the invoked applications. The type of data sent to an application by one invocation is modeled by adding *InputPins* and/or *ValuePins* to the *CallOperationAction* (Figure 8: "xIn"). In contrast, *OutputPins* represent the data returned from an application (Figure 8: "xOut"). According to the UML meta model [1], a *Pin* is derived from the *TypedElement* and the *MultiplicityElement* meta class by Generalization. The former enables the user to type a *Pin* with a *PrimitiveType* (such as String, Integer, etc.) or one of the data objects modeled earlier as a *Class*. The later allows the collection of complex data structures in one invocation. The same applies for the *ActivityParameterNodes*.

To represent the data flow between the invocations, we add *ObjectFlows* between *InputPins* and *OutputPins*. The *ObjectFlows* also specify in which order the invocations must be executed. Additionally, if a typed *InputPin* does not have a matching incoming *ObjectFlow*, the required data has to be collected by an additional *invocation*. In such a case, we need to model new *CallOperationActions*, which return the required data and provide an *OutputPin* for that. Of

course, the appropriate application which holds the data must be known in advance. Thus the application has to be added as an *ActivityPartition*, if not present yet.

The model containing the *Activity* formalizes the workflow and the legacy applications to be invoked. Due to the *ObjectFlows* it is further specified how data is processed in the workflow and in which order the invocations occur.

D. Transformation to Service Model

To generate standardized Web service-based interface descriptions and data types, the next step is to generate a new model by using model-driven transformation techniques. From the domain model and the workflow model a transformation specification generates a service model [7], which, among other details, specifies the interfaces for each legacy application and the study progress workflow itself.

Since the transformation to the service model generates a new model from existing models, the transformation rules are formalized in the transformation language "Queries, Views, Transformations" (QVT) [12], a standard specified by the Object Management Group (OMG). Several QVT implementation exists, e.g. in Borland Together [27], but also as plug-ins for the Eclipse IDE (mediniQVT [28], smartQVT [29]). Simply speaking, the transformation rules are described by mapping the elements of the source meta model to elements of the target meta model. Since the source meta model and target meta model is the UML Superstructure [2] the transformation itself is independent from a concrete platform or technology and thus can be reused for other integration projects of the same kind.

The transformation uses the created *Activity* and the containing model elements as the source model and generates a target model according to a set of transformation rules. Since each *ActivityPartition* represents an application, which will be invoked during the execution of the workflow, each *ActivityPartition* is transformed into an *Interface* (stereotyped as "ServiceInterface") and a *Component* (stereotyped as "ServiceComponent") with a *Realization* relationship between (c.f. Figure 8: Target model, "AppXService" and "AppX"). Each *CallOperationAction* contained in an *ActivityPartition* results in an *Operation* of the created *Interface* (Figure 8: "+opX()"). Figure 6 shows this transformation in the graphical notation of QVT.

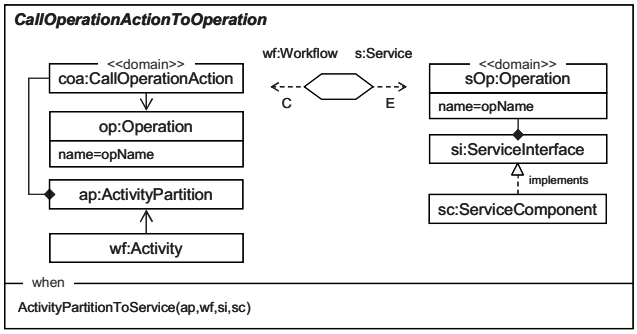


Figure 6. QVT Transformation of CallOperationAction

Finally, *InputPin*s and *OutputPin*s of the *CallOperationActions* are converted into *Parameter*s of the *Operation* (Figure 8: "wfIn" and "wfOut"). The *direction* property of each *Parameter* is set to "*in*" if it is an *InputPin* and no corresponding *OutputPin* of the same type and name is attached to the same *CallOperationAction* (Figure 7). An *OutputPin* results in the direction "*out*". If a *CallOperationAction* has an *InputPin* and an *OutputPin* with the same name, the same type and the same multiplicity, the direction property of the *Parameter* is set to "*inout*" and the *OutputPin* is ignored.

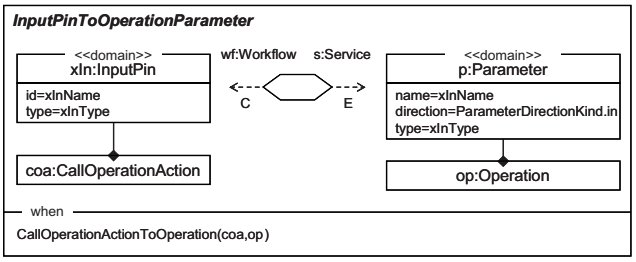


Figure 7. QVT Transformation of InputPin

In order to invoke the workflow itself an additional *Interface* and *Component* are generated from the *Activity* (Figure 8: "WfService" and "Wf"). The *Interface* contains exactly one *Operation* named "execute<ActivityName>" (Figure 8: "+executeWf()"). The *Parameters* for this *Operation* are generated according to the *ActivityParameterPins* attached to the *Activity* (Figure 8: "wfIn" and "wfOut"). In total, n + 1 *Interfaces* are generated, whereby n correlates to the number of invoked applications (or *ActivityPartitions*). Finally, the generated *Component* has *Uses* relationship to all other *Interfaces* generated from the *ActivityPartitions*.

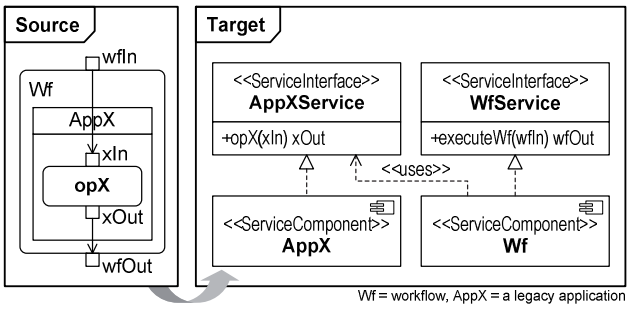


Figure 8. Transformation to the Service Model

It is not required to transform the entities from the domain model. Still, the specified data types are needed in the target model. Therefore the *Classes* from the source model, which represent the data types can either be imported in the target model or copied to the target model. The same applies for the *Activity* and the containing *Actions*. The property "*operation*" of the *CallOperationActions* can now be associated with the generated *Operations* of the *Interfaces*.

E. Transformation into Web Service Artifacts

As the final modeling step, we transform the three UML models into concrete XML artifacts. The transformation converts the domain model into XML Schema definitions [4], the service model into WSDL documents [3], and the workflow model into a BPEL process [17, 30]. As far as we know the relatively new "MOF Model to Text Transformation Language 1.0" [31] specified by the OMG is currently not implemented in common UML tools. As we also prefer a more established Model-to-Text transformation language, we decided to use Xpand [32], a templated-based approach from the openArchitectureWare toolkit, which is now part of the Eclipse IDE.

TABLE I. DOMAIN MODEL TO XML SCHEMA

UML	XML Schema
Package p	`<xsd:schema>` target namespace is derived from *p* and its parents

Class c	`<xsd:complexType>`
	name of complex type is name of *c*
	`<xsd:sequence>`
	model group for elements

Attribute a	`<xsd:element>` name of element is name of *a*, type of element is used from *a* (Primitive types are matched to similar build-in XSD types), minOccurs and maxOccurs of element is cardinality of *a*
Association s	`<xsd:element>` name is set to name of *s*, type of element is complex type of *Class* at *AssociationEnd*, minOccurs and maxOccurs of element is cardinality of *s*

Enumeration e	`<xsd:simpleType>`
	name is set to name of *e*
	`<xsd:restriction>`
	base type of restriction is set to "string"
	`<xsd:enumeration>`
	for each *EnumerationLiteral el*, value is name of the *el*

Generalization g	`<xsd:complexContent>`
	as child for complex type of specialized *Class*
	`<xsd:extension>`
	base type is set to complex type of generalized *Class*

The entities defined in the domain model and specified as UML *Classes* are transformed into XML Schema definitions (XSD) [3]. Most transformation rules are mainly straightforward (Table I). The name of the model and the structure of UML *Packages* (if present) are used to generate the target namespace of the XSD. UML *Classes* are transformed into XSD complex types with a sequence model group and all *Properties* of *Classes* into XSD elements of the generated model group. These XSD elements are typed depending on the kind of the UML *Property*: If it is an *Attribute* with a primitive type, a build-in XSD data type is used. If the *Attribute* uses a custom UML *Enumeration*, an additional XSD simple type with a restriction of base type set to "string". For each literal of the Enumeration an enumeration with the value set to the name of the literal is generated. If an *Association* is used, the XSD element is typed with the corresponding XML complex type of the associated *Class*. Furthermore the cardinalities of *Properties* are considered by using the "minOccurs" and "maxOccurs" attributes in the XSD element definition. UML *Generalizations* are also supported by complex content and extensions in XSD, using the complex type of the generalizing *Class* as "base" attribute of the "extension" element.

TABLE II. SERVICE MODEL TO WSDL

UML	Web Service Description Language (WSDL)
Component c	`<wsdl:definition>` name of definition is name of *c*
Interface i	`<wsdl:portType>` name of port type is name of *i* appended with "PortType"

Operation o	`<wsdl:operation>`
	name of operation is name of *o*
	`<wsdl:input>` and `<wsdl:ouput>`
	name of message is name of *o* appended with "Request" or "Response"
	`<wsdl:message>`
	name of message is name of *o* appended with "Request" or "Response"
	`<wsdl:part>`
	name is "parameters" and element is corresponding XSD element name
	`<xsd:element>`
	name of element is name of *o*, appended with "Response" once
	`<xsd:complexType>`
	`<xsd:sequence>`
	model group for *Parameters*

Parameter p	`<xsd:element>` name of element is name of *p*, containing model group of element depends on *p.direction* ("in" or "out"/"reply"), type of element is type of *p* (either a build-in XSD type or a complex type)

The WSDL documents are generated from the service model (Table II). Each UML *Interface* is transformed into a WSDL document, importing the generated XSD files in the "types" section of the WSDL document. Each interface is transformed into an abstract part of a WSDL file with one port type. The port type contains the operations as the UML *Interface* specifies. The generation of the messages for the input and output messages of the Web service depends on the WSDL style. Since it is most common and recommended by WS-I [13], we use the style "document/literal-wrapped" [14]. For this style, each message element in the WSDL document must contain exactly one part, even if multiple UML *Parameters* are specified as input or output. To distinguish between the Parameters, XML Schema is used to build an RPC-like XML structure, using the operation name as the top XML element and an embedded complex type

defining a sequence of child elements, which represent Parameters. To generate the concrete parts of the WSDL file, the proposed service model uses UML *Components* and attached *Ports*, as in [6], [9]. A *Port* acts as a WSDL binding, specifying a name and location information about the service. It refers to the *Interface* as provided interface.

The workflow model is implemented in the Business Process Execution Language (BPEL) [30] and is provided as a Web service. The BPEL code is generated from the UML Activity Diagram, which is already handled in some works [17], [18]. The *Activity* and each *ActivityPartition* of the workflow are defined as partner links in the BPEL process and partner link types in the WSDL document of the BPEL process. The defined *ActivityParameterNodes* of the *Activity* transforms to the initial "receive" and the final "reply" action in the BPEL process. In addition corresponding variables for are generated. Each *CallOperationAction* is transformed to an "invoke" action, using the partner link derived from the *ActivityPartition* the *CallOperationAction* is placed in. Corresponding variables in BPEL for the input and output message of the "invoke" action are generated. *ObjectFlows* in the Activity are transformed into "assign" actions, which copy the content of the output variable of one invocation to the input variable of a following invocation. The sequence in which the BPEL actions occur is mainly determined by the *ObjectFlows*. However, since UML Activity Diagrams are based on graphs, whereas BPEL is structured in blocks, the generation of the BPEL processes has certain limitations [33] that we are aware of.

F. Implementing the Web Services Artifacts

To finalize the integration, the required Web services have to be implemented. The generated WSDL and XML Schema files are used to create skeletons for the adapter logic implementation of the Web service. For this purpose, existing approaches are applied that are part of several development tools (like WSDL2Java from the Apache Axis2 framework [15]). The choice of the programming language and the framework for generating and implementing the web service adapters should be depend on the legacy application, so that existing interfaces of the legacy application can be used if possible. In addition the Web service adapter must also provide the transformation into the data objects specified in the domain model. To deploy the generated BPEL process usually a deployment descriptor has to be created, which is specific to the selected BPEL engine.

IV. CASE STUDY "STUDY PROGRESS"

The KIT offers its students the KIT-Portal [18], where each student can access his/her personal data and perform actions (e.g., to register for an examination) in a simple and intuitive way. In this chapter, we apply our model-driven software integration development process presented in the previous chapter to the development of a service-oriented application to visualize a student's progress in his/her studies for the KIT-Portal.

The KIT-Portal integrates several existing applications in a service-oriented manner using Web technologies and Web standards. At the KIT, several applications are available, each storing and providing individual data for students. However, none of the applications provides interoperable interfaces, hence preventing an easy and straightforward service-oriented integration. An important step towards service-orientation is the development of standardized and technology-neutral interfaces for accessing and manipulating the data provided by existing legacy applications [9]. These interfaces and the corresponding adapter logic have to be developed to allow the integration of existing applications.

A. Analysing the Requirements for the "Study Progess"

One feature of the KIT-Portal to be developed is to facilitate a student's overview of his/her passed, failed or outstanding examinations in a graphical and easily understandable manner. Hence, several GUI sketches and prototypes were created prior to starting the development process, to get the look-and-feel for an adequate visualization form of the study progress. A modified version of a tree map provided the most promising results. It visualizes all the learning modules of a study course by rectangles using an equal width, but different height, depending on the amount of credit points (c.f. European Credit Transfer System, ECTS) of the module. The same applies for the examinations allocated to a module. In addition, each examination is color-coded depending on the current state or result with regard to the student.

The required data for generating the study progress visualization are persisted in two legacy systems: The study system stores the degree programs and its structures, whereas the examination system holds the data for the offered examinations and the examination results for each student.

B. Creating the Domain Model "Study Progress"

Having defined the requirements, we model the domain and needed data objects for the study progress tree map, such as examination results or information about the student. We create a UML model and model the entities as *Classes*. We also add the data structure which is needed to generate the study progress tree map (Figure 9: B).

C. Designing the Workflow "Study Progress"

Next we design the workflow bottom-up. The workflow for visualizing the study progress is represented by a UML *Activity* "StudyProgress" (Figure 9: C). To specify the data types the workflow is called with respectively returns, the *Activity* has two *ActivityParameterNodes* attached to it. The KIT-Portal invokes the study progress workflow by passing the login name from the KIT-Portal (student's university e-mail address) as initial input data (*ActivityParameterNode* "loginEmail") of type string. The workflow completes by returning the output type of the workflow is the tree map data type (*ActivityParameterNode* "studyProgress"). The study system and the examination system are modeled as *ActivityPartitions* ("Study" and "Examination"). The invocation to one of the systems is modeled as a *CallOperationAction* in the corresponding *ActivityPartition* and in addition the type of data transferred to or from a system on each invocation is added as *Pins*.

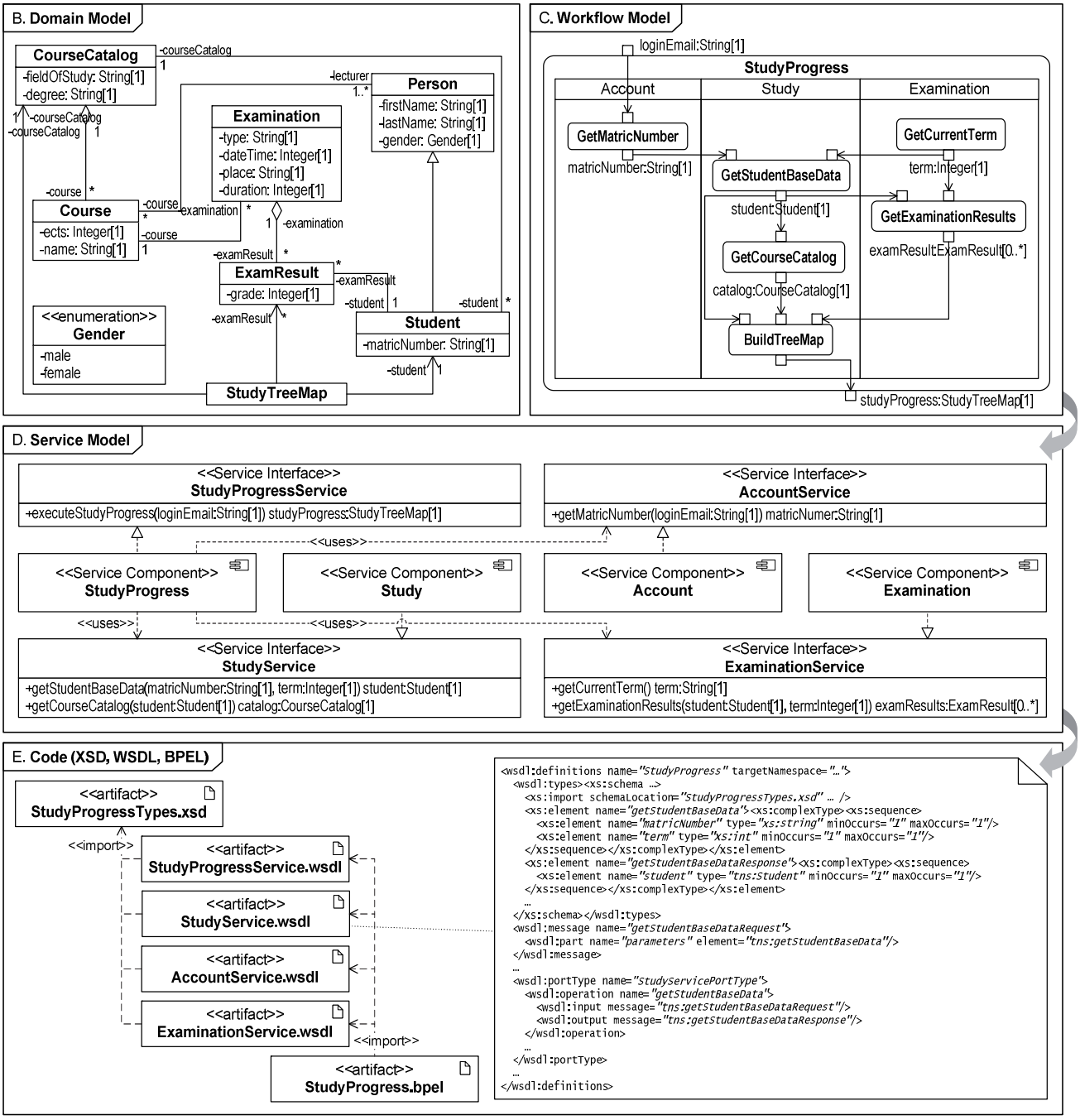


Figure 9. Model-Driven Development Process of the "Study Progress"

For example in order to receive the student's base data from the study system we model the *CallOperationAction* "GetStudentBaseData" in the *ActivtyPartition* "Study" and add the *OutputPin* "student" of the type "Student" (the classes modeled before). The call to the study system requires the matriculation number and the current term, so we model those by adding the two *InputPins* "matricNumber" and "term". Since the portal system only knows the student's university e-mail address, which has to be entered during the KIT-Portal login, we add an *ActivityPartition* for the accounting system and model the *CallOperationAction* "GetMatricNumber" inside. It accesses the accounting system, maps the student's email address to his/her matriculation number and returns the number (*OutputPin* "matricNumber"). The current term can be retrieved from the examination system. Thus, we add the

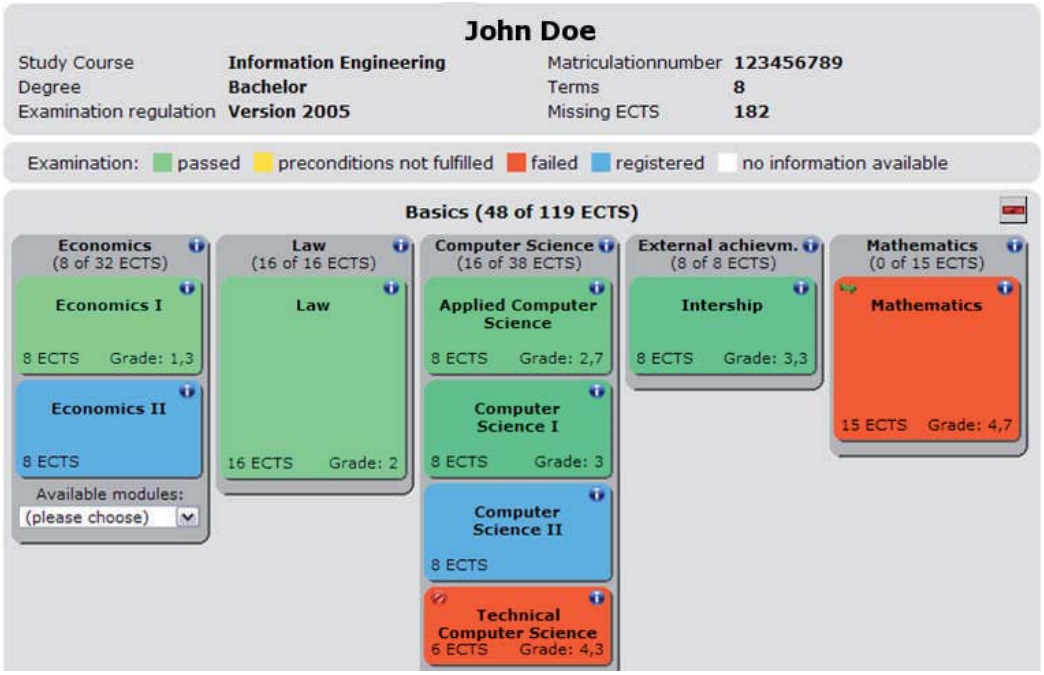


Figure 10. Screenshot of the "Study Progress"

CallOperationAction "GetCurrentTerm" in the *ActivtyPartition* "Examination" with only one *OutputPin* "term" containing the current term as an integer value.

To represent the data flow between the invocations, we add *ObjectFlows* between *InputPins* and *OutputPins* that have the same type. The *ObjectFlows* specify how data objects flows from the outcome of a previous invocation to the input of a following invocation during the execution of the workflow. Thus the order in which invocations occur can be derived from the *ObjectFlows*.

We have formalized which applications are invoked and how the data is processed. Figure 9 shows the final workflow model as an UML Activity Diagram labeled as "Study Progress" in part C.

D. Transformation to the Service Model

Taking the activity diagram as a source model, we use a QVT-based model-to-model transformation to generate service interfaces using the QVT transformation rules described above. The transformation generates a service interface for each invoked application. In order to invoke the workflow itself from the KIT-Portal, another service interface "StudyProgressService" that contains the *Operation* "executeStudyProgess" is generated. The Parameters for this *Operation* are generated according to the *ActivityParameterPins*.

Part D of Figure 9 shows the resulting *Interfaces*, *Components* and the relations for each *ActivityPartition* and the *Activity* itself.

E. Transformation into Web Service Artifacts

The model-to-text transformation creates an XML Schema file [4] ("StudyProgressTypes.xsd") from the Classes in the domain model, four WSDL files [3] (one for each service interface) from the Interfaces and Components in the service model and a BPEL file ("StudyProgress.bpel") from the workflow model. To facilitate the reusability of the XML Schema definitions the StudyProgressTypes.xsd file is imported into the "types" section of each WSDL file. Also all WSDL files are imported into the BPEL process file in order to act as partner links.

Figure 9 illustrates the generated artifacts and the import of the central XML Schema definition at the bottom. Part of the WSDL document for the StudyService is also shown in detail.

F. Implementing the Web Service Adapters

Finally, the generated WSDL documents are used to create skeletons. We implement the adapter logic of the required Web services using Java. We further use an XSL transformation to generate XHTML from the tree map data structure defined in the domain model. Figure 10 gives the result of the engineered solution, showing a late prototype of the study process.

V. Related Work

As our approach targets a wide area of different artifacts supporting a model-driven development approach (service model, WSDL and Web services), there are several related studies.

The idea of visualizing hierarchically structured information in terms of tree maps initially was published by Johnson and Shneiderman [37]. Based on their concepts, Allerding, Buck et al. present an approach using tree map concepts and focusing the requirements of students managing their studies [38]. Adapting their idea of

visualizing the study progress of a student, we used an early sketch of a tree map as input for a model-driven development approach. The execution of integration projects following a model-driven development approach based on Web service technologies is discussed in scientific and commercial communities alike.

Considering the overall development approach, starting with formal requirements and leading to a set of executable code, Meijler, Kruithof et al. illuminate the advantages of model-driven integration aligned with service-oriented principles [5]. An integrated approach combining both top-down (requirements to software components) and bottom-up (existing tool assets) approaches is proposed. Therefore, we decided not to strictly follow a top-down development approach that would hinder the integration of existing applications, but to follow a combined middle-out approach enabling the description of existing applications early in the transformation process.

Model-driven development of Web services has already been discussed in several previous works, for instance in [6], [7], [8]. Based on these approaches, we focused on capturing business requirements with models and mapping these models to existing distributed legacy applications. Considering the integration of legacy applications using Web services, a generic model for application integration is presented in [9]. Since different legacy applications often use different formats and standards for describing their data schemas, a mapping of these different data schemas has to be realized additionally. The proposed approach in [9] focuses on the integration of several different data schemas by implementing adapter components realized with Web services. Within the special requirements of our scenario, not only the integration of existing data schemas but also the integration of existing business logic is needed; thus our approach considers the aspect of integration from a system-oriented direction.

Finally, the presented intermediate model for service descriptions (c.f. chapter 3) is based on the work of Emig, Krutz et al. [7]. While the approach presented in [7] targets towards a holistic and technology-independent possibility for describing service interfaces of service-oriented components, we improved the proposed development approach by the integration aspect of existing software assets. Similar to [7], Johnson demonstrates the use of a technology-independent approach for describing service-oriented software components [10]. An UML 2.0 Profile [2] as an extension to existing modeling tools is proposed, although specific modeling elements are introduced regarding the very special needs of the appointed vendor-specific tool chain.

VI. CONCLUSION

In this paper, we outlined a development approach for integrating existing applications in a service-oriented manner by using a model-driven approach in order to create new functionality. In a first step GUI sketches help eliciting the requirements of the new functionality. Additionally a domain model is created statically describing the main concepts and their relationship of the domain. Afterwards, the necessary workflow, which determines the integration of the existing functionality from existing legacy applications, is modeled using UML Activity Diagrams. The workflow and the domain model are used to automatically generate artifacts which help in implementing the workflow. Such artifacts include adapters for accessing only required functionality of existing applications, relevant data types and an executable workflow for which we used Web service adapters using WSDL, data schemas described with XSD and BPEL process definitions respectively. Due to the usage of the domain model as source for the data types, the resulting services abstract from existing applications and their specific data types. This enables a wider usage of the created Web services without knowledge about platform specific details.

To exemplify our approach, we demonstrated our approach by realizing a study progress visualization at the Karlsruhe Institute of Technology (KIT). In this scenario, the goal was to provide a new feature that allows students to gain insight into their current study progress, by combining and visualizing their data from disparate sources. The required functionality and data is shared across several existing applications, so that they need to be integrated.

Even though a complete role model of an integration process is not in the focus of our work we are certain that our approach is helpful to all participants of an integration project. Our approach helps IT architects with the development of new functionality that requires the integration of existing applications. Domain experts are supported by visually describing the required functionality and integration experts can use the workflow to map the requested functionality to existing applications. Additionally the workflow is used to derive implementation artifacts by using automatic transformations, which helps in avoiding transformation errors due to human interpretations. As the approach started by gathering user requirements by means of a GUI sketch, we consider our solution user-aligned and a promising enhancement of existing integration approaches.

Although the application of model-driven approaches has several advantages, such as the convenient transformation of Web service adapters, data types and running BPEL processes from formalized design models, the usage of this modern technology is hampered by lack of a complete tool support. The application of UML Activity Diagrams enables the usage of several existing UML modeling tools and allows a formalized and visual description of the workflow. While there exist mature tools in the context of UML modeling, the development of transformations is still a complex task.

The choice of using service-oriented architecture as the integration platform proved to be the right choice for our approach, as reusing the existing business logic of the legacy applications can now achieved at a level of higher abstraction. Yet, the full potential of service-orientation such as the design of services to achieve certain design characteristics, the security of data within the workflow, the interaction with human users and the management of the services were not considered within our approach so far. These aspects are motivation for further work within this context and are part of our outlook.

VII. OUTLOOK

Based on our latest research results presented before, there are several topics we want to investigate in more detail. Firstly all functional components are provided as services via Web service interfaces. Therefore these services have to follow design principles to allow for loose coupling or to achieve a certain granularity etc. So far our model-driven approach does not take these design principles into account during the transformation from workflow model to service model. Focusing on service design principles, different variations of transformation rules could be applied to achieve a set of services with different attributes like granularity etc. suitable for different scenarios [43].

Secondly human users have to interact within a process by, e.g. inserting some data or making a decision. Hence user interfaces have to be developed to enable theses human tasks (c.f. [44]). Since the information which is to be passed along by the human user is directly correlated to the domain model [45], an automated generation of user interfaces can be achieved. In the same manner it could be possible to automatically generate several adapters like e.g. database adapters which are commonly used among several processes.

As a third perspective one should think about the whole application itself. As it consists of several services being provided by likely different providers, the users like students need to have a centralized way to report errors and to start a problem solving process like e.g. ISO 20000's Incident Management Process [34]. Therefore the development process also has to take management information and management processes into account to allow for a manageable and sustainable service-oriented application [46].

Another important aspect of any service-oriented integration scenario is the consideration of security aspects. So far no information about the security requirements of the newly composed services is incorporated into the development approach. Even though the existing applications might be secured in the sense of e.g., a secure connection, the proposed approach would reduce it to a secure point-to-point communication between the service adapter and the application instead of providing a secure end-to-end communication. A solution to this would be to add additional information concerning security requirements into the modeled process. Furthermore it can be seen from the case study, that a user might have different identifiers to access different applications and services in an organization again using different authentication mechanisms. In the current version of our approach this leads to a significant amount of operation invocation to map the initial identifier to every necessary identifier. A better approach would be to use a global identifier at the service level and enhance the model-to-text transformation presented to generate necessary mapping code directly into the Web service adapters.

We have already presented some initial results on these topics in [39, 40, 41, 42] and will now focus on integrating our findings to our model-driven approach.

REFERENCES

[1] Hoyer P., Gebhart M., Pansa I., Link S., Dikanski A., Abeck S.: A Model-Driven Development Approach for Service-Oriented Integration Scenarios. First International Conferences on Advanced Service Computing (SERVICE COMPUTATION), Athens, Greece, November 2009.

[2] Object Management Group (OMG): Unified Modeling Language (UML), Superstructure Version 2.2. http://www.omg.org/cgi-bin/doc?formal/09-02-02

[3] World Wide Web Consortium (W3C): Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/wsdl20/

[4] World Wide Web Consortium (W3C): XML Schema Definition Language (XSD) 1.1 Part 1: Structures. http://www.w3.org/TR/xmlschema11-1/

[5] Meijler T.D., Kruithof G., Beest N.: Top Down Versus Bottom Up in Service-Oriented Integration: An MDA-Based Solution for Minimizing Technology Coupling, LNCS Volume 4294/2006.

[6] Marcos E., Castro V., Vela B.: Representing Web Services with UML: A Case Study. 1st International Conference on Service-Oriented Computing (ICSOC), Trento, Italy, December 2003.

[7] Emig C., Krutz K., Link S., Momm C, Abeck S..: Model-Driven Development of SOA Services, Cooperation & Management, Universität Karlsruhe (TH), Internal Research Report, 2008.

[8] Gronmo R., Skogan D., Solheim I., Oldevik J.: Model-driven Web Service Development. International Journal of Web Services Research, Volume 1, Number 4.

[9] Harikumar A., Lee R, Yang H., Kim H., Kang B.: A Model for Application Integration using Web Services, Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science, July 2005.

[10] Johnston S.: UML 2.0 Profile for Software Services, IBM developerWorks http://www.ibm.com/developerworks/rational/library/05/419_soa/, April 2005.

[11] Hay D.: Requirement Analysis – From Business Views to Architecture. Prentice Hall, 2003.

[12] Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.0. http://www.omg.org/spec/QVT/1.0

[13] Web Services Interoperability Organization: Basic Profile Version 1.2. http://www.ws-i.org/Profiles/BasicProfile-1_2(WGAD).html

[14] Butek R.: Which style of WSDL should I Use, IBM developerWorks, 2003. http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

[15] The Apache Software Foundation: Code Generator Wizard - eclipse Plug-in, http://ws.apache.org/axis2/tools/1_0/eclipse/wsdl2java-plugin.html

[16] Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[17] Mantell K.: From UML to BPEL. IBM developerWorks, 2005. https://www.ibm.com/developerworks/library/ws-uml2bpel/

[18] Skogan D., Groemno R., Solheim I.: Web service compositions in UML. Proceedings of Eudth International Enterprise Distributed Object Computing Conference, September 2004.

[19] Karlsruhe Institute of Technology (KIT): The KIT study portal, http://studium.kit.edu

[20] Conrad S., Haselbring W., Koschel A., Tritsch R.: Enterprise Application Integration: Grundlagen – Konzepte - Entwurfsmuster – Praxisbeispiele, Spektrum Akademischer Verlag, 2005, ISBN 3827415721.

[21] Linthicum D.: Next Generation Application Integration, Addison-Wesley Information Technology Series, 2004, ISBN 02018445667

[22] World Wide Web Consortium (W3C): SOAP Version 1.2, http://www.w3.org/TR/soap/

[23] The Object Management Group (OMG): Business Process Model and Notation (BPMN) 2.0 Beta 1, http://www.omg.org/cgi-bin/doc?dtc/09-08-14.pdf

[24] Valk R., Girault C.: Petri Nets for Systems Engineering – A Guide to Modeling, Verification, and Applications, Springer, 2001. ISBN 978-3540412175.

[25] Keller G., Nüttgens M., Scheer A.-W.: Semantische Prozessmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes, Heft 89, Januar 1992.

[26] S. Johnston, “Rational UML Profile for business modeling”, IBM Developer Works, http://www.ibm.com/developerworks/rational/library/5167.html, 2004.

[27] Borland, Borland Together, http://www.borland.com/de/products/together/index.html

[28] ikv++ technologies ag: medini QVT, http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77&lang=en

[29] SmartQVT, http://smartqvt.elibel.tm.fr/index.html.

[30] Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf

[31] The Object Management Group (OMG): MOF Models to Text Transformation Language V1.0, http://www.omg.org/spec/MOFM2T/1.0/PDF

[32] Eclipse Foundation: Xpand, http://wiki.eclipse.org/Xpand.

[33] Ouyang C., Dumas M., Breutel S., Hofstede A.: Translating Standard Process Models to BPEL, Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Luxembourg, Luxembourg, June 5-9, 2006, Proceedings 2006.

[34] ISO/IEC, ISO/IEC 20000-1:2005: Information Technology – Service Management, www.iso.org, 2005.

[35] Organization for the Advancement of Structured Information Standards (OASIS): ebXML Technical Architecture Specification v1.04, http://www.ebxml.org/specs/#technical_specifications.

[36] The Workflow Management Coalition Specification: Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011),http://www.wfmc.org/standards/docs/TC011_term_glossary_v3.pdf, 1999.

[37] Johnson B., Shneiderman B: Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures, IEEE Computer Society Press, http://hcil.cs.umd.edu/trs/91-06/91-06.html, October 1991.

[38] Allerding F., Buck J., Freudenstein P., Klosek B., Höllrigl T., Juling W., Keuter B., Link S., Majer F., Maurer A., Nussbaumer M., Ried D., Schell F.: Integriertes Service-Portal zur Studienassistenz, Proceedings of the 38th GI Conference - Lecture Notes in Informatics, München, Germany, Munich, 2008.

[39] Gebhart M., Abeck S.: Rule-Based Service Modeling, The Fourth International Conference on Software Engineering Advances, ICSEA 2009, 20-25 September 2009, Porto, Portugal 2009.

[40] Link S., Hoyer P., Kopp T., Abeck S.: A Model-Driven Development Approach Focusing Human Interaction, Second International Conference on Advances in Computer-Human Interaction, ACHI 2009, February 1-7, 2009, Cancun, Mexico 2009.

[41] Scheibenberger K., Pansa I.: Modelling dependencies of IT Infrastructure elements, Proceedings of BDIM 2008, 3rd IEEE/IFIP International Workshop on Business-Driven IT Management, April 7, 2008, Salvador, Brazil 2008.

[42] Klarl H., Wolff C., Emig C.: Identity Management in Business Process Modelling: A model-driven approach, Business Services: Konzepte, Technologien, Anwendungen. 9. Internationale Tagung Wirtschaftsinformatik 25.-27. Februar 2009, Wien 2009.

[43] T. Erl, “SOA – Principles of Service Design”, Prentice Hall, 2007. ISBN 978-0-13-234482-1.

[44] IBM: WS-BPEL Extenstion for People (BPEL4PEOPLE), 2007, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf

[45] Link S.: Benutzerinteraktion in dienstorientierten Architekturen, Dissertation, 2009, http://digbib.ubka.uni-karlsruhe.de/volltexte/1000012354

www.iariajournals.org

International Journal On Advances in Intelligent Systems
ICAS, ACHI, ICCGI, UBICOMM, ADVCOMP, CENTRIC, GEOProcessing, SEMAPRO, BIOSYSCOM, BIOINFO, BIOTECHNO, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS
issn: 1942-2679

International Journal On Advances in Internet Technology
ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING
issn: 1942-2652

International Journal On Advances in Life Sciences
eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO
issn: 1942-2660

International Journal On Advances in Networks and Services
ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION
issn: 1942-2644

International Journal On Advances in Security
ICQNM, SECURWARE, MESH, DEPEND, INTERNET, CYBERLAWS
issn: 1942-2636

International Journal On Advances in Software
ICSEA, ICCGI, ADVCOMP, GEOProcessing, DBKDA, INTENSIVE, VALID, SIMUL, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS, CLOUD COMPUTING, COMPUTATION TOOLS
issn: 1942-2628

International Journal On Advances in Systems and Measurements
ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL
issn: 1942-261x

International Journal On Advances in Telecommunications
AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA
issn: 1942-2601