BITSTREAM SYNTAX DESCRIPTION-BASED ADAPTATION IN STREAMING AND CONSTRAINED ENVIRONMENTS

Sylvain Devillers[*], Christian Timmerer[‡], Jörg Heuer[†], and Hermann Hellwagner[‡]

[*] France Telecom R&D, Issy les Moulineaux, France
sdevillers.ext@rd.francetelecom.com

[‡] Department of Information Technology (ITEC), Klagenfurt, Austria
{christian.timmerer, hermann.hellwagner}@itec.uni-klu.ac.at

[†] Siemens AG, Corporate Technology, IC 2, Munich, Germany
Joerg.Heuer@siemens.com

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

Department of Information Technology (ITEC)
Klagenfurt University
Technical Report No. TR/ITEC/05/1.05
June 2005

Bitstream Syntax Description-Based Adaptation in Streaming and Constrained Environments

Sylvain Devillers, Christian Timmerer, Jörg Heuer, and Hermann Hellwagner, *Member, IEEE*

***Abstract*—The seamless access to rich multimedia content on any device and over any network, usually known as Universal Multimedia Access, requires interoperable description tools and adaptation techniques to be developed. To address the latter issue, MPEG-21 Digital Item Adaptation (DIA) introduces the Bitstream Syntax Description (BSD) framework, which provides tools for adapting multimedia content in a generic (i.e., coding format independent) way. The basic idea is to use the eXtensible Markup Language (XML) to describe the high-level structure of a binary media bitstream, to transform its description [e.g., by means of eXtensible Stylesheet Language Transformations (XSLT)], and to construct the adapted media bitstream from the transformed description. This paper presents how this basic BSD framework, initially developed for nonstreamed content and suffering from inherent limitations and high memory consumption of XML-related technologies such as XSLT, can be advanced and efficiently implemented in a streaming environment and on resource-constrained devices. Two different attempts to solve the inherent problems are described. The first approach proposes an architecture based on the streamed processing of Simple Application Programming Interface for XML (SAX) events and adopts Streaming Transformations for XML (STX) as an alternative to XSLT, whereas the second approach breaks a BSD up into well-formed fragments called process units that can be processed individually by a standard XSLT processor. The current status of our work, as well as directions for future research, are given.**

***Index Terms*—BSD, MPEG-21, MPEG-7 BiM, multimedia content adaptation, SAX, STX, transcoding, Universal Multimedia Access, XML, XSLT.**

I. INTRODUCTION

THE LAST DECADE has brought about an enormous growth of digital multimedia content on the Internet. This also pertains to the diversity of media formats and the richness of the contents. For instance, just MPEG-4 specifies object-based coding techniques, scene descriptions tools and a variety of specific codecs ranging from audio and video to graphics and synthetic content, providing interactivity and scalability. Additionally, various proprietary formats from industry, as well as from open source projects are in use.

At the same time, the diversity of devices via which access to, and interaction with, multimedia content is desired, has grown significantly. The spectrum ranges from workstations, personal computers and set-top boxes to portable devices like Web pads, personal digital assistants, and even mobile phones. Low-end devices are constrained in various ways, such as in terms of display size, color capabilities, input options, processing power, memory resources, and power supply.

Finally, a wide spectrum of networks for the transmission of the multimedia contents has emerged as well. In particular, wireless networks have proliferated in recent years, enabling users to access the Web and multimedia content from different locations, in different contexts, and with varying connectivity characteristics.

These three trends as well as additional factors, like specific preferences or impairments of the users, represent major barriers toward Universal Multimedia Access (UMA)—the ability to access and consume multimedia content on any device, anywhere, anytime, in a seamless manner and customized to the usage context and user preferences. [1] gives an overview of what MPEG-21 Part 7: Digital Item Adaptation (DIA) specifies to overcome those barriers. In particular, the Bitstream Syntax Description (BSD) framework [2], [3] incorporates a number of concepts for adapting multimedia content in a coding format independent way. The latter is a step toward bridging the gap between the diversity of media formats on the one hand and the variety of networks and devices on the other hand. In other words, this framework replaces potentially compute-intensive and specific transcoding operations, e.g., to adapt the spatial resolution of a video, by two "generic" steps: 1) transformation of the BSD characterizing the media bitstream (an eXtensible Markup Language (XML) document) via eXtensible Stylesheet Language Transformations (XSLT) [4], for instance, and 2) generation of the adapted bitstream using the transformed BSD. This relieves nodes in the media delivery and adaptation chain from having to implement potentially manifold transcoding operations.

However, in order for the BSD-based adaptation approach to become feasible for practical use, these very two steps are required to be implemented efficiently. This requirement represents a major challenge on resource-constrained devices such as the portables listed above. There are two major limitations that have to be considered here.

The first aspect is that these devices usually do not have enough memory resources to fully download and play a video, for instance, but that streaming and piecewise or progressive rendering techniques have to be applied. Moreover, such devices may be even so short of memory that a BSD cannot to be stored as a complete XML document. This requires that the BSD has to be fragmented, deliv-

Manuscript received May 5, 2004; revised September 16, 2004. This work was supported in part by the European research projects ISIS (IST-2001-37253) and DANAE (IST-1-507113). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Anthony Vetro.

S. Devillers was with IMEC, Interuniversity MicroElectronics Center, Leuven, Belgium. He is now with France Telecom, R&D Division, Issy les Moulineaux, France (e-mail: Sylvain.Devillers@free.fr).

C. Timmerer and H. Hellwagner are with the University of Klagenfurt, Klagenfurt, Austria (e-mail: christian.timmerer@itec.uni-klu.ac.at; hermann.hellwagner@itec.uni-klu.ac.at).

J. Heuer is with Siemens Corporate Technology, Munich, Germany (e-mail: Joerg.Heuer@siemens.com).

Digital Object Identifier 10.1109/TMM.2005.846794

ered, and processed in pieces as well, potentially in synchrony with the media bitstream.

The second limitation is the usually poor computational power of such devices. This restriction requires that BSD-based adaptation has to be computationally cheap in order to be executable on constrained portable devices.

These aspects, i.e., the need for a low-complexity, memory-efficient, and streaming-compliant BSD framework implementation, are closely interrelated and require new concepts to be developed. The usual implementation approach, as e.g., taken in [3], relies on a legacy XSLT processor, which usually loads the entire BSD into memory. This approach has to be abandoned in a streaming or/and constrained environment.

This paper describes two different attempts to cope with these resource constraints. The first approach proposes an architecture based on the streamed processing of SAX events and adopts STX, a streaming transformation language for XML, as an alternative to XSLT, whereas the second approach breaks the BSD up into well-formed fragments called process units (PUs) that can be processed by a standard XSLT processor.

The remainder of the paper is organized as follows. Section II describes the background of this work, i.e., reviews the basic BSD framework. Section III presents application scenarios where BSD-based adaptation can be beneficial. In Section IV, the two basic approaches, the STX/SAX-based and the PU-based adaptation, are introduced. Section V outlines directions for future research. Conclusions are given in Section VI.

II. Background

MPEG-21 DIA specifies a generic framework that facilitates media bitstream adaptation based on its BSD. A BSD is a well-formed XML document describing the high-level structure of a bitstream. It is important to note that the aim of the BSD is not to describe the bitstream on a bit-by-bit basis but rather to record its organization in terms of packets, headers, or layers of data. The adaptation of the actual bitstream is performed by first transforming the BSD according to the usage environment using standardized XML transformation languages; subsequently a generic software module is used to generate the adapted version of the bitstream. To this end, this generic processor named BSDtoBin utilizes the information conveyed by another XML document named Bitstream Syntax Schema (BS Schema), which specifies the constraints on the structure and datatypes of the BSDs describing bitstreams of a given coding format. A new language named Bitstream Syntax Description Language (BSDL) was developed, presented in [5] and [6] and standardized in MPEG-21 DIA [7]. It is built on top of XML Schema [8] with a set of syntactical extensions and restrictions. The BS Schema may also be used by a second generic processor named BintoBSD to parse a bitstream and generate its BSD.

MPEG-21 DIA does not mandate any specific BS Schema for well-known formats such as JPEG2000 or MPEG-4 Visual Simple Profile. Rather, it standardizes an abstract, generic BS Schema (gBS Schema), applicable to any coding format. A description conforming to this schema is called a generic BSD (gBSD). Due to its abstract nature, the gBS Schema cannot

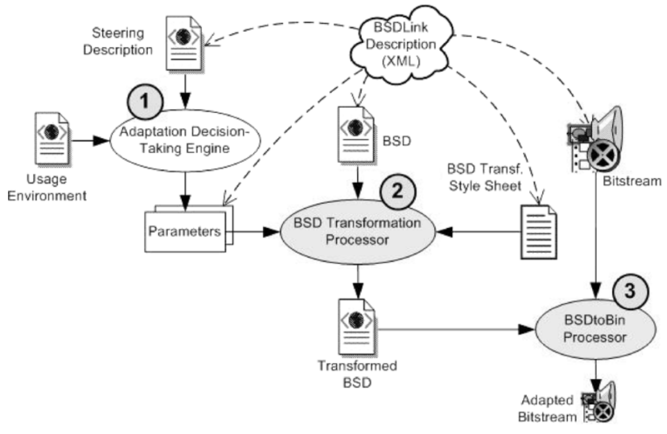


Fig. 1. MPEG-21 DIA descriptions used for multimedia content adaptation.

be used by the BintoBSD processor to generate the BSD. On the other hand, the main advantage of this complementary approach is that it is possible to use a dedicated and more efficient processor named gBSDtoBin to generate the bitstream from its gBSD, which does not need to load and parse the gBS Schema since it is standardized and can be hard-coded in the software. The use of gBSD can thus be profitable in environments such as proxies or thin clients where computing resources are limited. The application scenarios and architectures presented in this paper are applicable to BSDs and gBSDs unless explicitly specified. We refer readers interested in further details of the BSD framework to [2], [3], [5], and [6], as well as to other papers appearing in this special issue, specifically [1].

In order to link the BSDs to other DIA descriptions used in the content negotiation, DIA also specifies a tool, namely the BSDLink description, which provides references to information assets enabling interoperable bitstream adaptation based on BSDs (Fig. 1). One reference identifies the so-called Steering Description which is metadata associated with a bitstream supporting the adaptation decision taking process [(1) in Fig. 1], e.g., this description conveys possible parameter settings for the adaptation engine such that the constraints given by the Usage Environment (e.g., terminal and network capabilities) can be satisfied. The results of the decision-taking process are parameters used to configure the target of another reference, the BSD Transformation Style Sheet. Based on this style sheet, a BSD transformation processor (2) transforms the BSD by applying remove as well as minor editing operations, such as modifications of element values. The resulting Transformed BSD provides the input for the generic BSDtoBin Processor (3), which generates the adapted bitstream based on the input bitstream.

The left part of Fig. 1, i.e., how to perform adaptation decision taking based on constraints of the Digital Item provider and consumer, is intensely discussed in [9] and therefore not covered in this paper.

Please note that Fig. 1 describes only one adaptation step. In practice, however, several adaptation steps could be distributed over different devices along the delivery path as discussed in Section III-B.

This work has been initially inspired by the XML/XSLT publishing framework where the structure of the content is separated from its presentation and is adapted to the requesting client in a Web context. However, to the best of our knowledge, no work has been done so far to extend these principles to binary multimedia data and to other non Web-based applications. Some work that is distantly related to BSD, e.g., Flavor [10] and XFlavor [11], is discussed in [3].

III. Application Scenarios for BSD-Based Adaptation

This section describes several application scenarios demonstrating the flexibility of the BSD-based multimedia content adaptation framework. Firstly, a classical scenario, where adaptation is performed on a server, is given. Then, it is shown how using the BSD framework on the client can reduce the amount of transferred data. Lastly, other scenarios where adaptation takes place on an intermediate network node such as a proxy or a gateway are described, raising new technical issues.

A. *Adaptation on Server or Client*

A simple and classical scenario for multimedia content adaptation is the one where a Web page displays a thumbnail image with a hyperlink to a larger resolution image. With a nonscalable format such as the sequential mode of JPEG, two versions of the same image at the required resolutions need to be stored on the server. In contrast, the use of a scalable format such as JPEG2000 [12] allows the dynamic creation of a smaller resolution image from a single encoded source image. Using the BSD-based adaptation approach, the source image is stored along with its BSD, the corresponding BS Schema, and the relevant transformation style sheet. The two latter documents are shared by all JPEG2000 images on the server. Upon request from the client, the server applies the style sheet to the BSD and dynamically generates the adapted bitstream from the resulting transformed description. Thus, the adaptation is performed on the server in the compressed domain and independently from the media's coding format, rather than during decoding and rendering at the client.

In this scenario, however, when the user successively retrieves two resolutions of the same image, the data corresponding to the lower frequencies, although found in both resolutions, are downloaded twice. JPIP, a standard Internet Protocol presented in [13], allows selectively requesting and retrieving image data from the server and hence avoiding the unnecessary duplication of download. However, this solution is specific to JPEG2000. To enable selective download in a generic way with the BSD-based approach, we propose a modified architecture where the server sends the transformed BSD to the client, which then has to generate the adapted bitstream on its own. Each time the BSD points to a segment of data in the source image which is still located on the server, the client performs a partial HTTP request for the relevant byte range. When the client retrieves a new version of the image, the segment of data corresponding to the lower frequencies and already contained in the previous version, does not need to be downloaded a second time, provided that it has been stored in a local cache. The traffic overhead due to the transmission of the BSD and due to multiple partial HTTP requests is amortized if multiple versions of the same content are successively retrieved. For example, this is the case for an image representing a map where the user can zoom in and out through a wide range of resolutions and at any location on the map. This can be achieved by encoding the image in the JPEG2000 format with several resolutions and the use of the so-called precincts, which spatially divide the image to provide spatial scalability. The low-resolution image is downloaded once; then by zooming at a given location in the image, the client will download the relevant packets of data corresponding to the new required resolution and location and build the new image by using the low frequency data already available. Similarly, when the user requests a shifted view at the same resolution, the client will not need to download the data corresponding to the overlapping region already stored in the local cache. The use of a scalable format along with the BSD-based adaptation framework therefore supports easy retrieval of multiple image versions from a single source image.

B. *Distributed Adaptation*

In the second use case, an end user is interested in a certain movie, described by his/her user preferences, which he/she would like to consume on his/her mobile computer. The DIA specification provides means for describing the usage environment including the user preferences, the terminal capabilities, and the network characteristics.

Assume that another end user is interested in consuming the same movie but within another usage environment, i.e., on a different device and/or network. Optimal network utilization is achieved by transmitting the movie to an intermediate node, i.e., a proxy, between the provider and the consumers such that the movie satisfies a set of usage environment constraints common to all consumers. On reception of the movie, the proxy caches, adapts, and forwards the movie in order to satisfy the usage environment constraints of the specific users who requested this particular movie. The adaptation according to the common usage environment constraints is performed at the server and the adaptation for individual users is conducted at the proxy. Hence, this kind of adaptation is referred to as distributed adaptation [14].

This kind of proxy incorporates functionality that is different to what is done by a traditional proxy. This proxy not only intercepts and services client requests, but also oversees the usage environment downstream toward the clients and reshapes their various usage environment constraints to one or more common sets. These common constraints are then attached to the request for the multimedia contents from the actual server (or yet another proxy).

Fig. 2 depicts the concept of distributed adaptation within a heterogeneous environment.

In a conventional approach, each intermediate node would have to implement and maintain a separate (encoding, decoding, and adaptation) module for each coding format encountered. However, when following the BSD-based adaptation approach, only one adaptation module needs to be implemented which adheres to the processing model as defined in the DIA specification. This adaptation module can be used to adapt content based on all coding formats according to the various user, terminal, and network characteristics.

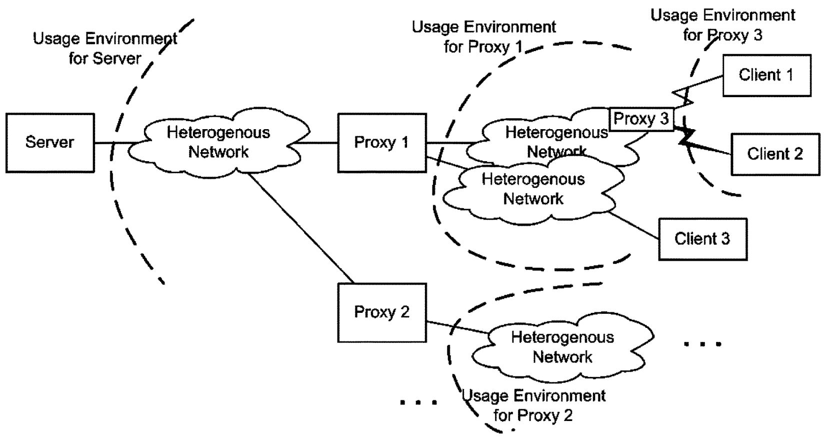


Fig. 2. Concept of distributed adaptation.

IV. Architecture for Streaming and Constrained Environments

The previous section shows application scenarios where the BSD framework may encounter specific challenges resulting from constraints of the environment. In particular, limited computing and memory resources of the involved nodes (proxy or a thin client) as well as streaming the media to terminal nodes may pose problems. The first main challenge pertains to the complexity and memory requirements of the processors involved and particularly the BSD transformation when an off-the-shelf XSLT processor is to be used. Such a case, i.e., standard XSLT, requires that the complete BSD (which is potentially memory consuming) is available to the transformation module (which is potentially compute intensive). The second main limitation pertains to the type of media being adapted: in case of a streamed video or audio, the BSD will become available to the transformation only piecewise, with the pieces preferably being properly synchronized with the bitstream segments they describe. However, since XSLT requires loading the complete input XML document prior to transforming it, the adaptation cannot take place until the streaming session is over. Both issues are interrelated and may be considered as two aspects of the global issue of adapting and improving the BSD framework to constrained and/or streaming environments.

Two complementary solutions are proposed below to tackle this issue. The first solution proposes a software architecture using SAX and a new transformation language named STX as an alternative to XSLT. The second solution proposes to segment the input BSD into successive PUs that can be independently processed by the XSLT processor, hence minimizing the memory requirements. Note that both solutions are applicable to BSDs and gBSDs to some extent.

A. *SAX-Based Bitstream Adaptation*

The left part of Fig. 3 depicts the principal software architecture initially developed for simple scenarios such as the adaptation of nonstreamed content on a server, as described in Section III-A. The upper part in the diagram is optional, depending on whether a BSD is already available or not. A BSD may be generated at encoding time or later by a dedicated software module, in which cases the generation is coding format-dependent. Conversely, the generic BintoBSD processor allows generating the BSD in a coding format-independent way by using the information conveyed by the BS Schema. The remainder of the architecture is as follows: the BSD is transformed by an XSLT processor and the resulting transformed BSD is then processed by the BSDtoBin processor to generate the bitstream. Note that although it is duplicated in the figure for the sake of clarity, the same BS Schema is normally used by the BintoBSD and BSDtoBin processors. Furthermore, the input bitstream intervenes in the bitstream generation (BSDtoBin), which is not depicted in the figure.

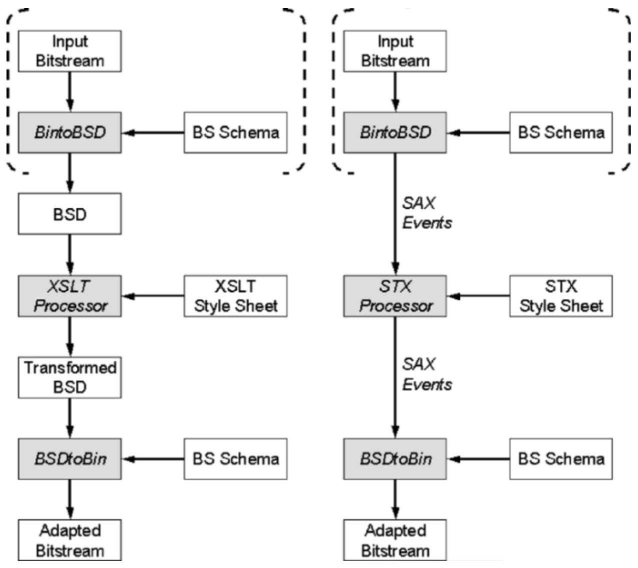


Fig. 3. Architecture for nonstreamed (left) and streamed (right) content.

The architecture is similar for a generic Bitstream Syntax Description (gBSD), except for the description generation step (BintoBSD), which is not applicable in this case. The BSDtoBin processor can then be replaced by a dedicated and more efficient processor, named gBSDtoBin, which does not need to load and parse the gBS Schema since this one is standardized and can be hard coded.

In this processing chain, the BintoBSD processor does not need to load the complete input bitstream to parse it. Similarly, the BSDtoBin step progressively generates and writes the output bitstream, with no need to store it internally. As stated at the beginning of this section, the critical issue for using the BSD framework in constrained or streaming environments is the internal memory requirements of the BintoBSD, XSLT, and BSDtoBin processors with regard to the input and output XML documents. Note though that we do not consider the memory footprint of each software module, but only of the data sets being used or produced. The following subsections study the cases of the bitstream generation (BSDtoBin) and BSD transformation. The BintoBSD is itself mentioned in Section V as a future research topic.

1) SAX Implementation of BSDtoBin: The case of the BSDtoBin processor can easily be solved by using a SAX interface [15] instead of an input document object model (DOM) [16] tree. DOM and SAX are two widely used Application Programming Interfaces (API) for accessing the content of an XML document, explained hereafter.

DOM specifies an abstract model for XML documents consisting of a hierarchical tree of "nodes", which allows an easy access to data, but is memory consuming since the full document usually needs to be loaded. Alternatively, SAX is a lower level API specifying a set of callback functions called events. In this case, the parser and the application do not exchange a memory representation of the input document as with the DOM, but a stream of events.

A new lightweight implementation of the BSDtoBin processor has been developed, using this set of input SAX events to generate the bitstream. In this implementation, the BSDtoBin processor does not need to reconstruct or store the full document in memory, but instead processes the input SAX events on the fly. In other words, the bitstream is progressively generated while the input BSD is being parsed. The input SAX events consumed by the BSDtoBin processor can be emitted by a SAX parser reading the BSD, or, in our case, transmitted as an output of the BSD transformation as described below.

Note that, conversely, and as for BintoBSD, the BS Schema still needs to be fully loaded, which, due to its limited size, is not an issue though.

2) Use of Streaming Transformations: STX: Unlike BSDtoBin where the memory requirements can be reduced by an appropriate implementation, the fact that the XSLT processor needs to load the full XML document before processing it is an inherent constraint of the language. As seen above, this may be a blocking issue in constrained or streaming environments. It should be noted though that the choice of the transformation language is not mandated by the DIA specification; the use of XSLT is provided as a preferred example due to its wide acceptance in the XML world. To overcome this inherent limitation of XSLT, we propose to use another XML transformation language named STX. STX [17], [18] is a promising ongoing open source project, aiming at specifying a new transformation language for processing a stream of SAX events. STX uses a syntax similar to XSLT, and, according to the authors, can achieve any transformation specified in XSLT. Implementations are available, though not optimized, which still impedes objective performance evaluation compared to XSLT. The input SAX events are processed according to a set of templates and trigger new output events. For the matching process, and unlike XSLT, the STX processor internally stores a limited context comprising the current event, the following one, and a stack of the ancestors of the current element node. STX style sheets can also be cascaded to form a bank of transformations.

In some cases, to process an event, an information is required that is available later in the stream of events. STX allows buffering a substream of events until the required information is retrieved, and then re-processes this buffer. In this case, the extra memory consumption due to buffering is not inherent in the transformation language, but is due to the bitstream structure and is controlled by the style sheet. Any dedicated software would face the same constraint.

By using STX and a SAX-based implementation of BSDtoBin, it is then possible to plug the output of the STX processor into the input of BSDtoBin. In this way, no intermediate data needs to be produced. Reciprocally, the input SAX events consumed by the STX filter are themselves produced by a SAX parser reading the input BSD, or by the BintoBSD processor when the BSD is produced on the fly. The full architecture based on SAX events is depicted on the right-hand side of Fig. 3.

B. PU-Based Bitstream Adaptation

In the previous section, an approach of event triggered processing of BSDs is explained which can be described as streamed processing of an XML document. In this section, a mechanism for streaming the BSD itself will be described and concepts for packetization of this stream will be specified to enable XSLT transformations under the following requirements.

- Synchronized transport, consumption, and play-out of media streams such as audio or video has to be supported.
- Adaptation on constrained devices has to be supported, e.g., the adaptation of a JPEG2000 image or MPEG-4 video on a network proxy limited in terms of memory size.

Accordingly, streaming of BSD is not only considered for time-based media streams but also for nonsynchronized media streams. In the following, the approach developed for the streaming of the descriptions is applicable to both BSDs and gBSDs. Conversely, the processing of the PUs is more specifically dedicated to gBSDs. Consequently and for the sake of consistency, we will focus on gBSDs.

1) Streaming of gBSDs: The motivation to investigate streaming formats for gBSDs is given by the block diagram of a gBSD-based bitstream adaptation in Fig. 4. In this block diagram, the whole gBSD is broken down into PUs that describe single segments of the bitstream (BS-S). The stream of PUs is transmitted to a gBSD processor (gBSD-P) along with the described bitstream. Note that we do not cover the issue of stream synchronization in this paper, but rather focus on the streaming and processing of the gBSD. As soon as the processor receives a PU, an appropriate transformation (T) can be applied. Based on the resulting transformed PU (T-gBSD-PU), an adaptation of the described bitstream segment can be conducted by means of the bitstream processor (BS-P) which generates the adapted bitstream segment (A-BS-S), as well as its corresponding PU (TU-gBSD-PU). Both, the TU-gBSD-PU and the A-BS-S can now be streamed further to the receiver. As a result, in each

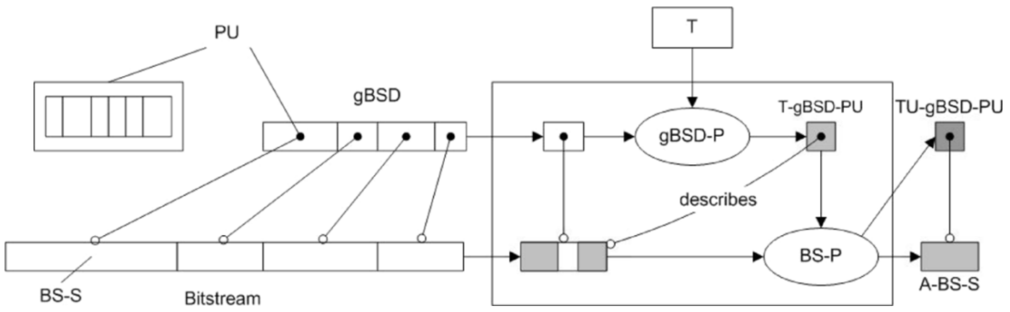


Fig. 4. gBSD transformation and bitstream adaptation using PUs.

transformation step only a fragment of the gBSD, called a PU, and the bitstream segment described by that PU are dealt with.

However, to support such an architecture, partitioning of the gBSD into PUs has to be provided. XML documents in general, and therefore also gBSDs, have a hierarchical structure: they are built by cascading elements which are signaled by opening and closing tags. Thus, XML documents do not natively support streaming as required by the described architecture—a transmitted XML document is only well-formed if also the closing tag of the root element is received.

Recently, also in the domain of content description, specifically for MPEG-7 metadata, the need for streaming of XML documents was identified and investigated. As a result, an XML streaming format for textual (TeM) and efficient binary (BiM) representations of XML documents was standardized in MPEG-7 Part 1 (Systems) [19], [20]. Both formats, TeM and BiM, are capable of breaking up the XML document (metadata) into so-called fragment update units (FUUs), which contain composition instructions (*FragmentUpdateCommands*), context information (*FragmentUpdateContext*) along with the fragment of the actual XML document. The *FragmentUpdateCommand* can signal that content be added or deleted, among other operations. The *FragmentUpdateContext* contains information about data types, parent elements, and the position of the transmitted fragment in the gBSD document. Both formats, TeM and BiM, have a broad set of tools to support functionalities such as dynamic documents on the receiver side or transmission in arbitrary order. For streaming of gBSDs, we concentrate here on the partitioning of a static XML document into consecutive BiM FUUs. Hence, only the *addContent FragmentUpdateCommand* is signaled and the BiM FUUs are arranged according to the order of the contained elements in the gBSD. As a result, the order of the transmitted FUUs corresponds to the order of the described bitstream segments.

2) Concept of PUs: So far, a mechanism to fragment and stream an XML document was described. According to the MPEG-7 Systems specification [19], [20], the fragments (FUUs) can be used on the receiver side to reconstruct the original XML document by applying the received FUUs based on the *FragmentUpdateContext* and the *FragmentUpdateCommand* to a so called binary description tree which is similar to DOM, but includes type and position information of each node. To avoid invalid states, the tree shall only be processed by an application if an access unit (AU) consisting of an arbitrary number of FUUs has completely been applied to the binary description tree. By this convention, the encoder can signal to the decoder the states in which the binary description tree is valid and can be consumed.

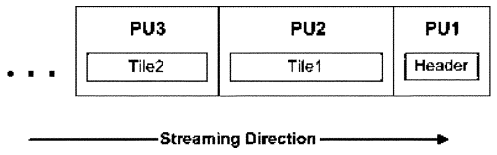


Fig. 5. PU fragmentation of a gBSD describing a JPEG2000 bitstream.

In contrast to the reception of AUs on the receiver side, the fragments of the gBSD (PUs) in the architecture of Fig. 4 are processed individually. Therefore, a PU is defined as a set of one or more FUUs which comprises all the information required for the adaptation of the bitstream segment described by this PU. For instance, the syntax of a JPEG2000 bitstream includes several parameters indicating the length of bitstream segments, e.g., within the bitstream header or tiles. According to the definition of the PU, the description of these parameters has to be contained in a single PU along with the gBSD elements describing the corresponding bitstream segment. An example of this partitioning into PUs is given in Fig. 5.

In Fig. 6, a video bitstream encoded using the MPEG-4 Visual Advanced Simple Profile is illustrated. The bitstream comprises frames of different types and the gBSD is fragmented into PUs describing a so-called group of visual object planes. The fragmentation could be also based on visual object planes, shots, or scenes depending on the application requirements.

3) Processing of PUs: To process the gBSD partitioned into PUs by means of, e.g., an XSLT processor, the PUs have to be: 1) identified in the gBSD stream and 2) instantiated as XML document representations. Mechanisms for both required processing steps are described in this section.

For signaling the PUs, two approaches are applicable: on the gBSD (XML) level and on the systems (binary) level. On the gBSD level, a PU is identified by specific markers in a gBSD element which needs to be referenced accordingly, e.g., by an additional parameter in the BSDRef attribute of the BSDLink description. The advantage of this approach is that the definition of PUs can vary with respect to the transformation applied. If signaling on the systems level is used, the PUs are represented by AUs in the MPEG-7 BiM stream. As described above, this requires the encoder to ensure that after reception of each AU the binary description tree is valid. However, the partitioning of the gBSD into elements based on gBSDUnitType is usually the finest level of partitioning and fulfills this requirement. The

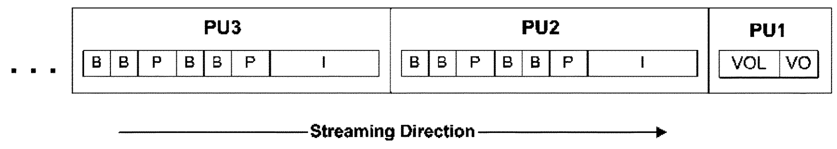


Fig. 6. PU fragmentation of a gBSD describing an MPEG-4 video bitstream.

TABLE I
SIZES OF BITSTREAM SEGMENTS AND CORRESPONDING PUs FOR TEST SEQUENCES

Codec	BS-S	PUs (text.)	PUs (binary)
MPEG-4 Visual	87.7–168.7 kB	5.7–6.8 kB	549–596 Bytes
BSAC	125–358 Bytes	5.1–5.3 kB	594–614 Bytes

advantage of this approach is fast determination of the start and end of a PU without interpretation of the MPEG-7 fragment information.

The restriction of context information in the transmission of gBSDs by using PUs based on MPEG-7 Systems enables the usage of legacy transformation processors such as XSLT in streaming scenarios. However, for this purpose, the so-called PU documents need to be decoded. A PU document is an XML representation of the binary description tree which is built from the *DecoderInit* initially sent to configure the MPEG-7 BiM decoder and the PU to be processed. Accordingly, legacy XSLT transformations can be specified which are applicable to these PU documents. Using this approach, existing and efficient implementations of XSLT processors can be applied with limited context information. Fragmenting the gBSD into PUs helps to efficiently resolve the major problem of XSLT processors which is that, in legacy mode, they build up a context of the whole gBSD of a stream before a transformation is applied.

Experiments were conducted using the “Foreman” test sequence encoded in CIF resolution according to MPEG-4 Visual Advanced Simple Profile @ Level 1 and the music test sequences of the European ISIS project encoded with the bit-sliced arithmetic coding (BSAC) codec according to MPEG-4 Mobile Audio Internetworking Profile @ Level 1. In the first case, a gBSD was generated to support B-VOP dropping. In the case of BSAC, SNR scalability was supported by the gBSD. In Table I, the size ranges of the described BS-S and the resulting PUs in textual and binary representations are listed. The typical sizes of textually represented PU documents for MPEG-4 video and MPEG-4 BSAC streams are about 5–6 kB and their corresponding binary representations are about 550–600 B. Even in the textual representation, the resulting PUs require modest memory in XSLT processing. This enables transformation and adaptation even on memory constrained devices.

V. Future Work

Beyond the developments presented above for adapting the architecture to streaming and constrained environments, we are currently working on evaluating and optimizing the overall performance of the adaptation methods. For this, we are investigating how to reach the best trade-off between preserving a modularized and generic approach applicable to both BSDs and gBSDs, and optimizing the performance by using dedicated tools for gBSDs. In the PU approach in particular, we are concentrating on the use of gBSDs and XSLT which allows using the dedicated and optimized gBSDtoBin processor instead of the more generic BSDtoBin.

The computation time of the transformation can be decreased by an adequate design of the transformation style sheet, minimizing in particular the number of candidate templates to be matched. First experiments show that the gain is significant, regardless of the XSLT implementation being used. Furthermore, it is possible to gain processing time by combining the XSLT transformation and the adapted bitstream generation (gBSDtoBin) in a single dedicated implementation.

Another challenging research topic is the possibility to process the gBSDs in the binary domain. As shown in Section IV-B, it is possible to use the efficient binary representation of a gBSD in the MPEG-7 BiM format. A first advantage of this format is obviously the reduction of the description size. Furthermore, in this binary format, the XML elements and their contents are not represented by strings, but by binary symbols deduced from the schema. Using the binary version of a gBSD should therefore significantly reduce the complexity of some processing steps requiring string comparisons otherwise, which are particularly resource consuming. For this, new transformations need to be specified that would be capable of processing binary descriptions. Processing and applying transformations in the binary domain is therefore a challenging and promising research topic for our future work.

Lastly, another issue is the adaptation of the BintoBSD processor to streaming environments where the BSD needs to be generated on the fly, e.g., in conversational scenarios where the video is produced and encoded continuously. A specific feature of BSDL with respect to the BSD generation is the use of XPath expressions [21], as explained in [2] and [6]. These expressions are evaluated at run-time, i.e., while parsing the bitstream and progressively generating the BSD, against the partially instantiated description. For this, the BintoBSD processor needs to store an internal representation of the description, typically as a DOM. In streaming scenarios or for very large bitstreams, the growing size of this internal representation may generate a memory overflow. To solve this issue, we are investigating how to restrict the use of XPath expressions so that only a minimal part of the description needs to be stored and the BintoBSD processor can complete the fully streamable architecture depicted in Fig. 3.

VI. Conclusion

In previous work, a generic framework for adapting multimedia content in a coding format-independent way was introduced. This method uses XML for describing the high-level structure of a bitstream and the XSLT language to transform this description into a new document, from which the adapted bitstream can be generated. This framework was initially developed for nonstreamed content; this paper shows how it can be adapted to streaming and constrained environments. To this end, new solutions are required to overcome some inherent limitations of XML-related technologies such as XSLT, which were initially developed for static, text documents. New multimedia applications have raised and will raise new, challenging requirements such as streaming, synchronization, and efficient binary representation. This paper shows that by using recent technologies such as MPEG-7 BiM or the STX transformation language, XML can be efficiently used for multimedia content adaptation in a streaming and constrained environment.

Acknowledgment

This paper is based on the work conducted by several people in the European research projects ISIS and DANAE. In particular, the authors thank G. Panis for his valuable input regarding the experimental results for this contribution.

References

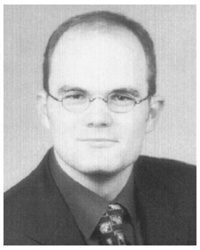
[1] A. Vetro and C. Timmerer, "Overview of the digital item adaptation standard," *IEEE Trans. Multimedia*, vol. 7, no. 3, pp. 418–426, Jun. 2005.
[2] A. Vetro, C. Timmerer, and S. Devillers, "Digital item adaptation," in *The MPEG-21 Book*. Hoboken, NJ: Wiley, 2005, to be published.
[3] G. Panis, A. Hutter, J. Heuer, H. Hellwagner, H. Kosch, C. Timmerer, S. Devillers, and M. Amielh, "Bitstream syntax description: A tool for multimedia resource adaptation within MPEG-21," *EURASIP Signal Process.: Image Commun. J.*, vol. 18, no. 8, pp. 721–747, Sep. 2003.
[4] XSL Transformations (XSLT), W3C Recommendation (1999). [Online]. Available: http://www.w3.org/TR/xslt
[5] M. Amielh and S. Devillers, "Bitstream syntax description language: Application of XML schema to multimedia content adaptation," in *Proc. 11th Int. World Wide Web Conf. (WWW2002)*, Honolulu, HI, May 2002.
[6] S. Devillers, "An extension of BSDL for multimedia bitstream syntax description," in *Proc. 9th Int. Conf. Parallel and Distributed Computing (Euro-Par 2003)*, vol. 2790, Lecture Notes in Computer Science, Aug. 2003.
[7] *Information Technology – Multimedia Framework – Part 7: Digital Item Adaptation*, ISO/IEC 21 000-7.
[8] XML Schema Part 0: Primer, Part 1: Structures and Part 2: Datatypes, W3C Recommendation (2001). [Online]. Available: http://www.w3.org/XML/Schema
[9] D. Mukherjee, E. Delfosse, J. G. Kim, and Y. Wang, "Adaptation QoS and UCD," *IEEE Trans. Multimedia*, vol. 7, no. 3, pp. 454–462, Jun. 2005.
[10] A. Eleftheriadis, "Flavor: A language for media representation," in *ACM Multimedia Conf.*, Seattle, WA, Nov. 1997, pp. 1–9.
[11] D. Hong and A. Eleftheriadis, "XFlavor: Bridging bits and objects in media representation," in *Proc. IEEE Int. Conf. Multimedia and Expo (ICME'2002)*, Lausanne, Switzerland, Aug. 2002.
[12] *Information Technology – JPEG 2000 Image Coding System – Part 1: Core Coding System*, ISO/IEC 15 444-1:2000.
[13] D. Taubman and R. Prandolini, "Architecture, philosophy and performance of JPIP: Internet protocol standard for JPEG2000," in *Proc. Int. Symp. Visual Communications and Image Processing (VCIP2003)*, vol. 5150, SPIE, Jun. 2003, pp. 791–805.
[14] C. Timmerer, G. Panis, H. Kosch, J. Heuer, H. Hellwagner, and A. Hutter, "Coding format independent multimedia content adaptation using XML," in *Proc. SPIE Int. Symp. ITCom 2003*, vol. 5242, Orlando, FL, Sep. 2003.
[15] Simple API for XML [Online]. Available: http://www.saxproject.org
[16] Document Object Model (DOM) Level 2 Core Specification, W3C Recommendation (2000). [Online]. Available: http://www.w3.org/TR/DOM-Level-2-Core
[17] O. Becker, "Transforming XML on the fly," in *XML Europe 2003*, May 2003.
[18] Streaming Transformations for XML (STX), Working Draft (2004). [Online]. Available: http://stx.sourceforge.net
[19] *Information Technology – Multimedia Content Description Interface – Part 1: Systems*, ISO/IEC 15 938-1.
[20] J. Heuer, C. Thienot, and M. Wollborn, "Binary format," in *Introduction to MPEG-7: Multimedia Content Description Interface*, B. S. Manjunath, P. Salembier, and T. Sikora, Eds. New York: Wiley, 2002, pp. 61–80.
[21] XML Path Language (XPath), W3C Recommendation (1999). [Online]. Available: http://www.w3.org/TR/xpath

Sylvain Devillers received the engineer degree from Ecole Nationale Supérieure des Télécommunications de Paris (ENST), Paris, France, in 1993, with a specialization in image processing.

After working for Alcatel CIT, Cork, Ireland, he joined Philips Research France in 1996, where he worked on image processing algorithms for medical applications, then on multimedia indexing and multimedia content adaptation with a strong involvement in MPEG standardization. In 2003, he then moved to IMEC, Leuven, Belgium, to continue his contribution to MPEG-21 DIA. His current research interests are related to multimedia applications, scalable media, and XML technologies.

Christian Timmerer received the Dipl.-Ing. degree in applied informatics from the Department of Information Technology (ITEC), University of Klagenfurt, Klagenfurt, Austria, where he is currently pursuing the Ph.D. degree in the field of multimedia adaptation in the context of MPEG-21.

He is currently a University assistant and chairs the IT administration group of ITEC. At the University of Klagenfurt, he has been working on coding-format agnostic resource adaptation within the MPEG-21 Multimedia Framework and he actively participated in the work of ISO/MPEG for several years. His other research interests include the transport of multimedia content, multimedia adaptation in constrained and streaming environments, and distributed multimedia adaptation.



Jörg Heuer received the Dipl.-Ing. degree in electrical engineering at the Friedrich-Alexander University, Erlangen, Germany, 1997, with a specialization on digital signal processing and high-frequency engineering and the Ph.D. degree (supported by the Corporate Technology of Siemens) from the Friedrich-Alexander University in 2003 for his work on multimedia content description.

He joined the Corporate Technology of Siemens AG, Munich, Germany, in 2002, where he is working as Senior Scientist in the fields of multimedia content description and metadata coding in the domain of communication applications. His other research interests include multimedia adaptation in heterogeneous environments and indexing of metadata.

Hermann Hellwagner (S'85–M'95) is a Full Professor of Computer Science at the Department of Information Technology (ITEC), University of Klagenfurt, Klagenfurt, Austria. His current areas of research include distributed multimedia systems, multimedia communications, and Internet QoS. His current projects involve digital video communication, a streaming protocol and a system supporting media adaptation, and multimedia bitstream description techniques within the MPEG-21 Digital Item Adaptation (DIA) standardization effort. He is the editor of several books and has published papers on parallel computer architecture and parallel programming and, more recently, on multimedia communications and adaptation.

Dr. Hellwagner is a member of the German Informatics Society (GI) and the Austrian Computer Society (OCG), as well as the Head of the Austrian delegation to the Moving Picture Experts Group (MPEG – ISO/IEC JTC1/SC29/WG11). He has also organized several international conferences and workshops.