G

GENERIC MULTIMEDIA CONTENT ADAPTATION

Christian Timmerer, Michael Ransburg, and Hermann Hellwagner
Department of Information Technology, Klagenfurt University, Austria

Synonym: *Coding format independent multimedia content adaptation; coding format agnostic multimedia content adaptation; XML metadata-based adaptation of multimedia content independent of the coding format*

Definition: *Generic multimedia content adaptation is referred to as the customization of multimedia content to various usage contexts without being aware of the actual coding format used.*

Acknowledgment: This work was/is support in part by the European Commission in the context of the DANAE (FP6-IST-1 507113), ENTHRONE (FP6-IST-1-507637), and P2P-Next (FP7-ICT-216217) projects. Further information is available under http://danae.rd.francetelecom.com/, http://www.ist-enthrone.org/, and http://www.p2p-next.eu/.

Introduction

The ever growing variety of audio/visual coding formats calls for a methodology which enables the processing of multimedia content without considering the specific coding format used. In practice, the end user is not interested in the details of the coding format as she/he is usually interested in the actual content only, e.g., a movie or a picture. On the other hand, the content and service providers aim at avoiding maintaining multiple variations of the same content, each conforming to a different coding format, in order to fulfill the requirements of their customers. A means to close this gap between providers and consumers in terms of format incompatibilities (among others) is generally referred to as Universal Multimedia Access (UMA) [1]. Multimedia content adaptation is one technique that contributes to the vision of UMA, namely to access multimedia content from anywhere, anytime, and with any device. As devices may have varying decoding capabilities, transcoding of the content to these capabilities is required. However, transcoding is to be considered like a patch to the issues indicated above, which requires specific solutions for a growing number of coding formats. Scalable coding formats would be a generalized solution to the transcoding/adaptation problem and would facilitate UMA in a generic way. In practice, however, the variety of competing or complementary scalable coding format, e.g., [2-9], leads back to the transcoding issue where a device needs to maintain the corresponding number of processing modules in order to facilitate the manipulation of bitstreams conforming to these formats.

This article discusses means to process (i.e., adapt, customize, manipulate, etc.) multimedia content independently of the actual coding format by utilizing XML-based metadata describing the high-level structure (i.e., syntax) of a bitstream. That is, the resulting XML document describes the bitstream how it is organized at different syntactical and even semantic levels, e.g., in terms of packets, headers, layers, units, segments, shots, scenes, etc., depending on the actual application requirements. It is important to note that the XML description does not describe the bitstream on a bit-by-bit basis, i.e., it does not replace the actual bitstream but provides metadata regarding bit/byte positions of meaningful segments for the given application. Therefore, the XML description does not necessarily provide any information of the actual coding format used as only the positions and – in some cases – meanings are required for processing. The main application of such an XML description is currently adaptation of multimedia bitstreams as described in the following sections.

High-level Architecture of Generic Content Adaptation

Figure 1 depicts the high-level architecture of generic multimedia content adaptation which can be logically divided into two processes, namely the *Description Transformation* and the *Bitstream Generation*.

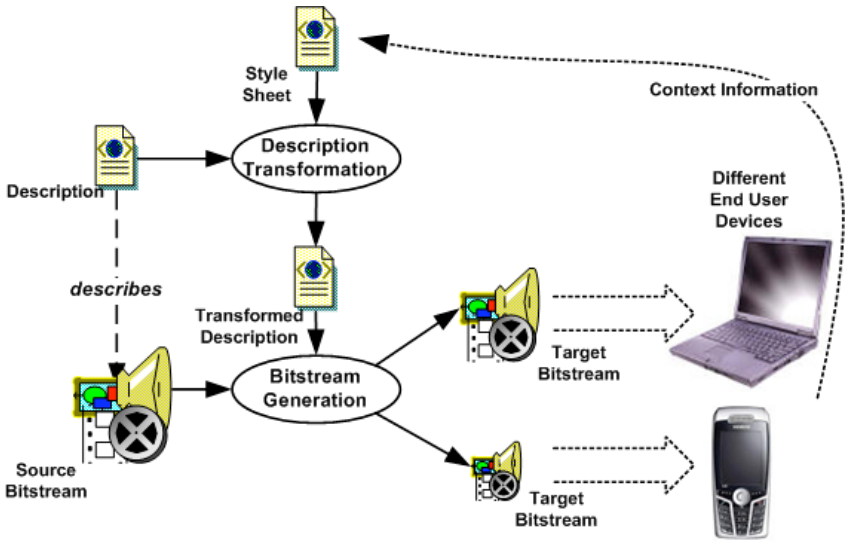

Figure 1. High-Level Architecture of Generic Multimedia Content Adaptation (adopted from [10]).

The *description transformation* process receives as an input the XML *description* of the *source bitstream* and a so-called *style sheet* that transforms the XML document according to the *context information*, e.g., the device capabilities. The output of this process is a *transformed description* which already reflects the bitstream segments of the target (i.e., adapted) bitstream. However, the transformed description still refers to the bit/byte positions of the source bitstream which needs to be parsed in order to generate the *target bitstream* within the second step of the adaptation process, i.e., the *bitstream generation*.

Please note that the description transformation and bitstream generation processes should be combined by applying appropriate implementation techniques in order to achieve the required performance. However, implementation and optimization techniques for this kind of approach are out of scope of this article and the interested reader is referred to [11-14].

Technical Solution Approaches

The literature offers several technical solution approaches for generic multimedia content adaptation which are briefly highlighted in the following:

- **(X)Flavor** [15]: A Formal Language for Audio-Visual Object Representation which has been extended with XML features.
- **Bitstream Syntax Description Language (BSDL)** [16]: An XML Schema-based language for constructing a Bitstream Syntax Schema (BS Schema) for a given coding format [17]. It enables the generation of a Bitstream Syntax Description (BSD) based on a given bitstream and vice versa. The generic counterpart of the coding format-specific BS Schema is referred to as gBS Schema which is fully coding format-agnostic. An XML document conforming to the gBS Schema is referred to as a generic Bitstream Syntax Description (gBSD) [18].
- **BFlavor** [19]: A method that combines **B**SDL and X**Flavor** and basically uses XFlavor techniques – enhanced with BSDL concepts – to generate Java code which is used for automatic generation of BSDs.

In the following, the focus will be on gBSD as it does not require conveying any coding format-specific information, neither in the schema nor in the instance. Please note that gBSD (and also BSDL) is standardized as part of MPEG-21 Digital Item Adaptation (DIA) [20] which enables interoperability between different vendors.

generic Bitstream Syntax Description

The main advantage of the gBSD-based approach is that the XML Schema to which a gBSD shall conform does not convey any information pertaining to a certain coding format. Thus, the gBSD-based approach enables fully coding format independence and provides the following functionalities:

- The description may convey arbitrary bitstream segments and also individual parameters.
- The bitstream segments may be grouped in a hierarchical way allowing for efficient, hierarchical adaptation.
- A flexible addressing scheme supports various application requirements and allows for random access into a bitstream.
- The so-called "marker" attribute provides semantically meaningful marking of syntactical elements.
- The gBSD-based approach enables support for dynamic and distributed adaptation (discussed in the next section).

Elements of a gBSD

The two main elements of a gBSD are the `gBSDUnit` and the `Parameter` elements. Before going into details of these elements, it is noted that each gBSD can be embedded in any

other XML document by means of the `gBSDType` complex type which constitutes the root data type of a gBSD. Within this data type, several addressing-related attributes may be defined such as the address mode, address unit, and a Uniform Resource Identifier (URI) identifying the actual bitstream.

The address mode can be *absolute*, *consecutive*, and *offset*. With the *absolute* address mode it is possible to describe the *start* and *length* of a gBSD element, which is useful for bitstream segments that may be dropped during adaptation. The other two modes, *consecutive* and *offset*, facilitate fast access to small, contiguous bitstream segments that may be updated only, possibly due to the removal of other segments. The address unit differentiates between *bit* and *byte* addressing and the bitstream URI identifies the actual bitstream that is described by this gBSD.

The scope of the address attributes defined as part of the `gBSDType` is the whole gBSD unless superseded by child elements, i.e., the `gBSDUnit` and the `Parameter` elements which are described in the following. Note that the example gBSD fragments used in the following describe bitstreams conforming to ISO/ITU-T Scalable Video Coding (SVC) [5] and JPEG2000 [9].

gBSDUnit. This kind of element may be used to describe a specific segment of a bitstream without including the exact values of the bitstream. It is particularly designed for bitstream segments that may be removed during adaptation (e.g., discarding enhancement layer data) or for containing further gBSD elements (i.e., `gBSDUnit` or `Parameter`) which leads to a hierarchical structure. An example `gBSDUnit` element of an SVC network abstraction layer (NAL) unit that corresponds to the base layer is shown in Example 1.

```
<gBSDUnit start="124" length="89" syntacticalLabel=":SVC:NALU" marker="Q0 S0 T0"/>
```

Example 1. `gBSDUnit` for an SVC NAL unit corresponding to the base layer in terms of quality (Q), spatial (S) and temporal (T) resolution.

As shown in the example, each `gBSDUnit` element may contain several attributes:

- `start` and `length`: Attributes conveying addressing information depending on the address mode/unit.
- `syntacticalLabel`: Attribute for including coding format-specific information identified via a pre-defined vocabulary of controlled terms. In practice, however, this attribute is rarely needed and included here only for the sake of completeness and presentation.
- `marker`: Attribute which provides a handle for application-specific information. In the example above it is used to convey information about the scalability layer of the described bitstream segment. In particular, the `gBSDUnit` element in Example 1 describes a bitstream segment of the SVC base layer.

Although the `syntacticalLabel` and `marker` attributes provide means for including coding format-specific information, it is worth mentioning that both attributes are optional and, thus, can be omitted. Furthermore, the syntax and semantics of the `marker` attribute are neither defined within gBSD nor SVC and, hence, coding format independence is preserved.

Example 2 shows a `gBSDUnit` element describing an SVC access unit (AU) which contains three `gBSDUnit` elements each describing an SVC NALU. The example demonstrates the feasibility of hierarchical `gBSDUnit` elements as – in case all temporal layers ≥ 2 should be dropped – it would require to parse and inspect only the `gBSDUnit` element enclosing all its child elements.

```
<gBSDUnit syntacticalLabel=":SVC:AU" marker="Q0 T2">
  <gBSDUnit start="2805" length="10" syntacticalLabel=":SVC:NALU" marker="S0"/>
  <gBSDUnit start="2815" length="110" syntacticalLabel=":SVC:NALU" marker="S1"/>
  <gBSDUnit start="2925" length="335" syntacticalLabel=":SVC:NALU" marker="S2"/>
</gBSDUnit>
```

Example 2. Hierarchical `gBSDUnit` elements.

Parameter. This kind of element may be used to include an actual numerical value and its data type of a bitstream segment. Therefore, it can be updated during the adaptation process. Example 3 shows a `Parameter` element which indicates the number the color components of a JPEG2000 image.

```
<Parameter syntacticalLabel=":J2K:Csiz" length="2">
  <Value xsi:type="xsd:unsignedShort">3</Value>
</Parameter>
```

Example 3. JPEG2000 `parameter` providing information about the size (number) of the color components (`Csiz`).

In this example three color components are available and, thus, a colored image is described. If the picture has to be adapted to a grayscale image, the number of color components has to reduced to one which results in updating the value of the `Parameter` element. The data type – signaled through the `xsi:type` attribute – and the `length` attribute provide the details for the correct encoding of this parameter during bitstream generation.

Description Transformation and Bitstream Generation using gBSD

As mentioned in the introduction, the generic adaptation of multimedia content can be logically divided into two steps, namely the *description transformation* and *bitstream generation*. The gBSD-based approach – as well as the other approaches – does not define any restrictions which kind of description transformation shall be applied. In fact, each method that is able to transform XML documents could be applied and should be carefully chosen according to the application domain that adopts this approach. A simple (but expensive – in terms of memory requirements [12][14]) method is the usage of the well-known Extensible Style Sheet Language for Transformations (XSLT)[1] which is used in the following examples.

The gBSD fragment as shown in Example 4 describes an SVC elementary stream with a single quality layer (Q0, e.g., PSNR=32 dB), two spatial layers (S0-1, e.g., QCIF and CIF), and three temporal layer (T0-2, e.g., 7.5 Hz, 15 Hz, and 30 Hz). Interestingly, the gBSD is designed in a way that the temporal layers are hierarchically summarized featuring

[1] http://www.w3.org/TR/xslt

efficient, hierarchical adaptation in the temporal domain. Note that `gBSDUnit` elements belonging to the temporal base layer are not grouped as they are usually not removed.

```
<?xml version="1.0" encoding="UTF-8"?>
<dia:DIA xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS" xmlns="urn:mpeg:mpeg21:2003:01-DIA-
gBSD-NS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dia:Description xsi:type="gBSDType" addressUnit="byte" addressMode="Absolute">
    <gBSDUnit start="0" length="124"/>
    <gBSDUnit start="124" length="89" marker="Q0 S0 T0"/>
    <gBSDUnit start="213" length="311" marker="Q0 S1 T0"/>
    <gBSDUnit start="524" length="38" marker="Q0 S0 T0"/>
    <gBSDUnit start="562" length="51" marker="Q0 S1 T0"/>
    <gBSDUnit marker="Q0 T1">
      <gBSDUnit start="613" length="24" marker="S0"/>
      <gBSDUnit start="637" length="110" marker="S1"/>
    </gBSDUnit>
    <gBSDUnit marker="Q0 T2">
      <gBSDUnit start="747" length="10" marker="S0"/>
      <gBSDUnit start="757" length="110" marker="S1"/>
      <gBSDUnit start="867" length="10" marker="S0"/>
      <gBSDUnit start="877" length="110" marker="S1"/>
    </gBSDUnit>
    <gBSDUnit start="987" length="93" marker="Q0 S0 T0"/>
    <gBSDUnit start="1080" length="318" marker="Q0 S1 T0"/>
    <!-- ... and so on ... -->
  </dia:Description>
</dia:DIA>
```

Example 4. gBSD fragment describing an SVC elementary stream.

An XSLT style sheets which removes all `gBSDUnit` elements that belong to the temporal layer 2 is shown in Example 5. As shown in the XSLT style sheet, only the `gBSDUnit` element containing a `marker` with value 'T2' needs to be investigated without processing its child elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS" xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-
NS">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="gbsd:gBSDUnit[contains(@marker, 'T2')]">
    <xsl:comment>removed !!!</xsl:comment>
  </xsl:template>
</xsl:stylesheet>
```

Example 5. XSLT style sheet for removing `gBSDUnit` elements belonging to temporal layer 2.

The resulting gBSD fragment when applying the XSLT style sheet of Example 5 is shown in Example 6. The gBSDUnit elements belonging to the second temporal layer have been removed (i.e., replaced by the XML comment "removed !!!" for demonstration purposes).

```
<?xml version="1.0" encoding="UTF-8"?>
<dia:DIA xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS" xmlns="urn:mpeg:mpeg21:2003:01-DIA-
gBSD-NS" xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dia:Description xsi:type="gBSDType" addressUnit="byte" addressMode="Absolute">
```

```
    <gBSDUnit start="0" length="124"/>
    <gBSDUnit start="124" length="89" marker="Q0 S0 T0"/>
    <gBSDUnit start="213" length="311" marker="Q0 S1 T0"/>
    <gBSDUnit start="524" length="38" marker="Q0 S0 T0"/>
    <gBSDUnit start="562" length="51" marker="Q0 S1 T0"/>
    <gBSDUnit marker="Q0 T1">
      <gBSDUnit start="613" length="24" marker="S0"/>
      <gBSDUnit start="637" length="110" marker="S1"/>
    </gBSDUnit>
    <!–removed !!!-->
    <gBSDUnit start="987" length="93" marker="Q0 S0 T0"/>
    <gBSDUnit start="1080" length="318" marker="Q0 S1 T0"/>
    <!-- ... and so on ... -->
  </dia:Description>
</dia:DIA>
```

Example 6. Transformed gBSD fragment of Example 4 after applying the style sheet of Example 5.

The bitstream generation process – also referred to as *gBSDtoBin* – takes the transformed gBSD (cf. Example 6) as input and generates the target bitstream from its source. Note that the URI of the source bitstream is not included in the gBSD and, thus, it is up to the application invoking *gBSDtoBin* to provide the source bitstream. Alternatively, the source bitstream's URI could be provided with the `bitstreamURI` attribute within the `dia:Description` element. The *gBSDtoBin* process hierarchically parses the transformed gBSD in depth-first order and constructs the target bitstream. Therefore, *gBSDtoBin* copies the bitstream segments from the source to the target based on the `start`/`length` information provided by the remaining `gBSDUnit` elements of the transformed gBSD. In this way, the target bitstream is successively generated by means of its source and the transformed gBSD.

Adding Support for Dynamic and Distributed Adaptation

The gBSD-based adaptation approach, as introduced above, was originally developed with static, server-centric application scenarios in mind. That is, the adaptation of the bitstream is performed at the server; the necessary details for dynamic and distributed adaptation are not considered. *Dynamic adaptation* is referred to as the adaptation of bitstreams according to dynamically changing usage environments and *distributed adaptation* refers to multiple adaptations steps successively performed on different processing modules/nodes. The processing and delivery of media in a streamed (i.e., dynamic) fashion has many advantages, e.g., low start up delay, and is therefore commonly used to consume large media files over networks. Server-centric adaptation is only ideal if the problem which adaptation tries to address occurs on the server or in its close vicinity. It is generally better to perform adaptation close to the location of the problem, since the potentially high delay when adapting on the server to a problem which occurs at the end device can decrease the quality of experience for the end user. Therefore, the gBSD-based adaptation approach has been extended to apply to dynamic and distributed adaptation of streaming media, while staying backwards compatible with the original mechanisms [21]. In particular, the extension is enabled by the introduction of:

- *XML Streaming Instructions (XSI),* which adds the concept of ”samples” to metadata; and
- *Media Streaming Instructions (MSI)* to allow for synchronized processing of both media and metadata.

The two types of streaming instructions are briefly reviewed in the following.

The *XML Streaming Instructions (XSI)* provide the information required for streaming an XML document by the composition and timing of so called *Process Units (PUs)*, i.e., metadata samples. The XSIs allow firstly to identify Process Units (PUs) in an XML document and secondly to assign time information to them. More formally, a PU is defined as a well-formed fragment of XML metadata (i.e., a gBSD fragment) that can be consumed as such and to which time information can be attached. It is specified by the `anchor` element and by a PU mode indicating how other – connected – elements are aggregated to this anchor to compose the PU. The various supported composition modes are illustrated in Figure 2, in which each node represents an XML element and the white node represents the `anchor` element. Depending on the mode, the `anchor` element is not necessarily the root of the PU; `anchor` elements are ordered according to the navigation path of the XML document. PUs may overlap, i.e., some elements (including `anchor` elements) may belong to several PUs. Additionally, the content provider may require that a given PU be encoded as a *random access point*, i.e., that the encoded PU does not require any other encoded PUs to be decoded.

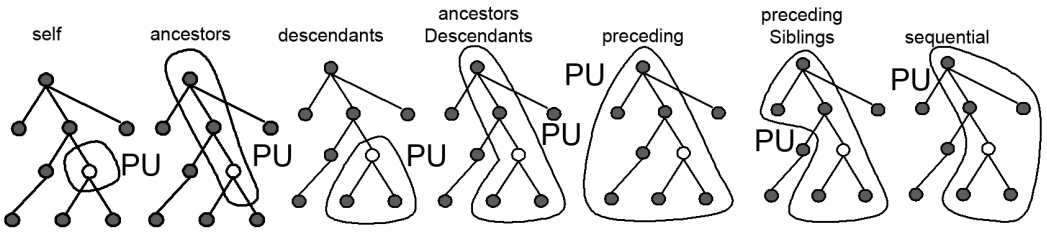

Figure 2. PU modes with the anchor element illustrated as a white node.

The *Media Streaming Instructions (MSI)* specify two sets of properties for annotating an XML document. The first set indicates the media *Access Units (AUs)* and their location in the described media bitstream, the random access points, and the subdivision into AU parts. The second set provides the AU time stamps.

Example 7 shows a fragment of a gBSD which includes both XML and Media Streaming Instructions as XML attributes. Note that an alternative representation is possible, which assigns the Streaming Instructions from a separate XML document, i.e., a so called Properties Style Sheet (PSS) [13]. Within the *Description* element the streaming instructions are specified that are valid for the complete gBSD, i.e., which are inherited by all descendant elements. These top level attributes includes the time scale for both the media and the metadata stream and the way in which media AUs and PUs shall be composed, which is specified through the `auMode` and `puMode` attributes. On AU level, for each AU, the `anchorElement` is defined which, together with the PU mode (cf. Figure 2), steers the composition of the PU. Additionally, timing information for both the PU and the media AU is provided.

```
<?xml version="1.0" encoding="UTF-8"?>
<dia:Description xmlns:xmlsi="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
  xmlns:msi="urn:mpeg:mpeg21:2003:01-DIA-MSI-NS" msi:timeScale="1000" msi:auMode="tree"
  xmlsi:timeScale="1000" xmlsi:puMode="ancestorsDescendants">
   <!-- ... further elements ... -->
   <gBSDUnit syntacticalLabel=":SVC:AU" msi:dts="80" msi:cts="1360" msi:au="true"
     xmlsi:anchorElement="true" xmlsi:absTime="80">
```

```
        <gBSDUnit start="124" length="89" syntacticalLabel=":SVC:NALU" marker="Q0 S0 T0"/>
        <gBSDUnit start="213" length="311" syntacticalLabel=":SVC:NALU" marker="Q0 S1 T0"/>
    </gBSDUnit>
    <!-- ... further elements ... -->
</dia:Description>
```

Example 7. gBSD fragment with Streaming Instructions as attributes.

Figure 3 represents the extension of the previously introduced high-level architecture for generic multimedia content adaptation. Thus, it depicts how the streaming instructions can be integrated with the gBSD-based adaptation approach in order to enable dynamic and distributed adaptation.

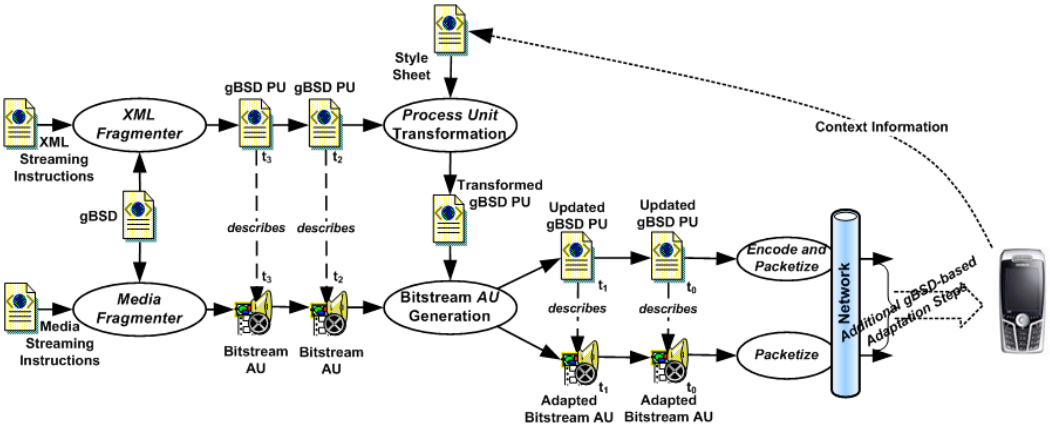

Figure 3. gBSD-based Dynamic and Distributed Adaptation.

The gBSD is provided together with the XSIs, logically separated in the figure, to the *XML fragmenter*. The XML fragmenter then determines the next PU from the gBSD and assigns a time stamp to it. This PU is then transformed by the *process unit transformation* process in the same way as a complete gBSD would be transformed. The *transformed PU* is forwarded to the *bitstream AU generation* process, which has the appropriate media AU and its time stamp available, thanks to the *media fragmenter* which extracted it based on the MSIs. The *bitstream AU generation* process adapts the media AU in order to correspond to the transformed PU and updates the `start` and `length` attributes of the PU to reflect the adapted media AU. Since each AU is adapted individually and just before it is streamed out, this allows to react to dynamically changing usage environment updates, such as the available network bandwidth. The updated PUs which are still represented in the text domain, are then encoded into AUs using a proper encoding mechanism, e.g., MPEG's Binary Format for Metadata (BiM) [22]. After encoding the PUs into binary AUs, the media and gBSD AUs are packetized for transport. In this step, the timing information provided by MSIs and XSIs is mapped onto transport layer protocols (e.g., the Real-Time Transport Protocol), by including it into the packet header. Both the media and gBSD AUs are then streamed into the network, where an adaptation proxy could perform additional adaptation steps, or to an end device where the dynamically adapted media is consumed. In the latter case, the transport of the metadata may be omitted.

Concluding Remarks

Due to the increasing amount of – complementary and competing – (scalable) coding formats, content and service providers demand for efficient and flexible processing/adaptation mechanisms in order to satisfies their customer's needs. The generic multimedia content adaptation framework as described in this article provides a technical solution for this problem which has been validated with existing coding formats and which is also future-proof for coding formats yet to be developed.

See: Universal Multimedia Access, MPEG-21 Digital Item Adaptation, MPEG-21 Multimedia Framework

References

1. A. Vetro, C. Christopoulos and T. Ebrahimi, Eds., *Special Issue on Universal Multimedia Access*, *IEEE Signal Processing Magazine*, vol. 20, no. 2, March 2003.
2. ISO/IEC 11172-2:1993, *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2: Video*, January 2005.
3. H.M. Radha, M. van der Schaar, and Y. Chen, "The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP", *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 53-68, March 2001.
4. T. Wiegand, G.J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, July 2003.
5. H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1103-1120, September 2007.
6. S.-T. Hsiang and J. W. Woods, "Embedded video coding using motion compensated 3-D subband/wavelet filter bank", *Signal Processing: Image Communication*, vol. 16, no. 8, pp. 705–724, 2001.
7. S. Park, Y. Kim, S. Kim, and Y. Seo, "Multi-Layer Bit-Sliced Bit-Rate Scalable Audio Coding", *103rd AES Convention, preprint 4520*, New York, September 1997.
8. M. Jelínek, T. Vaillancourt, A. Erdem Ertan, J. Stachurski, A. Rämö, L. Laaksonen, J. Gibbs, and S. Bruhn, "ITU-T G.EV-VBR Baseline Codec", *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2008)*, Las Vegas, USA, March 30 – April 4, 2008.
9. C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 Still Image Coding System: An Overview", *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103-1127, November 2000.
10. C. Timmerer and H. Hellwagner, "Interoperable Adaptive Multimedia Communication", *IEEE Multimedia Magazine*, vol. 12, no. 1, pp. 74-79, January-March 2005.
11. C. Timmerer, G. Panis, and E. Delfosse, "Piece-wise Multimedia Content Adaptation in Streaming and Constrained Environments", *Proceedings of the 6th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2005)*, Montreux, Switzerland, April 2005.
12. C. Timmerer, T. Frank, and H. Hellwagner, "Efficient processing of MPEG-21 metadata in the binary domain", *Proceedings of SPIE International Symposium ITCom*

2005 on Multimedia Systems and Applications VIII, Boston, Massachusetts, USA, October 2005.
13. M. Ransburg, C. Timmerer, H. Hellwagner, and S. Devillers, "Processing and Delivery of Multimedia Metadata for Multimedia Content Streaming", *Proceedings of the Workshop Multimedia Semantics - The Role of Metadata*, RWTH Aachen, March 2007.
14. M. Ransburg, H. Gressl, and H. Hellwagner, "Efficient Transformation of MPEG-21 Metadata for Codec-agnostic Adaptation in Real-time Streaming Scenarios", *Proceedings of the 9th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2008)*, Klagenfurt, Austria, May 2008.
15. D. Hong and A. Eleftheriadis, "XFlavor: Bridging Bits and Objects in Media Representation", *Proceedings IEEE International Conference on Multimedia and Expo (ICME)*, Lausanne, Switzerland, pp. 773- 776, August 2002.
16. M. Amielh and S. Devillers, "Bitstream Syntax Description Language: Application of XML-Schema to Multimedia Content", *11th International World Wide Web Conference (WWW 2002)*, Honolulu, May, 2002.
17. 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", *Signal Processing: Image Communication*, vol. 18, no. 8, pp. 721-747, September 2003.
18. C. Timmerer, G. Panis, H. Kosch, J. Heuer, H. Hellwagner, and A. Hutter, "Coding format independent multimedia content adaptation using XML", *Proceedings of SPIE International Symposium ITCom 2003 on Internet Multimedia Management Systems IV*, Orlando, Florida, USA, pp. 92-103, September 2003.
19. W. De Neve, D. Van Deursen, D. De Schrijver, S. Lerouge, K. De Wolf, and R. Van de Walle, "BFlavor: A harmonized approach to media resource adaptation, inspired by MPEG-21 BSDL and XFlavor", *Signal Processing: Image Communication*, vol. 21, no. 10, pp. 862-889, November 2006.
20. ISO/IEC 21000-7:2007, *Information technology – Multimedia framework (MPEG-21) – Part 7: Digital Item Adaptation*, 2007.
21. M. Ransburg, C. Timmerer, and H. Hellwagner, "Dynamic and Distributed Multimedia Content Adaptation based on the MPEG-21 Multimedia Framework", in: Mathias Lux et al., Eds., *Multimedia Semantics: The Role of Metadata*, Springer, March 2008.
22. ISO/IEC 23001-1:2006, *Information technology – MPEG system technologies – Part 1: Binary MPEG Format for XML*, 2006.