Available online at www.sciencedirect.com
SCIENCE DIRECT®
Signal Processing: *Image Communication* 18 (2003) 721–747
SIGNAL PROCESSING: IMAGE COMMUNICATION
www.elsevier.com/locate/image

Bitstream syntax description: a tool for multimedia resource adaptation within MPEG-21

Gabriel Panis[a,*], Andreas Hutter[a], Jörg Heuer[a], Hermann Hellwagner[b], Harald Kosch[b], Christian Timmerer[b], Sylvain Devillers[c,1], Myriam Amielh[c]

[a] *Siemens AG, CT IC 2, Munich 81730, Germany*
[b] *University Klagenfurt, Klagenfurt, Austria*
[c] *Philips Digital Systems Labs, 51, rue Carnot-BP 301, Suresnes 92156, France*

Abstract

In this paper, a generic method is described to allow the adaptation of different multimedia resources by a single, media resource-agnostic processor. This method is based on an XML description of the media resource's bitstream syntax, which can be transformed to reflect the desired adaptation and then be used to generate an adapted version of the bitstream. Based on this concept, two complementary technologies, BSDL[2] and gBS Schema, are presented. The two technologies provide solutions for parsing a bitstream to generate its XML description, for the generic structuring and marking of this description, and the generation of an adapted bitstream using its transformed description. The two technologies can be used as stand-alone tools; however, a joint approach has been developed in order to harmonise the two solutions and exploit their strengths. This paper is focusing on the gBS Schema and the joint BSDL/gBS Schema harmonised approach.

1. Introduction

The information revolution of the last decade has resulted in a phenomenal increase in the quantity of content (including multimedia content) available to an increasing number of users who access it through a plethora of devices and networks. Devices range from mobile phones to high definition TVs and access networks can be as diverse as GSM and broadband cable networks. In order to enhance the user's experience, the delivered content[3] must be tailored (adapted) to the specific device and network capabilities, as well as to the natural environment and user characteristics and preferences.

*Corresponding author. Tel.: +49-89-636-49015; fax: +49-89-636-51115.
E-mail addresses: Gabriel.Panis@mchp.siemens.de (G. Panis), Andreas.Hutter@siemens.com (A. Hutter), Joerg.Heuer@siemens.com (J. Heuer), hellwagn@itec.uni-klu.ac.at (H. Hellwagner), harald@itec.uni-klu.ac.at (H. Kosch), christian.timmerer@itec.uni-klu.ac.at (C. Timmerer), Sylvain.Devillers@imec.be (S. Devillers), Myriam.Amielh@ifrance.com (M. Amielh).
[1] Sylvain Devillers is now with IMEC, Kapeldreef 75, B-3001 Leuven, Belgium.
[2] Part of the work described in this paper (Philips contribution) was funded by the European project OZONE (IST-2000-30026).
[3] *Content* is understood here as a general term, including Web content, multimedia resources, presentations, etc., whereas *resource* is an individually identifiable asset such as a video or audio clip.

0923-5965/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0923-5965(03)00061-4

The need to standardise a framework that facilitates the control of adaptations has been recognised by several standardisation bodies (W3C, WAP, and MPEG). Standards have already been established or are currently under development that provide tools for content description (MPEG-7 [11]) and capability description and negotiation (CC/PP [21], UAProf [23], and MPEG-21 DIA [16]).

But still a major obstacle to apply these frameworks in adaptive multimedia systems is the adaptation of the encoded multimedia resource itself. The adaptation of encoded media bitstreams is, in general, coding format-specific. Given the variety of coding formats available, the implementation of an adaptation processor covering a high percentage thereof is expensive and difficult to maintain with respect to new codecs.

In this paper, we describe an approach that overcomes this obstacle and facilitates the adaptation of encoded multimedia bitstreams in a generic way, by describing the high-level structure of the bitstream with an XML document called *Bitstream Syntax Description* (BSD). This description allows a device, even if it is agnostic of the specific coding format of a binary media resource, to adapt such a media resource.

This novel approach is currently under consideration for Part 7 of the MPEG-21 standard (Digital Item Adaptation) [16]. In this paper, besides the illustration of the BSD, the overall adaptation architecture is described, examples are given, and experimental performance measurements, in terms of file sizes and computational times, are discussed.

The remainder of the paper is organised as follows. In Section 2, the general adaptation approach and possible applications are described; brief background information on content adaptation, XML, and media scalability is also provided. In Section 3, the adaptation process using BSD is explained in detail; a special, but anticipated use case of multi-step adaptation is also analysed in Section 3. The application of the proposed BSD-based adaptation methods on MPEG-4 Visual Elementary Streams and JPEG2000 images is demonstrated in Section 4; performance results are presented as well. Finally, conclusions are given in Section 5.

2. Media resource adaptation framework

2.1. Application scenarios

The overall approach to multimedia resource adaptation as outlined above delineates the capabilities and application areas of the proposed techniques.

The application scenarios are in general limited by the capabilities of the engaged technologies. For the approach described in this paper, on the one hand, versatile and powerful adaptation tools are facilitated even on devices with limited computational and storage capabilities, since the high-level, XML-based Bitstream Syntax Descriptions abstract from the "bits and pieces" of the binary media resources. In addition, since the adaptation processes become independent from the specific coding formats of the media resources, adaptation engines can be located anywhere in a distributed multimedia system: on a server, a client, and conceptually even on network nodes like proxies, gateways, or routers.

On the other hand, producing an adapted version of a media bitstream requires XML and XSLT processing, as described in the remainder of the paper. Also, adaptation according to a given set of constraints involves a well-informed and potentially non-trivial decision process, taking as input capability, preference, and constraint descriptions. These factors rule out low-end devices, due to insufficient computational resources, and make the approach ill-suited for hard real-time adaptation applications, such as on-the-fly manipulation of media bitstreams in an intermediate network device that is, e.g., forced to drop packets from the bitstream due to temporary congestion. Further constraints are represented by the pre-requisite of suitable scalable media resource coding and by the restrictions to editing-style adaptation operations on bitstreams.

Thus, adaptations as considered here, based on bitstream syntax descriptions, are best performed on devices with reasonable computational power and storage capacity, capable of XML and XSLT processing, and applications tolerant to small delays. Two examples are given below; other application scenarios in the same spirit are easily derived.

The evolving MPEG-21 standard aims to provide a multimedia framework that will "*enable transparent and augmented use of multimedia resources across a wide range of networks and devices*" [12]. The MPEG-21 defined use cases [15] span already a wide range of envisaged applications of the approach described in this paper. An obvious application area is the *delivery of digital resources in a heterogeneous environment*, most often referred to as *Universal Multimedia Access* (UMA). Simple transformations can be performed to adapt the resources to e.g.:

- client devices with different audio–visual (e.g., display size), user interaction, communication, computational and storage capabilities;
- user preferences, constraints, and behaviour (e.g., movement, speed); and
- (static) network characteristics and QoS (e.g., bandwidth, delay, jitter, error rate).

These adaptations can be performed on servers, proxy servers, dedicated media gateways, or on reasonably powerful client devices. It is likely that highly dynamic QoS fluctuations, e.g. network contention, need other adaptation mechanisms which can be conducted with low delays on a router. Such cases are not considered here.

Another promising application is *quality-aware multimedia caching* as discussed in [3]. In the scenario reported there the video cache stores videos in several quality layers. If the cache is about to run out of storage, it degrades the quality and thus decreases the size of unpopular videos rather than completely discarding them from the store, according to different strategies. Since this type of quality reduction is a background (i.e., non-real-time) process and the video cache is assumed to be in a computationally powerful machine, the adaptation can be performed using the proposed bitstream syntax description and transformation techniques.

It must be noted that one of the techniques (gBS Schema; cf. Section 3.4) goes beyond pure syntactical modifications of the media resource bitstreams. It introduces a mechanism (a handle called "marker") that allows semantic-driven adaptations to be performed by resource adaptation engines that are agnostic to the resource's coding format and semantics, based on the very same process as used in other bitstream manipulations. The semantic information, expressed using metadata schemes like MPEG-7, can be linked to the corresponding bitstream segments through the use of the marker. The marker concept is described in more detail in Section 3.4.1. Examples of this type of semantic-related adaptations are the removal of violent scenes from a video (cf. Section 4) and the display of specific regions of interest from JPEG2000 images. Applications of this technique are manifold.

2.2. Overall approach

As pointed out in the previous section, the motivation for multimedia resource adaptation is to deliver the best possible content (in terms of information and perceived quality) to the end user given the current set of constraints, e.g. device characteristics and user preferences. The high-level adaptation architecture is depicted in Fig. 1.

Even though this architecture is applicable to the adaptation of any type of content, the focus of this work is on binary multimedia resources. Hence, any reference to "resource" will imply a binary multimedia resource.

The *Adaptation Engine* is a logical unit that can be located on the originating server, an intermediate network device (gateway, router, proxy) or even on the client. This engine comprises two logical modules, the Adaptation Decision Engine and the Resource Adaptation Engine.

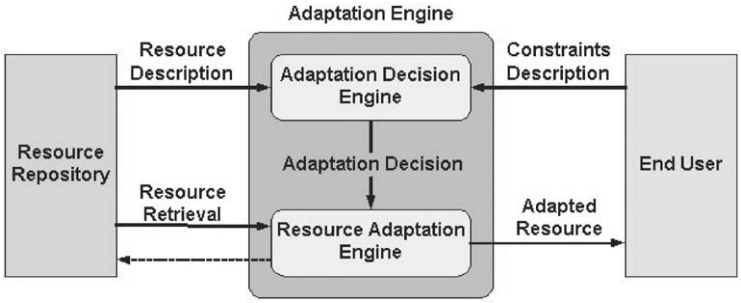

Fig. 1. High-level adaptation architecture.

The *Adaptation Decision Engine* receives the metadata information about the available content (semantics, format, and adaptation options) from the *Resource Repository* and the constraints (terminal and network capabilities, user characteristics and preferences, and natural environment characteristics) from the receiving side. If there are multiple versions of the content pre-stored in the repository and one of these versions matches the constraints, then this version is selected, retrieved, and sent to the end user. However, if the available resource does not match the constraints, but can be adapted, then the Adaptation Decision Engine determines the optimal adaptation for the given constraints and passes this decision to the Resource Adaptation Engine. Then the *Resource Adaptation Engine* retrieves the resource from the repository, applies the selected adaptation and sends the adapted resource to the end user.

The described architecture exhibits the following essential elements of multimedia resource adaptation:

1. *Resource Description*: The *Resource Description* contains all metadata that are related to the adaptation process. This includes metadata that provide information assisting the adaptation decision making and metadata that are used for the resource adaptation. The description should therefore include information on the resource type and semantics (e.g. MPEG-7 descriptors [11]), the available adaptation options and characteristics of the adapted versions of the resource and the bitstream syntax (e.g. MPEG-21 DIA descriptors [16]).
2. *Constraints Description*: The constraints can be grouped into four broad categories: user and natural environment characteristics, terminal capabilities, and network characteristics. The terminal and network constraints will set an upper bound on the resources that can be transmitted over the network and rendered by the terminal. Information like the network's maximum bandwidth, delay and jitter, or the terminal's resolution, buffer size, and processing power, will help the Adaptation Engine determine the optimal version of the resource for the given network and terminal device. As the user is the actual consumer (and judge) of the resource, user related information, including user preferences, user demographics, usage history and natural environment, is equally important in deciding which resource should be delivered to the terminal.
3. *Resource Scalability*: A multimedia resource is defined to be scalable if the data is organised in such a way that, by retrieving parts of the data, an adapted version of the original content can be rendered. Scalability, therefore, can be available in one of the following forms:

   - an individual resource that is encoded with increasing SNR quality, spatial, or temporal resolution;
   - an individual resource, parts of which can be removed (e.g., a scene from a video sequence);
   - a multimedia presentation where one or more individual resources may be removed (e.g., video is removed, only audio is kept from a news broadcast).

4. *Adaptation Decision Engine*: This algorithm takes as input the resource and constraints description and outputs the adaptation that needs to be applied to the resource. The adapted resource should be of optimal quality under the given constraints.
5. *Resource Adaptation Engine*: This module performs the selected adaptation of the resource. It can include various sub-modules, ranging from a simple bitstream parser to a sophisticated transcoder.

2.3. Resource descriptions and adaptations considered

The work described in this paper is within the scope of *Resource Adaptation Engine* as shown in Fig. 1. Specifically, the proposed solution applies mainly to adaptations which involve editing a binary media resource. This editing includes removing bitstream blocks and modifying the values of syntactical elements of the bitstream. "Transmoding" of multimedia resources, i.e. adaptations from one media type to another (e.g., text to speech), and transcoding, i.e. adaptations that require the partial or full decoding of a resource (e.g., MPEG-2 to MPEG-4), require detailed knowledge of the specific coding formats involved and are therefore out of the scope of this work.

Given the variety of available multimedia coding formats and adaptation options, an adaptation engine would be expected to incorporate several codec-specific modules for performing even simple adaptations involving removal or modification of syntactical elements of the bitstream. This approach would yield expensive and difficult to maintain adaptation engines, in particular with respect to upcoming coding formats and multimedia types. As an alternative, we propose a generic tool that allows an adaptation engine to transform a multimedia resource without any prior knowledge of the resource's coding format and therefore without the need to maintain codec-specific modules. With the proposed solution, the resource adaptation engine requires a single processor that uses the XML description of the bitstream syntax to produce an adapted version of the bitstream.

The algorithm behind the adaptation decision engine is an interesting subject of research in itself. MPEG-21 has specified descriptors to assist the decision process (e.g. Choice/Selection in the Digital Item Declaration [13] and AdaptationQoS in DIA [16]), however, the decision-making algorithm itself is implementation specific. It is considered to be beyond the scope of this paper.

2.4. Related work

In the sequel, we give a brief overview of related technologies and standards. On the one hand, several standardisation groups have investigated and standardised capability description schemes which can be used to control bitstream adaptations. IETF has created the Content Negotiation Protocol (CONNEG) [7] that enables protocol-independent negotiation covering both the content and the user. The W3C has built the RDF-based (which can be XML serialised) Composite Capability/Preference Profiles (CC/PP) [21] protocol. The WAP Forum has used the CC/PP to define the User Agent Profile (UAProf) [23] which provides a CC/PP vocabulary describing the characteristics of WAP-enabled devices. MPEG has created the MPEG-7 standard, which specifies an extensive metadata scheme for the description of multimedia resources and user characteristics [11]. Finally, MPEG is currently defining descriptors of user, natural environment, terminal and network characteristics and other resource and adaptation related tools within the MPEG-21 Digital Item Adaptation (DIA) framework [16]. Most of the metadata schemes mentioned above are defined using the World Wide Web Consortium's (W3C) XML Schema Language [22] and expressed using the W3C's Extensible Markup Language (XML) [20].

On the other hand, scalable coding formats supporting bitstream adaptations have been the subject of research for many years. The most widely used type of video scalability is temporal, where predicted frames (B- and P-type) may be dropped to reduce the frame rate and consequently the video's bitrate. For SNR and spatial scalability, a layered approach has been adopted in standardised codecs (MPEG-2, MPEG-4,

H.263), where, on top of a base layer, several enhancement layers may be added to provide an improvement in SNR quality or spatial resolution. A fine granular approach (Fine Granular Scalability, FGS) has been proposed and amended to the MPEG-4 Visual standard [18]. For image compression, the JPEG2000 standard uses wavelet transformation, which provides layered SNR, spatial, and colour scalability [9].

The BSDL approach was originally inspired by XML publishing frameworks such as Cocoon [4] developed by the Apache Software Foundation to publish XML content adapted to the requesting client in a Web context. However, to the best of our knowledge, no work had been done yet to extend these principles to binary multimedia data and to other non-Web-based applications.

Flavor, a language for media representation [5,6], is a technology which has similarities, in some aspects, with the BSDL/gBS Schema approach. Flavor, which stands for *F*ormal *L*anguage for *A*udio–*V*isual *O*bject *R*epresentation, was initially designed as a declarative language with a C++-like syntax to describe the bitstream syntax on a bit-per-bit basis. Its aim is to simplify and speed up the development of software that processes audio–visual bitstreams by automatically generating the required C++ and Java code to parse the data, hence allowing the developer to concentrate on the processing part of the software. Unlike the BSD approach, it does not generate a permanent description of the parsed data, but only an internal memory representation in the form of a collection of C++ or Java objects.

Recently, Flavor was enhanced to support XML features (XFlavor) and tools for generating an XML description of the bitstream syntax, transforming the XML description and regenerating the adapted bitstream [6]. There are, however, some fundamental differences to BSDL/gBS Schema, stemming mainly from the original focus of the two technologies. In XFlavor, the schema does not play part in the bitstream parsing and description generation process, which is done by the ad-hoc C++ or Java code generated from the Flavor description. For the bitstream generation, unlike BSDL/gBS Schema which relies on the XML Schema datatype definition, XFlavor uses proprietary attributes declared in the schema and indicating the binary encoding of the element content. A major drawback of this design is that two elements with the same type (e.g. encoded on two bits) will require the same verbose declaration in the schema, while BSDL/gBS Schema will define the corresponding type once, and use it for all elements. Furthermore, using XML Schema native mechanisms for datatype definition and derivation allows validating the value and optimally binarising it with tools such as MPEG-7 BiM [10]. Lastly, in XFlavor, the complete bitstream data are actually embedded in the XML document, resulting in potentially huge descriptions, while BSDL/gBS Schema uses a specific datatype to point to a data range in the original bitstream when it is too verbose to be included in the description. This is why, unlike Flavor/XFlavor, BSDL is rather a *description* rather than a *representation* language, and can describe the bitstream at a high syntactical level instead of a low-level, bit-per-bit basis.

With respect to the gBS Schema approach in particular, the main differences lie in the generic, universal and codec independent format of the corresponding generic Bitstream Syntax Descriptions (gBSDs), and in the introduction of means to mark individual elements in the gBSD in order to enable and simplify complex and semantic-based adaptations. Such marking is not possible with XFlavor.

The principles of BSDL have been described in [1] and an example application of BSDL for MPEG-4 Visual Texture Coding was introduced in [17]. This paper focuses on the generic Bitstream Syntax Schema (gBS Schema) and the harmonised BSDL/gBS Schema approach, which have not yet been presented in the scientific literature.

3. Media resource adaptation using bitstream syntax descriptions

For most binary media resources, a variety of adapted versions can be retrieved from a single bitstream by performing simple operations such as the removal of data blocks. In order to provide interoperability and to avoid the need for multiple codec-specific adaptation modules, it is desirable that a processor that is

unaware of the specific coding format can be used for this task. For this, a generic approach is taken by providing a method based on XML for describing and manipulating bitstreams.

A binary media resource consists of a structured sequence of binary symbols, this structure being specific to the coding format. A bitstream is defined as the sequence of binary symbols representing this resource. XML is used to describe the high-level structure of a bitstream; the resulting XML document is called a *Bitstream Syntax Description (BSD)*. This description is not meant to replace the original binary data, but acts as an additional layer, similar to metadata. In most cases, it will not describe the bitstream on a bit-per-bit basis, but rather address its *high-level structure*, e.g. how the bitstream is organised in layers or packets of data. For this, a specific mechanism is provided to indicate a segment of data in the original bitstream instead of actually including the data since this would make the description too verbose. Furthermore, the bitstream description is itself scalable, which means it may describe the bitstream at different syntactic layers, e.g. finer or coarser levels of detail, depending on the application. The bitstream description itself must also be adaptable in order to properly reflect bitstream adaptations. Lastly, unlike many metadata schemes such as MPEG-7, it initially does not deal with the semantics of the bitstream, i.e. does not provide metadata information regarding the original object, image, audio or video it represents, but only considers it as a sequence of binary symbols.

With such a description, it is then possible for a resource adaptation engine to transform the XML document and then generate back an adapted bitstream. XSLT (Extensible Stylesheet Language for Transformations) [19], which provides a standardised method for transforming XML documents, can be advantageously used for this purpose. In order to provide full interoperability, it is then necessary that a processor that is not aware of the specific coding format can nevertheless be used to produce a BSD, and/or generate a bitstream from its BSD. For this, it will rely on the information on the specific coding format provided by a schema. In this way, communicating devices (e.g., Resource Repository and Adaptation Engine of Fig. 1) will be able to exchange BSDs and transformation descriptions, for example in the form of XSLT style sheets. Based on the concept described above, two complementary technologies have been developed.

- *BSDL*: A new language based on XML Schema, called *Bitstream Syntax Description Language (BSDL)*, was created by Philips Research, France [1,2]. With this language, it is possible to design specific *Bitstream Syntax Schemas (BS Schemas)* describing the syntax of a particular coding format. These schemas can then be used by a generic processor to automatically parse a bitstream and generate its description, and vice versa.
- *gBS Schema*: BSDL provides means for describing a bitstream syntax which relies on a codec-specific BS Schema. This requires a resource adaptation engine to know the specific schema in order to parse the BSD to generate the corresponding bitstream. In some use cases this is not desired, for instance if the adaptation takes place on remote devices with constrained resources, e.g., in proxies. In these cases, a codec-independent schema is more appropriate. Therefore, a *generic Bitstream Syntax Schema (gBS Schema)* was created by the collaborative parties at the University Klagenfurt, Austria, and Siemens AG, Munich, Germany. The gBS Schema enables the codec-agnostic description of the bitstream and introduces means for describing syntactical bitstream units in a hierarchical fashion and addressing means for efficient bitstream access. Additionally, it provides semantic handles to sections of the bitstream (the so-called "markers") that facilitate semantic-based manipulation or removal of bitstream portions.

The two technologies described in this paper can be applied independently. Each approach carries its own merits and advantages which make it the preferred choice for specific applications. Since their common added value is to be applicable to any multimedia coding format, it is vital to be able to produce such descriptions without having to modify the encoders for each targeted format. BSDL solves this issue by providing a generic way to generate a BSD from a bitstream. For this, a single generic software is required,

exploiting the coding format-specific information provided by the BS Schema. Symmetrically, the same schema is used for producing the adapted bitstream from the transformed description. In the cases where the availability of the specific BS Schema may become an issue, for example on constrained devices, the gBS Schema approach provides a further abstraction level with codec-independent BSDs which are named generic BSDs (gBSD). In this case, the use of a generic, normalised schema along with semantic handles in the gBSDs enables more efficient implementations for both description transformation and bitstream generation processes. Furthermore, the semantic handles provided by the gBS Schema enable semantic and complex adaptations which would otherwise require the use of algorithms that have extensive knowledge of the coding format specifics. The flexibility provided by BSDL (in providing a generic way to generate BSDs from bitstreams) and the processing efficiency provided by gBS Schema (through the use of a normalised schema and marker handles), are thus key to the deployment of this novel multimedia resource adaptation framework based on Bitstream Syntax Descriptions. In an effort to exploit the benefits of both technologies, a joint approach was attempted by defining a harmonised architecture and schema hierarchy. The schema hierarchy is specified in Section 3.1 and the joint architecture in Section 3.2. The BSDL is briefly explained in Section 3.3 and in more detail in [2]. The gBS Schema technology is described in Section 3.4.

3.1. Schema hierarchy

BSDL provides a number of built-in datatypes derived from XML Schema, with specific semantics in the context of generating an adapted bitstream. These datatypes are defined in a schema named *Schema for BSDL-1 Extensions*.

On top of this, BSDL introduces a number of language constructs defining new types of constraints on XML documents. These constructs are added to the XML Schema language by using annotation mechanisms, and are therefore ignored by XML Schema for the validation, but carry specific semantics in the context of description generation. These language extensions are defined in a schema named *Schema for BSDL-2 Extensions*. This schema imports the Schema for BSDL-1 Extensions.

Applications may define Bitstream Syntax Schemas (BS Schemas) for multimedia coding formats. As a BS Schema can define the bitstream syntax for specific codecs and in a level of detail suitable for the specific adaptations, these schemas are application-dependent. It is expected, however, that schemas for common multimedia coding formats (e.g., the various MPEG-4 Visual profiles, JPEG2000) will be defined and be widely used. In order to use the built-in datatypes (BSDL-1) and language constructs (BSDL-2) of BSDL, these codec-specific schemas need to import one or both of the schemas mentioned above. Note that for the description generation, the language constructs (i.e., the Schema for BSDL-2 Extensions) are required, whereas for bitstream generation only the datatypes (i.e., the Schema for BSDL-1 Extensions) are needed. The gBS Schema proposed in this paper uses BSDL datatypes and therefore imports the Schema for BSDL-1 Extensions.

A BS Description is an instantiation of a codec-specific BS Schema, whereas a gBS Description is an instantiation of the gBS Schema.

The different inclusion mechanisms between the schemas explained above are illustrated in Fig. 2.

3.2. Adaptation architecture

Fig. 3 depicts the architecture of one resource adaptation step. The architecture comprises the original scalable bitstream and its BSD, one (or more) description transformations, the adapted bitstream and its BSD and two additional processes, the description generation and bitstream generation processes. The description may be a gBS Schema-valid gBS Description (gBSD), or a BS Schema-valid BS Description (BSD). The output produced in one adaptation step is an adapted bitstream and possibly a transformed

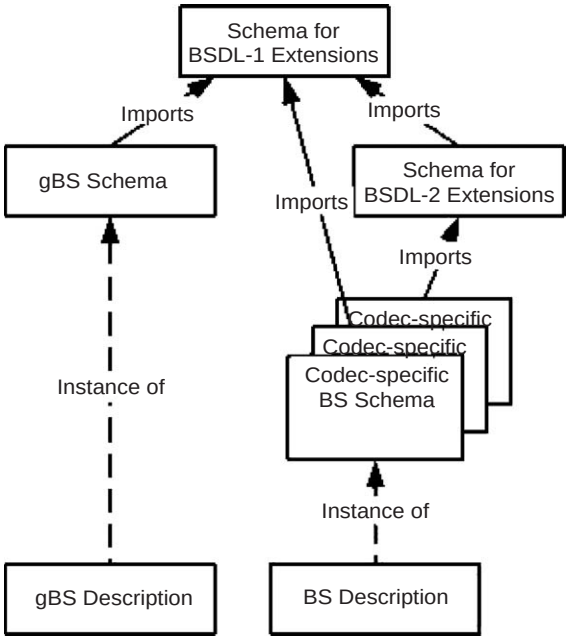

Fig. 2. BSD schema hierarchy.

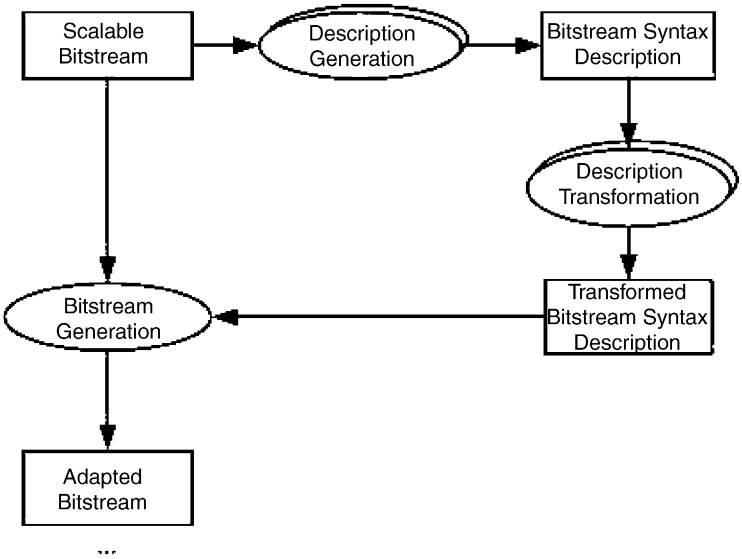

Fig. 3. Adaptation architecture.

BSD that correctly references the new bitstream. This output enables the application of multiple successive adaptations. A walkthrough of the process is given below. Note that Fig. 3 focuses on the resource adaptation engine, therefore other elements of the adaptation framework, like the decision making engine and the constraints description, are not shown here.

The BSDs or gBSDs may be generated in a proprietary way, e.g. during encoding. Alternatively, the BSDL-defined *Description Generator (BintoBSD)*, described in Section 3.3, may be used to parse a bitstream described by a BS Schema and generate its description.

The bitstream and its description are subject to the adaptation. The adaptation decision engine is assumed to determine the optimal adaptation for the media resource, given the constraints as described in Section 2 and depicted in the high-level architecture in Fig. 1. Based on that decision, if the resource is not pre-stored but needs to be derived by adapting an existing resource, then one (or several) description transformation(s) is (are) selected to be applied to the input description. The result of these transformations is a transformed description which is the basis for the generation of the adapted bitstream. The generation of the adapted bitstream is achieved using the bitstream generation process shown in Fig. 3 and specified in more detail later in this section. This *Bitstream Generator* might be a so-called *BSDtoBin* (processing a BSD) or a *gBSDtoBin* parser (utilising a gBSD), as explained below. It should be noted that due to data dropping, some address references (e.g. of elements after a removed data block) in the Transformed BSD might not point to the correct sections of the Adapted Bitstream and therefore need to be adjusted. The address correction process can either be incorporated into the Bistream Generation process or be applied on the Transformed BSD using an XSLT (not shown in Fig. 3).

3.3. Bitstream Syntax Description Language (BSDL)

This section briefly describes how the Bitstream Syntax Description Language (BSDL), introduced above, is built on top of XML Schema. A more detailed specification may be found in [2,14,16]. The work presented here incorporates the harmonisation with the gBS Schema approach along with the new developments required for the description generation.

The primary role of a schema is to validate an XML document, i.e. to check that it follows a set of constraints on its structure and datatypes. On top of this, BSDL adds a new functionality, which is to specify how to generate an adapted bitstream from its description and vice versa. For this, a number of restrictions and extensions covering structural and datatype aspects of XML Schema are required. This is done in a compatible way such that a BS Schema still conforms to XML Schema and can be used to validate the BS Description. In the following, we name *BSDtoBin Parser* the generic software parsing the BSD and generating the bitstream, and *BintoBSD Parser* the software performing the reverse operation. For their operation, both parsers use the BS Schema specifying the syntax of the coding format. Note that proprietary software may be used instead of the BintoBSD parser to generate the BSD as long as the result conforms to the BSDL specification.

In the XML Schema terminology used below, *component* is the generic term for the building blocks that comprise the abstract data model of the schema; `xsd:choice`, `xsd:sequence`, `xsd:element` or `xsd:complexType` are examples of schema components.

For BSDtoBin, the BSDL uses a set of XML Schema components and defines built-in datatypes with specific semantics in the context of the bitstream generation. On the other hand, some XML Schema components should be ignored by the BSDtoBin parser because they have no meaning in the BSDL context. This set of extensions and restrictions is referred to as BSDL-1, and the syntax of the extensions is specified in the *Schema for BSDL-1 Extensions* introduced above.

The BintoBSD parser requires more advanced features on top of BSDL-1, including some new language constructs such as variables and conditional statements. In order to remain compatible with XML Schema, these extensions are introduced as application-specific annotations, namely by using the `xsd:appinfo` schema component and adding attributes with non-schema namespaces. This set of extensions is referred to as BSDL-2, where BSDL-1 is a subset of BSDL-2. Their syntax is specified in the *Schema for BSDL-2 Extensions* introduced above. In the following, BSDL refers to both BSDL-1 and BSDL-2.

The fundamental principle of BSDL is that the successive parameters of the bitstream are included in the XML elements of the description. In the example shown in Document 1, a two-byte parameter with the hexadecimal value FF91 is embedded in the description by the element ⟨Marker⟩FF91⟨/Marker⟩. Other parameters may be written in a decimal format, depending on their semantics and the application. Furthermore, a type must be assigned to each element in order to specify the encoding scheme of its content. While the function of a datatype definition in XML Schema is to constrain the *value* of an element or attribute content for validation, the datatype specifies its *binary encoding* for BSDL. BSDL datatypes are therefore restricted to those for which a binary representation may be defined. For example, the XML Schema `xsd:integer` datatype represents the mathematical concept for an unbounded integer. No implicit binary representation may be assigned to this type, which is therefore excluded from BSDL. On the other hand, `xsd:int` is derived from `xsd:integer` by restricting its value space to the values that may be represented on four bytes. BSDL includes this type and assigns a binary representation on four bytes. According to this principle, BSDL uses a defined list of primitive XML Schema datatypes, including integer types with a finite number of bytes (`xsd:long`, `xsd:int`, `xsd:short`, `xsd:byte`) and their unsigned derivatives (`xsd:unsignedLong`, ...), floating point numbers (`xsd:float` and `xsd:double`) and binary data (`xsd:hexBinary` and `xsd:base64`). Furthermore, BSDL introduces a new built-in datatype named `bsdl:byteRange` to point to a data segment in the original resource, used when the size of the segment would make the description too verbose. In this case, the description does not actually include the data, but a reference locating the data in the original bitstream.

An example of a BSD and a BS Schema is given in Documents 1 and 2, respectively.

Document 1: Example of a BSD of a JPEG2000 bitstream

```
⟨?xml version="1.0"?⟩
⟨Codestream xmlns="JP2" xmlns:jp2="JP2" xsi:schemaLocation="JP2 JP2.xsd"⟩
  ⟨MainHeader⟩
    ⟨!--and so on... --⟩
  ⟨/MainHeader⟩
  ⟨Bitstream⟩
    ⟨Packet⟩
      ⟨SOP⟩
        ⟨Marker⟩FF91⟨/Marker⟩
        ⟨LMarker⟩4⟨/LMarker⟩
        ⟨Nsop⟩0⟨/Nsop⟩
      ⟨/SOP⟩
      ⟨PacketData⟩155 242⟨/PacketData⟩
    ⟨/Packet⟩
    ⟨!--and so on... --⟩
  ⟨/Bitstream⟩
⟨/Codestream⟩
```

Document 2: Example of a BS Schema of a JPEG2000 bitstream

```
⟨xsd:schema targetNamespace="JP2"
  xmlns:bsdl-1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"⟩
  ⟨xsd:import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS" schemaLocation="BSDL-1.
xsd"/⟩
```

```
  〈xsd:element name="SOP"〉
    〈xsd:complexType〉
      〈xsd:sequence〉
        〈xsd:element name="Marker" type="xsd:hexBinary"/〉
        〈xsd:element name=‘‘LMarker’’ type=‘‘xsd:unsignedShort’’/〉
        〈xsd:element name="Nsop" type="xsd:unsignedShort"/〉
      〈/xsd:sequence〉
    〈/xsd:complexType〉
  〈/xsd:element〉
  〈xsd:element name="PacketData" type="bsdl-1:byteRange"/〉
〈!-- and so on... --〉
〈/xsd:schema〉
```

It is important to note that, while a coding format may be specified as a "flat" structure, i.e. as a sequence of binary symbols, these symbols may be grouped in a hierarchical manner in the description, since the inherent tree structure of the XML document has no impact on the bitstream generation process. Furthermore, the choice of tag names is also fully transparent. The design of a BS Schema for a given coding format is therefore not unique and is fully application dependent. For example, Document 6 shows the high-level structure of an MPEG-4 Visual Elementary Stream BSD, where Visual Object Planes (VOPs) are described as a single segment of data and not further detailed, since the adaptation consists of simply removing the B-VOPs. Another application may use finer scalability modes of MPEG-4 Visual and hence require a finer level of details within the VOPs. Depending on the application and typically on the scalability mode used, different levels of details may be required and thus different BS Schemas may be designed for the same coding format.

3.4. Generic Bitstream Syntax Schema (gBS Schema)

The *generic BS Schema* is targeting enhanced binary resource adaptations. The gBS Schema imports the Schema for BSDL-1 Extensions in order to use the datatypes specified in BSDL-1. The syntax of a gBSD is generic and codec-independent, therefore enabling the processing of binary resources without the need of codec-specific schemas. Additionally, the gBS Schema provides the following functionality:

- Semantically meaningful marking of described syntactical elements through the use of a "marker" handle.
- Description of a bitstream in a hierarchical fashion that allows grouping of bitstream elements for efficient, hierarchical adaptations.
- Flexible addressing scheme to support various application requirements and random accessing of the bitstream.

The gBSD approach is based on the same general concept as BSDL. However, the unique functionality that the gBS Schema provides, facilitates complex adaptations. Specifically, the "marking" of elements and the hierarchical structure of the description, can greatly simplify sophisticated bitstream manipulations (e.g., remove an SNR layer from a JPEG2000 bitstream when the progression order is resolution) and semantic-related adaptations (e.g., remove violent scenes from a video sequence). Furthermore, the gBSDtoBin process does not require an application or codec-specific schema, since all the necessary information to regenerate the bitstream is included in the gBSD and the semantics of the gBS Schema elements. This is particularly desired when the adaptation takes place on remote, constraint devices, since it saves bandwidth space and computational resources.

3.4.1. gBS Schema

The gBS Schema is fully specified in the MPEG-21 DIA standard [16]. We restrict ourselves here to introduce the most important elements of the gBS Schema approach. Examples of gBSDs illustrating the concepts described in the following are given in Section 4.

The gBS Schema defines three types of elements:

- *Header*: Contains information about the classification scheme used and sets default values required for resolving the address references in the gBSD. Specifically, the header defines the alias and URI of the classification scheme where the element names used in the gBSD are specified. These names are codec-specific names of the elements represented in the gBSD, thus enabling codec aware adaptation engines to navigate and use the gBSD. For interoperability and extensibility purposes, these names can be stored in a classification scheme which is identified in the Header by its URI. Additionally, the Header defines the default values used for addressing (the URI of the bitstream location, addressing mode (absolute, consecutive, offset), units used (bit or byte)). Only the root and the gBSDUnit may contain a Header. The values of the parameters defined in the Header apply to all descendants of the element that contains the Header. These parameters may be overridden by another Header further down in the description tree or addressing attributes in individual gBSDUnits or Parameters.
- *gBSDUnit*: Represents a section of the bitstream. The size of a gBSDUnit is not limited, so it could be used to represent a single bitstream syntactical element, a group of elements, or a block of data. As the gBSDUnit does not provide the actual value of the element it represents, just its location in the bitstream, it should not be used to represent elements the values of which might need to be modified during adaptation. A gBSDUnit may include one Header and zero or more Parameters and gBSDUnits. A gBSDUnit includes a start and length attribute to point to the section of the bitstream it describes. In addition, the optional addressing attributes open the possibility to override inherited addressing attributes. If these attributes are instantiated, they apply to the gBSDUnit and are inherited to its descendants but not its siblings.
- *Parameter*: Describes a syntactical element of the bitstream. The Parameter can be used to describe elements of the bitstream the values of which might need to be changed when adapting the bitstream (cf. Section 4.1.2). Unlike the gBSDUnit, the Parameter provides the actual value and datatype of the bitstream element, therefore enabling the alteration of the numerical values of bitstream elements in addition to removal. The datatype is necessary for the bitstream generation process (gBSDtoBin) to correctly encode the syntactical element in the bitstream. The Parameter also includes optional addressing attributes, similarly to the gBSDUnit, to allow overriding the inherited values.

A schema has been created to provide a library of basic datatypes for the value of the Parameter. This library includes some of the XML Schema built-in datatypes and others encountered often in existing multimedia coding formats. The library can be easily extended to include new datatypes.

The gBSDUnits and Parameters may carry a "marker" attribute. This attribute marks sections of the bitstream, therefore providing a handle which could link an external description of possible adaptations, using MPEG-7 and MPEG-21 description tools, to the specific bitstream elements that need to be either removed or modified to achieve the corresponding adaptation. The semantic intelligence lies in the MPEG-7 and MPEG-21 metadata as well as their linkage to the marker. The marker value and the resource adaptation engine are therefore semantically agnostic. This handle can be extremely useful in the case of complicated adaptations (e.g., select a specific region in an image) or semantic-related adaptations (e.g., remove violent scenes from a video, when it will be watched by children). The adaptation engine can then easily apply semantic adaptations that would otherwise require extensive knowledge about the resource's coding format and semantic analysis of the resource.

The gBS Schema provides a flexible addressing scheme which allows specifying and overriding the bitstream location, the addressing unit, and addressing mode. The location of the bitstream is expressed in

the form of a URI, the unit can be either bit or byte and the addressing mode can be absolute, offset, or consecutive. *Absolute* addressing mode means that the element in the gBSD carries a start attribute which gives the absolute position of the element relative to the beginning of the bitstream. *Offset* is similar to absolute; however, the start attribute gives the position relative to the position of the parent element in the gBSD. *Consecutive* means that the position of the element is the next bit or byte after the end of the previous element in the gBSD tree. If the element is the first child, then the previous element is the parent element.

3.4.2. Adaptation process using the gBSD

The architecture shown in Fig. 3 is used to generate a gBSD, adapt it, and then regenerate the adapted bitstream. The process is explained step-by-step in the sequel.

3.4.2.1. gBSD generation. The gBSD may be generated in any proprietary way. Due to the hierarchical structure of the description and the marking of gBSDUnits and Parameters for various adaptations, the production of a gBSD is application-specific. Some anticipated methods for gBSD production include the generation of a gBSD during media encoding by an enhanced encoder and the generation by a codec-specific bitstream parser. The gBSD may also be generated using the BintoBSD parser (cf. Section 3.3), which supports the Schema for BSDL-2 extensions. In the latter approach, a generic bitstream processor (BintoBSD) parses the bitstream using a bitstream-specific schema and an optional formatting XSLT style sheet to generate the gBSD. The XSLT style sheet which is used to transform the BSD generated by the BintoBSD parser into a valid gBSD, is a possible way of giving the gBSD the format that is suitable for the specific adaptations. The XSLT can specify the hierarchical structure and the level of detail of the gBSD and add the marker handles to description elements that could be removed or modified during the adaptation process. This generation method was used to validate and evaluate the gBS Schema approach as described in Section 4.

3.4.2.2. gBSD transformation. An XSLT style sheet may be used to transform the gBSD. The style sheet corresponds to the adaptation option selected by the adaptation decision engine and is input to the gBS Schema-based resource adaptation engine possibly along with input parameters to the XSLT. The changes performed on the description reflect the alterations needed to adapt the bitstream. Some of the most common changes include the removal of syntactical elements (Parameters) or portions of the bitstream (gBSDUnits) and the replacement of the values of bitstream syntactical elements (Parameters). Any other more complicated bitstream operations (e.g. data rearrangement) that can be achieved by transforming the gBSD are also possible.

3.4.2.3. Bitstream generation using gBSD. The gBSDtoBin processor hierarchically parses a gBSD and constructs the corresponding bitstream. The gBSDtoBin uses the addressing information in the description to retrieve the bitstream portions represented by gBSDUnits and it uses the value and datatype of the Parameters to correctly encode these syntactical elements into the bitstream. The gBSDtoBin processor is also responsible to update the addressing attributes that become incorrect after the removal of bitstream elements.

3.4.2.4. Multi-step adaptation. There are cases when a bitstream has to be adapted several times. For example, a JPEG2000 image might need to have its spatial resolution reduced to allow rendering on a PDA screen but it might additionally have to be scaled down in SNR quality to avoid overloading the slow wireless GPRS connection. If the adaptations are to be performed on the same device, e.g. the server, then all adaptations may be applied to the description before it is input to the gBSDtoBin to generate the final adapted bitstream. However, when the adaptations do not take place on the same device, then the bitstream

needs to be generated every time it is transferred from one device to the other, along with its description. It is then required that the description correctly describes the bitstream it references. Therefore, some addressing references in the gBSD might need to be updated during the gBSDtoBin process. A description "correctly describes" a certain bitstream when the gBSDUnits and Parameters of the description refer to actual parts of the bitstream and the addressing (start and length) attributes of the elements point to the correct parts of the bitstream. There is, however, no check on whether the actual values of the syntactical elements in the bitstream are correct with respect to the coding format.

4. Applications of BSDL/gBS Schema: adaptation of JPEG2000 images and MPEG-4 visual elementary streams

Images and videos represent two of the most commonly used multimedia types. JPEG2000 [9] and MPEG-4 Visual [8] are well-known standardised coding formats. Both formats offer various types of scalability and therefore provide suitable test cases for the application of the BSDL/gBS Schema adaptation method. Additionally, an application of BSDL to the MPEG-4 Visual Texture Coding, which also provides scalable features, can be found in [17].

4.1. JPEG2000 images

JPEG2000 [9] is an inherently scalable image coding format. Each JPEG2000 image may be encoded with several types of scalability, including SNR, resolution and colour component. The colour component scalability means that, depending on the colour space, the image may be stripped of one or even all colour components. In order to scale down a JPEG2000 image using any of the three types of scalability, certain packets of the image bitstream need to be removed and some of the header markers need to be modified. The complete adaptation process for a JPEG2000-encoded bitstream named "city.jp2" is described here, first using a BSDL-based approach and then using the gBS Schema method. The bitstream "city.jp2" is an image taken from the MPEG-7 test content set and encoded using a JPEG2000 [9] conformant encoder.

4.1.1. BSDL-based adaptation method

The application of BSDL to a JPEG2000 image is briefly explained here, a more extensive description may be found in [1]. Document 3 shows a fragment of the BSD "city.xml" obtained by running the BintoBSD parser on the "city.jp2" bitstream.

Document 3: Fragment of the "city.xml" BSD

```
〈?xml version="1.0"?〉
〈Codestream xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="JP2" xsi:schemaLocation="JP2 JP2.xsd"〉
  〈MainHeader〉
    〈SOC〉
      〈Marker〉FF4F〈/Marker〉
    〈/SOC〉
    〈SIZ〉
      〈Marker〉FF51〈/Marker〉
      〈LMarker〉47〈/Lmarker〉
      〈Rsiz〉0〈/Rsiz〉
```

```
      〈Xsiz〉768〈/Xsiz〉
      〈Ysiz〉512〈/Ysiz〉
      〈!-- and so on --〉
      〈Csiz〉3〈/Csiz〉
      〈!-- and so on --〉
    〈/SIZ〉
    〈otherSegmentMarkers〉
      〈!-- and so on --〉
      〈QCD〉
        〈Marker〉FF5C〈/Marker〉
        〈LMarker〉35〈/LMarker〉
        〈MarkerData〉69 33〈/MarkerData〉
      〈/QCD〉
      〈!-- and so on --〉
    〈/otherSegmentMarkers〉
  〈/MainHeader〉
  〈Tile〉
        〈!-- and so on --〉
      〈Bitstream〉
        〈Packet〉
          〈SOP〉
            〈Marker〉FF91〈/Marker〉
            〈LMarker〉4〈/Lmarker〉
            〈Nsop〉0〈/Nsop〉
          〈/SOP〉
          〈PacketData〉155 435〈/PacketData〉
        〈/Packet〉
      〈!-- and so on --〉
    〈/Bitstream〉
  〈/Tile〉
  〈EOC〉
    〈Marker〉FFD9〈/Marker〉
  〈/EOC〉
〈/Codestream〉
```

In short, a JPEG2000 bitstream consists of a main header, itself made of several "marker segments", followed by one or several "tile(s)" containing the payload, i.e. the packets of actual image data resulting from the encoding method. The marker segments give a number of parameters about the image and about the compression method, such as its dimension or the wavelet transformation used. In the JPEG2000 specification, they are referred to as mnemonics such as SIZ for the parameters relating to the image size. In the BSD, the XML elements describing the marker segments or their parameters are named after these mnemonics.

Note that for explanation purposes, all the parameters of the SIZ marker segment have been detailed by the BintoBSD parser (but not all are shown in Document 3), notably the Csiz parameter indicating the number of colour components that will be used by the adaptation process. On the other hand, the QCD marker segment has been left as a "black box" indicating a data segment of 33 bytes with no further details on the included parameters. This example shows that, depending on the application, it is possible to detail some parts of the bitstream and leave some other parts non-detailed.

For simplicity, three different XSLT style sheets have been written for adaptation, corresponding to the three scalability modes used in this application. Document 4 shows the style sheet modifying the SNR quality of the image. The number of data packets is a function of the number of colour components and of decomposition levels, which are indicated by the relevant parameters in the BSD and the number of output layers, which is given as an input parameter to the style sheet. The transformation thus consists of modifying the value of the latter parameter in the BSD and removing the relevant number of packets. An adapted bitstream is then re-generated from the transformed BSD by applying the BSDtoBin parser.

Document 4: XSLT style sheet modifying the number of SNR quality layers of a JPEG2000 bitstream

```
〈?xml version="1.0"?〉
〈xsl:stylesheet xmlns:j="JP2" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"〉
  〈xsl:output method="xml" indent="yes"/〉
  〈!-- Parameter: output number of quality layers, default value = 1 --〉
  〈xsl:param name="nLayersOut"〉1〈/xsl:param〉
  〈!-- Calculate number of output packets --〉
  〈xsl:variable name="nResOut"〉
    〈xsl:value-of select= "//j:COD/j:SPcod/j:nDecompLevels"/〉
  〈/xsl:variable〉
  〈xsl:variable name="nCompOut"〉
    〈xsl:value-of select= "//j:SIZ/j:Csiz"/〉
  〈/xsl:variable〉
  〈xsl:variable name="nSOPOut"〉
    〈xsl:value-of select= "($nResOut + 1) * $nLayersOut * $nCompOut"/〉
  〈/xsl:variable〉
  〈xsl:template name="tplAll" match="@*|node()"〉
    〈xsl:copy〉
      〈xsl:apply-templates select="@*|node()"/〉
    〈/xsl:copy〉
  〈/xsl:template〉
  〈xsl:template match="j:numLayers"〉
    〈!-- Set numLayers value to nLayersOut --〉
    〈xsl:copy〉
      〈xsl:value-of select="$nLayersOut"/〉
    〈/xsl:copy〉
  〈/xsl:template〉
  〈xsl:template match="j:Bitstream"〉
    〈xsl:copy〉
      〈!-- do not copy packets with position greater than nSOPOut --〉
      〈xsl:apply-templates select="j:Packet [position() <= $nSOPOut]"/〉
    〈/xsl:copy〉
  〈/xsl:template〉
〈/xsl:stylesheet〉
```

4.1.2. gBS Schema-based adaptation method

The same JPEG2000 bitstream "city.jp2" was parsed using BintoBSD and an XSLT style sheet was applied to produce the corresponding gBSD. The style sheet is used to specify the desired level of detail,

hierarchy, and also to mark Parameters and gBSDunits in order to facilitate the adaptation process. A fragment of the gBSD "city.xml" is shown in Document 5.

Document 5: Fragment of the "city.xml" gBSD

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈dia:DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:dt="urn:mpeg:mpeg21:2003:01-DIA-BasicDatatypes-NS"
  xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS gBSSchema.xsd urn:mpeg:mpeg21:
2003:01-DIA-BasicDatatypes-NS BasicTypes.xsd"〉
  〈dia:Description xsi:type="gBSDType"〉
    〈Header〉
      〈ClassificationAlias alias="J2K" href="urn:jpeg:jpeg2000:cs:syntacticalLabels"/〉
      〈DefaultValues unit="byte" addressMode="Absolute" globalAddress="Content/city.
jp2"/〉
    〈/Header〉
    〈gBSDUnit syntacticalLabel=":J2K:MainHeader" start="0" length="135"〉
      〈gBSDUnit syntacticalLabel=":J2K:SOC" start="0" length="2"/〉
      〈gBSDUnit syntacticalLabel=":J2K:SIZ" start="2" length="49"〉
        〈Header〉
          〈DefaultValues addressMode="Consecutive"/〉
        〈/Header〉
        〈Parameter name=":J2K:Marker" length="2"〉
          〈Value xsi:type="xsd:hexBinary"〉FF51〈/Value〉
        〈/Parameter〉
        〈Parameter name=":J2K:Lsiz" length="2"〉
          〈Value xsi:type="xsd:unsignedShort"〉47〈/Value〉
        〈/Parameter〉
        〈Parameter name=":J2K:Rsiz" length="2"〉
          〈Value xsi:type="xsd:unsignedShort"〉0〈/Value〉
        〈/Parameter〉
        〈Parameter name=":J2K:Xsiz" length="4" marker="R"〉
          〈Value xsi:type="xsd:unsignedInt"〉768〈/Value〉
        〈/Parameter〉
        〈Parameter name=":J2K:Ysiz" length="4" marker="R"〉
          〈Value xsi:type="xsd:unsignedInt"〉512〈/Value〉
        〈/Parameter〉
        〈!-- ... and so on (other parameters) ..... --〉
        〈gBSDUnit syntacticalLabel=":J2K:Comp_siz" length="3" marker="C0"/〉
        〈gBSDUnit syntacticalLabel=":J2K:Comp_siz" length="3" marker="C1"/〉
        〈gBSDUnit syntacticalLabel=":J2K:Comp_siz" length="3" marker="C2"/〉
        〈!-- ... and so on ..... --〉
      〈/gBSDUnit〉
      〈!-- ... and so on ..... --〉
```

```
      〈gBSDUnit syntacticalLabel=":J2K:COM" start="102" length="33"/〉
    〈/gBSDUnit〉
    〈!-- ... and so on ..... --〉
  〈/dia:Description〉
〈/dia:DIA〉
```

This description demonstrates most of the gBS Schema features. The Header, child of the gBSDType Description element, defines the classification alias and its URI as well as the default addressing values which include the location of the bitstream, the mode and unit used for addressing. Subsequently, the hierarchical description of the bitstream syntax follows.

Parameters represent syntactical elements of the bitstream, the values of which might be changed during certain adaptations. Therefore, the numerical value and datatype of these elements are provided through the Value element and the `xsi:type` attribute. For example, the Parameters with names Xsiz and Ysiz represent the corresponding bitstream elements which specify the spatial resolution of the image. These Parameters need to be adjusted accordingly when the bitstream is scaled down to a lower resolution.

gBSDUnits are used to represent segments of data in the bitstream, as is exemplified by the gBSDUnits with syntacticalLabel J2K:Comp_siz. gBSDUnits are also used to group other gBSDUnits and Parameters, as in the case of the J2K:MainHeader unit.

Note that the gBSDUnit with syntacticalLabel J2K:SIZ has a Header child element that defines a different addressMode attribute. This attribute overrides the addressMode attribute inherited by the gBSDUnit from its ancestors, i.e. the root element. Therefore, the addressMode for all descendants of the J2K:SIZ gBSDUnit is Consecutive.

The names of syntactical elements are also provided (syntacticalLabel in gBSDUnit, name in Parameter). Both of these attributes are optional. The gBSD creator may choose to include them so that the description can be used by adaptation engines that are aware of the encoding format and can adapt the bitstream by using the gBSD as a reference index in order to locate the syntactical elements in the bitstream. These names can be listed in a Classification Scheme, specified in the Header in terms of its URI and the alias used in the gBSD.

Lastly, note that the last three gBSDUnits include a "marker" attribute. The value of the attribute itself is not significant. What is important is the handle that the marker provides to bitstream segments. This handle can provide a link between a description of adaptation options (not given here, typically used by an adaptation decision engine) and the gBSD transformation process. In the example shown, each of the marked gBSDUnits is related to a colour component. Since in this specific image the colour space is YUV, C0 represents luminance, and C1 and C2 represent chrominance. The value of the marker is matched to an adaptation option listed in the metadata describing the content. The marker is also used by the XSLT style sheet to easily identify the elements that need to be changed for a specific adaptation. By marking a description in such a way, possibly offline on a powerful server, the complexity of the adaptation itself may be significantly reduced, which is desirable when the adaptation is expected to take place on a device with limited resources (e.g. network node).

The gBSD was transformed using two XSLT style sheets. The first one removed two of the five quality layers. The second style sheet reduced the spatial resolution from 768×512 to 192×128 pixels. The resulting description "city_adapt.xml" includes the adjusted values for the Parameters and only the gBSDUnits that are needed to produce the adapted version of the image with three quality layers and a resolution of 192×128 pixels.

The adapted image "city_adapt.jp2" was reproduced from the "city_adapt.xml" description and the original image "city.jp2" using the gBSDtoBin process.

The "city_adapt.xml" description was transformed once again using an XSLT style sheet that removes the chrominance to yield a greyscale version of the image. The resulting description "city_adapt2.xml" and

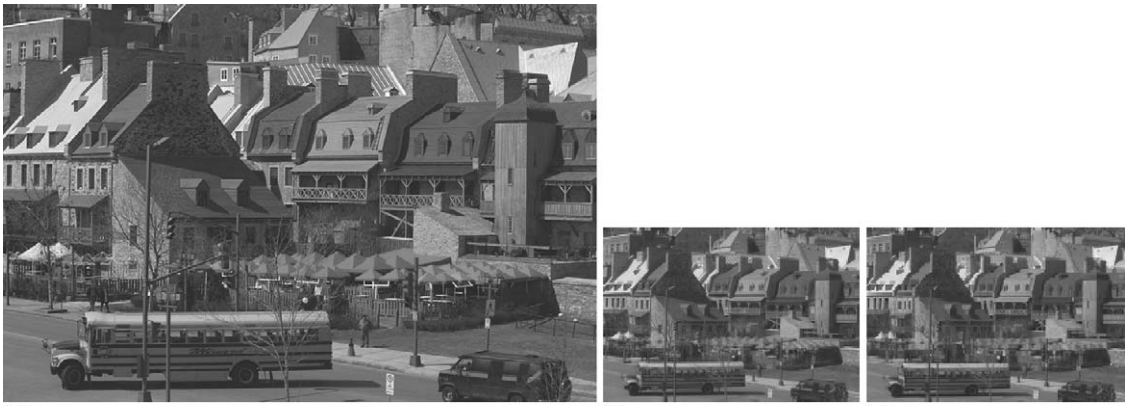

Fig. 4. (a) city.jp2: original image; (b) city_adapt.jp2: SNR, Resolution reduced; (c) city_adapt2 (greyscale): SNR, Resolution reduced, colour components removed.

the ''city_adapt.jp2'' bitstream were then parsed by gBSDtoBin to produce the final image ''city_adapt2.jp2'' which is a greyscale image with three quality layers and a resolution of 192×128 pixels.

The three images (city.jp2, city_adapt.jp2, city_adapt2.jp2) are shown in Fig. 4. Images (a) and (b) are in colour, whereas (c) is greyscale.

4.2. MPEG-4 Visual Elementary Streams

MPEG-4 video coding is based on the well-known motion-compensated hybrid DCT coding scheme. Therefore, I-, P- and B-frames are subject to adaptations required, e.g., for temporal scalability. Every MPEG-4 Visual Elementary Stream (MPEG-4 Visual ES) comprises several Video Object Planes (VOPs) classified as different types such as I-, P- and B-VOPs, among others. Each ES is headed by some configuration or header information depending on the ES entry point [8]. The complete hierarchy of an MPEG-4 Visual ES is illustrated in detail in [18]. A famous movie trailer, ''Star Wars Episode II–Attack of the Clones'', was selected to demonstrate the BSDL and gBS Schema approaches.

4.2.1. BSDL-based adaptation method

Document 6 shows a fragment of the BSD obtained after applying the BintoBSD parser to an MPEG-4 Visual stream, Advanced Simple Profile. Note that we detail the minimum of information required by the adaptation, namely the parameter vop_coding_type indicating the VOP type (0, 1 and 2 for I-, P- and B-VOPs, respectively).

Document 6: Fragment of an MPEG-4 Visual ES BSD

```
〈?xml version="1.0"?〉
〈Bitstream xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="MPEG4"
  xmlns:mp4="MPEG4" xsi:schemaLocation="MPEG4 MPEG4.xsd" xml:base="Content/starwars.
cmp"〉
    〈VO〉
      〈video_object_start_code〉00000101〈/video_object_start_code〉
    〈/VO〉
```

```
    〈VOL〉
      〈video_object_layer_start_code〉00000120〈/video_object_layer_start_code〉
      〈VOL_data〉8 18〈/VOL_data〉
    〈/VOL〉
    〈VOP〉
        〈vop_start_code〉000001B6〈/vop_start_code〉
        〈vop_coding_type〉0〈/vop_coding_type〉
        〈Stuffing〉16〈/Stuffing〉
        〈Payload〉31 2872〈/Payload〉
    〈/VOP〉
    〈VOP〉
        〈vop_start_code〉000001B6〈/vop_start_code〉
        〈vop_coding_type〉1〈/vop_coding_type〉
        〈Stuffing〉17〈/Stuffing〉
        〈Payload〉2908 59〈/Payload〉
    〈/VOP〉
    〈VOP〉
        〈vop_start_code〉000001B6〈/vop_start_code〉
        〈vop_coding_type〉2〈/vop_coding_type〉
        〈Stuffing〉16〈/Stuffing〉
        〈Payload〉2972 12〈/Payload〉
    〈/VOP〉
    〈!-- and so on --〉
〈/Bitstream〉
```

The BSD is then transformed by means of the style sheet shown in Document 7. Note that the transformation is very simple, since it only consists of removing the B-VOPs. The adapted bitstream is then re-generated by the BSDtoBin parser.

Document 7 XSLT style sheet for removing B-VOPs

```
〈?xml version="1.0"?〉
〈xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:mp4="MPEG4"
    version="1.0"〉
  〈xsl:output method="xml" indent="yes"/〉
  〈!-- Match all: default template --〉
  〈xsl:template name="tplAll" match="@*|node()"〉
    〈xsl:copy〉〈xsl:apply-templates select="@*|node()"/〉〈/xsl:copy〉
  〈/xsl:template〉
  〈!-- Match B_VOP - Overrides tplAll --〉
  〈xsl:template name="tplB_VOP" match="mp4:VOP[mp4:vop_coding_type=2]"/〉
〈/xsl:stylesheet〉
```

4.2.2. gBS Schema-based adaptation method

First, the generic bitstream processor (BintoBSD) is used to generate an intermediate XML representation of the bitstream, a BSD. Second, an XSLT transformation produces a gBSD which is depicted in Document 8. The additional XSLT style sheet adds syntactical labels and the hierarchical

structure, including a "marker" handle designating violent scenes. A description of the syntactical gBS Schema elements can be found in Section 3.4.1.

Document 8: Fragment of an MPEG-4 Visual ES gBSD

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈dia:DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
  xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS gBSSchema.xsd"〉
  〈dia:Description xsi:type="gBSDType"〉
    〈Header〉
      〈ClassificationAlias alias="MV4" href="urn:mpeg:mpeg4:video:cs:
syntacticalLabels"/〉
      〈DefaultValues addressUnit="byte" addressMode="Absolute" globalAddressInfo=
"Content/starwars.cmp"/〉
    〈/Header〉
    〈gBSDUnit syntacticalLabel=":MV4:VO" start="0" length="4"/〉
    〈gBSDUnit syntacticalLabel=":MV4:VOL" start="4" length="22"/〉
    〈gBSDUnit start="26" length="99983" marker="violent-5"〉
      〈gBSDUnit syntacticalLabel=":MV4:I_VOP" start="26" length="2877"/〉
      〈gBSDUnit syntacticalLabel=":MV4:P_VOP" start="2903" length="64"/〉
      〈gBSDUnit syntacticalLabel=":MV4:B_VOP" start="2967" length="17"/〉
      〈gBSDUnit syntacticalLabel=":MV4:B_VOP" start="2984" length="16"/〉
      〈!--... and so on ... --〉
      〈gBSDUnit syntacticalLabel=":MV4:P_VOP" start="98296" length="1713"/〉
    〈/gBSDUnit〉
    〈gBSDUnit start="100009" length="68022" marker="violent-5"〉
      〈gBSDUnit syntacticalLabel=":MV4:I_VOP" start="100009" length="1825"/〉
      〈gBSDUnit syntacticalLabel=":MV4:P_VOP" start="101834" length="1780"/〉
      〈!--... and so on ... --〉
      〈gBSDUnit syntacticalLabel=":MV4:I_VOP" start="166802" length="1229"/〉
    〈/gBSDUnit〉
    〈!--... and so on ... --〉
  〈/dia:Description〉
〈/dia:DIA〉
```

A so-called codec-aware adaptation engine that has knowledge about the classification scheme (CS) used to name the syntacticalLabels in the gBSD (in this case an MPEG-4 Visual Elementary Stream CS, specifying MPEG-4 Visual Elementary Stream element names, e.g. I_VOP, P_VOP and B_VOP) carries out the first adaptation. Assuming that the end user's terminal does not have the capability for decoding B-VOPs, the corresponding gBSDUnits are removed during this adaptation step. The second constraint retrieved from the end user in this example comprises information about user characteristics such as age of the beholder. Due to that fact, the adaptation decision engine decides to remove all gBSDUnits marked as "violent". An appropriate XSLT style sheet performs each transformation followed by a bitstream generation process which generates an adapted version of the bitstream after each description transformation. Each transformation must ensure to provide a correct bitstream description and

corresponding bitstream for the subsequent adaptation step. This might include additional (XSLT) transformations of the bitstream description after generation of the adapted bitstream—mostly some necessary adjustments of the address information. Specifically the start and length attributes of parents or elements after a removed segment, might need to be updated.

Document 9 exemplifies an XSLT style sheet for the first adaptation step. It comprises two XSLT templates where the first is the default template which copies nodes from the source tree to the result tree without changes. The second template removes all gBSDUnits with syntactical label ":MV4:B_VOP". Therefore, the template body is empty and nothing is copied from the source tree to the result tree.

Document 9: XSLT style sheet for removing B-VOPs only

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈xsl:stylesheet version="1.0"
  xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"〉
  〈xsl:output method="xml" indent="yes"/〉
  〈!--Match all: default template --〉
  〈xsl:template name="tplAll" match="@*|node()"〉
    〈xsl:copy〉
      〈xsl:apply-templates select="@*|node()"/〉
    〈/xsl:copy〉
  〈/xsl:template〉
  〈!-- Match gBSDUnit - Overrides tplAll --〉
  〈xsl:template name="tplB_VOP" match="gbsd:gBSDUnit[@syntacticalLabel=':MV4:B_VOP']"〉
    〈!-- Do nothing --〉
  〈/xsl:template〉
〈/xsl:stylesheet〉
```

In practice, however, XSLT style sheets become more complicated if modifying or removing is carried out in combination with address information adjustments. A fragment of the transformed gBSD is illustrated in Document 10 which is the result of applying the XSLT style sheet from Document 9 on the gBSD from Document 8. Removed gBSDUnits have been replaced with corresponding remarks.

Document 10: Fragment of a transformed MPEG-4 Visual ES gBSD

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈dia:DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
  xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS gBSSchema.xsd"〉
    〈dia:Description xsi:type="gBSDType"〉
      〈Header〉
        〈ClassificationAlias alias="MV4"
          href="urn:mpeg:mpeg4:video:cs:syntacticalLabels"/〉
        〈DefaultValues addressUnit="byte" addressMode="Absolute"
          globalAddressInfo="Content/starwars.cmp"/〉
```

```
      〈/Header〉
      〈gBSDUnit syntacticalLabel=":MV4:VO" start="0" length="4"/〉
      〈gBSDUnit syntacticalLabel=":MV4:VOL" start="4" length="22"/〉
      〈gBSDUnit start="26" length="99983" marker="violent-5"〉
        〈gBSDUnit syntacticalLabel=":MV4:I_VOP" start="26" length="2877"/〉
        〈gBSDUnit syntacticalLabel=":MV4:P_VOP" start="2903" length="64"/〉
        〈!-- :MV4:B_VOP removed --〉
        〈!-- :MV4:B_VOP removed --〉
        〈!--... and so on ... --〉
        〈gBSDUnit syntacticalLabel=":MV4:P_VOP" start="98296" length="1713"/〉
      〈/gBSDUnit〉
    〈gBSDUnit start="100009" length="68022" marker="violent-5"〉
      〈gBSDUnit syntacticalLabel=":MV4:I_VOP" start="100009" length="1825"/〉
      〈gBSDUnit syntacticalLabel=":MV4:P_VOP" start="101834" length="1780"/〉
      〈!--... and so on ... --〉
      〈gBSDUnit syntacticalLabel=":MV4:I_VOP" start="166802" length="1229"/〉
    〈/gBSDUnit〉
    〈!--... and so on ... --〉
  〈/dia:Description〉
〈/dia:DIA〉
```

4.3. Performance measurement

The performance of the BSDL/gBS Schema approach for multimedia resource adaptation was measured for the adaptation sequences described above. The performance was measured with respect to the description size (relative to the resource size) and the processing time for the adaptation of the resource.

4.3.1. Description size

Table 1 summarises the file sizes required for the bitstream descriptions (XML as well as binarised) that were used in the experiments. Table 2 shows the same data after one adaptation step. The data for the XML bitstream descriptions were compressed using the MPEG-7 BiM (Binary Model) codec, with the new string compression functionality ZLib turned on. This functionality is expected to be included in version 2 of MPEG-7 Part 1: Systems [10]. The BSDs could not be compressed with the current implementation of MPEG-7 BiM[4]. It is expected that newer versions of the encoder (starting from version 2) could successfully compress BSDs.

The difference in the XML/Bitstream ratio between the MPEG-4 and JPEG2000 examples is due to the difference in the level of description detail of the two cases. For the adaptation of the MPEG-4 sequence, individual syntactical elements are not modified and therefore not detailed in the descriptions. In the case of JPEG2000 images, however, several elements need to be modified for each type of adaptation and are therefore represented in the descriptions in detail. Note that the size of the description, relative to the resource, is significantly reduced through binarisation.

4.3.2. Processing time

Table 3 shows the processing time (in seconds) measured for each step of one adaptation process. These steps are: BintoBSD (generate BSD from bitstream), BSDtogBSD (XSLT applied to BSD to transform it to

[4] In version 1 of the BiM specification, some XML Schema constructs not used in MPEG-7 are not supported. Accordingly the reference software implementation of BiM does not compress these constructs. It is expected that in version 2 which is currently being standardised, all XML schema constructs will be supported.

Table 1
File sizes of original BSDs and gBSDs for "starwars" and "city" streams

Filename/method	Bitstream size (bytes)	XML descr. size (bytes)	BiM size (bytes)	XML/bitstream	BiM/bitstream
Starwars					
BSD	2,515,959	283,032	—	11.25%	—
gBSD	2,515,959	133,968	12,076	5.32%	0.48%
City					
BSD	240,369	18,088	—	7.53%	—
gBSD	240,369	39,113	3,725	16.27%	1.55%

Table 2
File sizes of adapted BSDs and gBSDs for "starwars" and "city" streams

Filename/ method	Adapted bitstream size (bytes)	Adapted XML description (bytes)	BiM size (bytes)	XML/ bitstream	BiM/bitstream
Starwars					
BSD	1,127,426	94,660	—	8.40%	—
gBSD	1,127,426	44,487	3966	3.95%	0.35%
City					
BSD	151,808	12,917	—	8.51%	—
gBSD	151,808	26,026	2536	17.14%	1.67%

Table 3
Processing times for each step of the adaptation process

Processing step	Proc. time (s)	
	Starwars	City
BintoBSD	5.0	1.6
BSDtogBSD	4.9	2.2
BintogBSD	9.9	3.8
XSLT (BSD)	1.8	1.3
XSLT (gBSD)	1.7	1.6
BSDtoBin	3.6	1.3
gBSDtoBin	6.4	1.4

gBSD), BintogBSD (the aggregated time of the previous two processes), XSLT (application of transformation style sheet to BSD and gBSD), BSDtoBin (parsing of BSD to generate the bitstream) and gBSDtoBin (parsing of gBSD to generate the bitstream). The machine used for this experiment had the following hardware characteristics: Pentium 4, 2 GHz, 512 KB cache, 1 GByte RDRAM.

The description generation is the most time and resource consuming process. This process, however, is expected to take place on the resource provider's side (most probably on a server) where computing resources will be amply available. In the case of gBSD, in particular, it is anticipated that if the description is properly prepared using the "marker" handle, it can make complex adaptations possible even on low-end devices.

The BSD/gBSD transformation and bitstream generation times vary depending on the length and level of detail in the description. It can be seen that the process for the JPEG2000 image is long relative to the

process for the MPEG-4 sequence, considering that the MPEG-4 bitstream is significantly larger. The reason is the presence of several fine grained syntactical elements in the JPEG2000 bitstream, the values of which are modified during the adaptation and need to be individually encoded into the bitstream (instead of just being copied from the original bitstream).

5. Conclusions

Resource adaptation plays an increasingly important role in the delivery of resources that meet the customer's preferences and constraints. XML technologies provide the tools that facilitate the adaptation of Web content and they are being widely adopted by the multimedia metadata community. In this paper, we presented a concept that uses XML technologies for generic, resource and coding format independent adaptation of multimedia binary resources. The concept is based on the creation of an XML description of the bitstream syntax. This description may be transformed and then be used to generate the adapted version of the bitstream. Two complementary technologies that are based on this concept were described. These technologies can be used independently or complementarily. The flexibility of BSDL allows to automatically produce BSDs from bitstreams in a generic way. On constrained devices, the gBS Schema approach allows efficient implementations of description transformations and adapted bitstream generations. The combined approach described in this paper provides an elegant, flexible and practical solution for adapting multimedia resources.

References

[1] M. Amielh, S. Devillers, Multimedia Content Adaptation with XML, Proceedings of Eighth International Conference on Multimedia Modeling (MMM'2001), Amsterdam, The Netherlands, November 5–7, 2001, pp. 127–145.
[2] M. Amielh, S. Devillers, Bitstream Syntax Description Language: Application of XML-Schema to Multimedia Content, Proceedings of the 11th International World Wide Web Conference (WWW 2002), Honolulu, May 6–11, 2002.
[3] L. Böszörményi, H. Hellwagner, H. Kosch, M. Libsie, S. Podlipnig, Metadata Driven Adaptation in the ADMITS Project, EURASIP Image Commun. J. (Special Issue on Multimedia Adaptation), Fall, 2003.
[4] Cocoon Framework, http://xml.apache.org/cocoon/.
[5] A. Eleftheriadis, Flavor: A Language for Media Representation, ACM Multimedia Conference, Seattle, WA, November 1997, pp. 1–9.
[6] D. Hong, A. Eleftheriadis, XFlavor: Bridging Bits and Objects in Media Representation, IEEE International Conference on Multimedia and Expo (ICME2002), Lausanne, Switzerland, August 2002.
[7] IETF: Protocol-independent Content Negotiation Framework, RFC 2703, September 1999.
[8] ISO/IEC 14496-2:2001: Information Technology—Generic Coding of Audio–Visual Objects—Part 2: Visual.
[9] ISO/IEC 15444-1:2000: Information Technology—JPEG 2000 Image Coding System—Part 1: Core Coding System.
[10] ISO/IEC 15938-1:2002: Information Technology—Multimedia Content Description Interface—Part 1: Systems.
[11] ISO/IEC 15938-5:2002: Information Technology—Multimedia Content Description Interface—Part 5: Multimedia Description Schemes.
[12] ISO/IEC 21000-1:2001: Information Technology—Multimedia Framework—Part 1: Vision, Technologies and Strategy.
[13] ISO/IEC 21000-2:2002: Information Technology—Multimedia Framework—Part 2: Digital Item Declaration.
[14] ISO/IEC JTC 1/SC 29/WG 11: MPEG-21 Digital Item Adaptation Application Model v6.0. Document N5846, Trondheim, Norway, July 2003.
[15] ISO/IEC JTC 1/SC 29/WG 11: MPEG-21 Use Case Scenario Document, Document N4991, Klagenfurt, Austria, July 2002.
[16] ISO/IEC JTC 1/SC 29/WG 11: MPEG-21 Digital Item Adaptation Final Committee Draft. Document N5845, Trondheim, Norway, July 2003, http://www.chiariglione.org/mpeg/working_documents.htm#MPEG-21.
[17] R. Osorio, S. Devillers, E. Delfosse, M. Amielh, G. Lafruit, Bitstream Syntax Description Language for 3D MPEG-4 View-dependent Texture Streaming, IEEE 2002 International Conference on Image Processing (ICIP 2002), Rochester, September 22–25, 2002.
[18] F. Pereira, T. Ebrahimi (Eds.), The MPEG-4 Book, Prentice-Hall PTR, Englewood Cliffs, NJ, 2002.

[19] W3C: XSL Transformations (XSLT), Version 1.0, W3C Recommendation, November 16, 1999, http://www.w3.org/TR/xslt.
[20] W3C: Extensible Markup Language (XML) 1.0, 2nd Edition, W3C Recommendation, October 6, 2000, http://www.w3.org/TR/2000/REC-xml-20001006.
[21] W3C: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. W3C Working Draft, March 15, 2001, http://www.w3.org/TR/CCPP-struct-vocab/.
[22] W3C: XML-Schema. Part 0: Primer. Part 1: Structures. Part 2: Datatypes. W3C Recommendations, May 2, 2001, http://www.w3.org/XML/Schema.
[23] WAP Forum: User Agent Profile Specification, Wireless Application Protocol Forum, November 10, 1999, http://www.wapforum.org/.