Design and Evaluation of a Metadata-Driven Adaptation Node[1]

Michael Ransburg, Christian Timmerer, Hermann Hellwagner
ITEC, Klagenfurt University
<firstname>.<lastname>@itec.uni-klu.ac.at

Sylvain Devillers
France Telecom R&D
sylvain.devillers@orange-ftgroup.com

Abstract

MPEG-21 Digital Item Adaptation (DIA) allows for a media codec agnostic multimedia adaptation approach which enables the implementation of generic adaptation engines. However, DIA is optimized for static, server-based adaptation. In this paper we introduce novel mechanisms to extend the DIA approach towards dynamic and distributed scenarios. This facilitates the placement of generic adaptation nodes which perform media codec agnostic and dynamic adaptation anywhere along the content delivery path. To validate our work we implemented such an adaptation node and evaluate its performance.

1. Motivation and State of the Art

Today's Internet is accessible to diverse end devices through a high variety of network types. Independent from this huge amount of usage contexts, content consumers desire to retrieve content with the best possible supported quality. The designers of new media codecs react to this diversity of usage contexts by including adaptation support into the codec design. These scalable codecs support the generation of a degraded version of the original bitstream by means of simply removing bitstream segments. All these variables (different end devices, network types, user preferences, media codec types, scalability options) lead to a manifold of needed and possible adaptation operations. In order to encounter this complexity, DIA [1] specifies a set of description tools (and related processes) in order to describe the media content, the adaptation possibilities and the usage context in the XML domain, which enables media codec agnostic adaptation as described in [2]. The relevant description tools are: 1) The Bitstream Syntax Description (BSD), which uses a generic language to describe, for instance, the parts of a media content which may be removed for scalability purposes. 2) The Adaptation Quality of Service Description (AQoS), which describes how (segments of) a media content need(s) to be adapted in order to correspond to the various usage contexts, e.g., how many quality layers need to be dropped to correspond to the currently available network bandwidth. 3) The Usage Environment Descriptions (UEDs) which describe the usage context, e.g., the available network bandwidth.

The content-related metadata (BSD and AQoS) is generated during the media encoding process and bundled together with the media as a Digital Item as depicted on the left side of Figure 2. A content creator (e.g., a news agency) can sell this Digital Item to a content provider (e.g., an Internet portal) which offers a very high quality version of the media content, together with the content-related metadata which describes how to extract lower quality versions. The content creator no longer needs to provide several versions of the content and the content provider no longer needs to keep many different versions of the content on its servers.

While the DIA framework fits our problem scope, it still comes with several limitations: The content-related metadata can be of considerable size (in the uncompressed domain sometimes as large as the described media content), depending on the bitstream syntax and the supported adaptation options. Furthermore, with the current set of description tools, any adaptation which is performed always impacts the complete media content, because the content-related metadata describes the complete media content. In streaming scenarios this leads to 1) high memory requirements, 2) a high start-up delay and 3) slow reaction in case of a dynamically changing usage context due to the need to parse the complete content-related metadata into memory for the adaptation. Additional startup delay occurs when multiple adaptation steps may take place along the delivery chain, since the complete content-related metadata needs to be transported in advance. The current mechanisms therefore only allow efficient application of DIA in "download and play" scenarios and not in streaming scenarios. Processing and delivering media in a streamed (i.e., dynamic) fashion has many advantages, e.g., minimized start up delay, and is

[1] Work partly supported by the IST European projects DANAE (IST-1-507113) and ENTHRONE (n° 038463)

therefore commonly used to consume large media files over the Internet. Additionally, server-centric adaptation is only ideal if the problem which adaptation tries to encounter 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 be disastrous to the quality of experience for the end user.

We therefore aim to extend the static, server-based DIA mechanism to dynamic and distributed adaptation of streaming media, while staying backwards-compatible with the original mechanisms, i.e., this extension shall sustain all of the benefits (in particular being media codec agnostic) of the static mechanisms. Additionally, an adaptation node which implements this extended mechanism shall be efficient enough to support concurrent media codec agnostic adaptations of several media streams.

2. Proposed Enhancements

This section describes our proposed enhancements, some of which (cf. Section 2.1) are currently considered for standardization in [3] to extend the DIA framework towards dynamic and distributed adaptation, as depicted in Figure 2. The enhancements can be categorized in three parts: 1) fragmentation and timing of metadata, 2) encoding of metadata, and 3) transport of metadata.

Please note that for completeness we also show the media streaming architecture on the bottom of Figure 2, which is implemented using existing solutions (e.g., hint tracks) and will therefore not be explained further.

2.1. Fragmentation and Timing of Metadata

Our enhancements in this area are based on an insight which we gained when analyzing the DIA framework: The processes, i.e., Adaptation Decision Taking Engine (ADTE), XML Transformation and BSDtoBin, in a DIA framework can principally be re-used (without changes) for dynamic adaptation, as they are all completely steered by metadata (e.g., AQoS, BSD). The same process which is applied to the whole media content [2], can be applied to a media segment (e.g., an access unit - AU) if the metadata is designed appropriately. As in DIA, whenever the usage context changes during the real-time streaming session, the ADTE needs to take a new adaptation decision / parameter (based on AQoS and UEDs), which is valid until the next usage context change occurs. This adaptation decision is then the input to the XML Transformation process, together with the BSD. The transformed BSD then steers the BSDtoBin process in order to adapt the media bitstream. The BSD and AQoS, however, shall only describe the current media segment which is about to be streamed out, rather than the complete content (as it would be the case with the static mechanism).

This requires that the content creator has a means to describe the fragmentation of the content-related metadata. There is also a need to describe the timing of the metadata in order to enable its synchronized delivery and processing with the media segments.

To this end, we introduce *XML Streaming Instructions (XSI)*, which provide the information required for streaming an XML document by the composition and timing of well-formed XML fragments which can be consumed as such by the intended processes (i.e., ADTE, XML Transformation, BSDtoBin). These fragments are called *Process Units (PUs)* and their definition of "being able to be consumed as such" ensures the backwards compatibility of our approach to the static DIA mechanism.

Figure 2 shows that the Fragment process takes as input the XML document to be streamed and a set of XSIs provided as XML attributes with their own namespace. The output of the Fragment process is a set of timed PUs. A PU is specified by one element tagged as *anchorElement* and by a *puMode* indicating how other connected elements are aggregated to this anchor to compose the PU. Several *puModes* are necessary, since the structure of the content-related metadata depends on the high-level structure of the media content. Figure 1 gives an overview of the currently defined *puModes* which we derived by analyzing the high-level structure of a variety of scalable media codecs [4][5][6]. The white node represents the anchor element and depending on the *puMode* several more nodes are included to compose a PU. It is important to note that PUs may overlap, i.e. some elements (including anchor elements) may belong to several PUs in order to meet the requirement of being able to be processed as such.

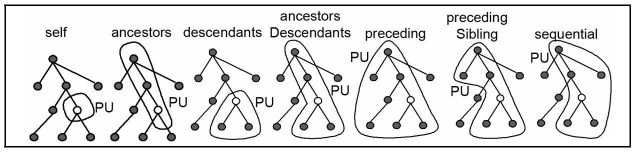

Figure 1: Examples for the different puModes

There are several more attributes which need to be assigned to each PU for streamed processing: The *encodeAsRAP* attribute is used to signal that the PU should be encoded as a random access point (RAP) in order to enable random access into an XML stream. The *timeScale* attribute provides the number of ticks

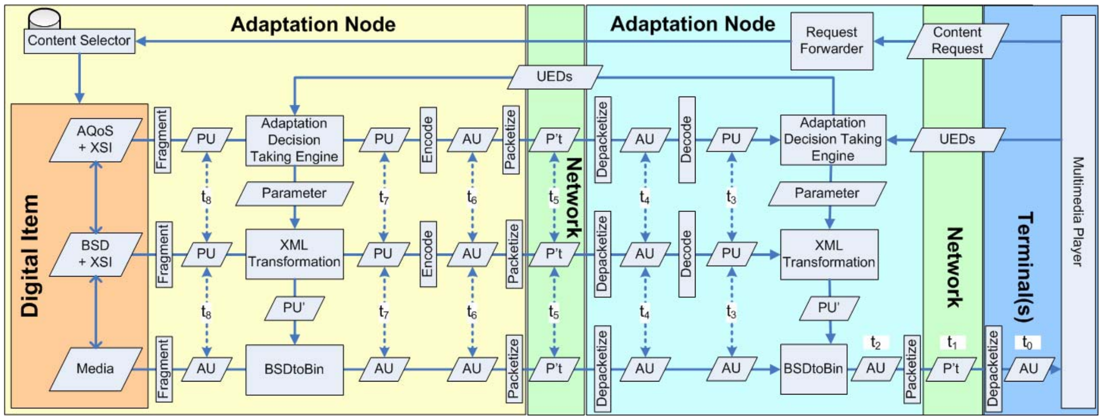

Figure 2: A dynamic and distributed system for media codec agnostic multimedia adaptation

per second. The *ptsDelta* attribute specifies the interval in time ticks after the preceding anchor element. Alternatively, the *pts* attribute specifies the absolute time of the anchor element as the number of ticks since the origin. The timing can not only be specified in ticks: the *absTime* attribute specifies the absolute time of the anchor element. Its syntax and semantics are specified according to the time scheme used (*absTimeScheme* attribute), e.g., NPT or UTC.

2.2. Encoding of Metadata

In order to enable the adaptation mechanism to be applied not only at the server, but anywhere along the delivery chain, the PUs need to be streamed together with the media data. As the PUs are still in the text-domain, which is quite verbose, we investigated several mechanisms to enable their efficient transport.

We differentiate three types of encoding / compression algorithms: 1) General redundancy-based algorithms have no knowledge about XML. For XML-conscious algorithms there are two types: 2) XML-syntax aware algorithms and 3) XML-schema aware algorithms. In previous research [7][8] we selected one candidate from each of these types for our measurements: WinZip uses a hybrid of LZ77 and Huffman coding algorithm which belongs to the first type. XMLPPM uses an XML-syntax aware algorithm and BiM uses an XML-schema aware algorithm. BiM has built-in streaming support: Each BiM AU only includes the changes to the previous PU. The *encodeAsRAP* attribute can be used to steer BiM to encode a PU as an independent BiM AU for random access at specific intervals. For all PUs between these random access points only the changes are encoded into AUs. BSDs use the majority of metadata bandwidth and therefore we restrict our investigations to them. Our results [7][8] demonstrate that BiM is superior to the other mechanisms. As a result of these evaluations, the Encode process (as depicted in the center of Figure 2) implements BiM as an encoding mechanism for our content-related metadata.

2.3. Transport of Metadata

After encoding the PUs into AUs, the media and metadata AUs are packetized into packets (P't in the center of Figure 2) for transport. We are using dedicated RTP (IETF RFC 3550) streams suggested by our previous research in this area [8]. In this step the timing and RAP information provided by the XSIs is used to steer the packetization process, e.g., by including them into the RTP packet header. The XSIs can be removed from the PUs for transport, in order to save bandwidth. As depicted in the center of Figure 2 both the media and metadata AUs are then streamed into the network, where an adaptation node can perform additional adaptation steps, in the same way as it is performed on the server side.

3. Evaluation

In order to validate our work, the system described in Section 2 and depicted in Figure 2 was implemented in C++ in the course of the EC IST FP6 project DANAE[1] (IST-1-507113). Darwin Streaming Server[2] was used as a streamer and the modules shown in Figure 2 were implemented as plug-ins. The libxml XMLTextReader interface[3] was used for processing XML. In our measurements we evaluate if our prototype implementation of a dynamic DIA adaptation node can be utilized in a real-time streaming scenario. To this end, we evaluate the CPU and memory efficiency of the adaptation node in Figure 2. All tests were performed on a Dell Optiplex

[2] http://developer.apple.com/opensource/server/streaming
[3] http://xmlsoft.org
[4] http://www.tiler.com/FreeMeter

GX620 desktop with an Intel Pentium D 2.8 GHz processor and 1024 MB of RAM using Windows XP SP2 as an operating system. Time measurements were performed using the ANSI-C clock method. CPU and memory efficiency was evaluated using the FreeMeter Professional logging software[4].

Table 1 provides an overview of the test data. Media and the corresponding BSDs for three different media codecs were selected. MPEG-4 BSAC [5] is a scalable audio codec, EZBC [6] is a scalable video codec based on wavelets and MPEG-4 SVC [4] is a scalable video codec based on conventional block transforms which is currently being standardized in MPEG.

Table 1: Characteristics of test data

	BSAC	EZBC	SVC
Media Size	12511 KB	450536 KB	538816 KB
Average AU Size	0,22 KB	197,86 KB	18,59 KB
BSD Size	196265 KB	144939 KB	123189 KB
Average PU Size	4,02 KB	63,80 KB	4,90 KB
Number of [A|P]Us	56100	2277	28980
Resolution	N/A	QCIF	QCIF
Frame Rate	21	12,5	12,5
Length in Minutes	44,52	48,58	193,2

Table 2 shows measurements of the memory utilization and CPU load of the adaptation node. For this evaluation we assume a static AQoS. We request a DI (consisting of media content and the corresponding BSD) to the Request Forwarder process which initializes the Adaptation Node and forwards the request to the Content Selector, as depicted in Figure 2. At the server-side Adaptation Node, 1) BSD PUs are composed and timed, 2) media AUs are adapted, 3) metadata is encoded and 4) media and metadata is packetized and streamed into the network. This is repeated until there are 5 DIs (i.e., 10 streams) being processed concurrently. One particular result which we derived from the measurements is that the CPU load strongly depends on the frame rate, e.g., the BSAC stream with a frame rate of 21 and small PUs causes much more CPU load than the EZBC with a frame rate of 12,5 and large PUs. As can be seen from the measurements, the adaptation node would be able to support several more content streams (or contents with a higher bitrate). With our enhancements the start-up / adaptation delay averages at 6 seconds. This is mostly due to static buffers being used on the player and on the adaptation nodes and can be reduced by optimizing these buffers, e.g., by adjusting their size based on the average size of the [A|P]U. Without our enhancements we measure a delay of 70 seconds (BSAC), 208 seconds (EZBC) and 103 seconds (SVC) at an available bandwidth of 1 Mbps.

Table 2: CPU load in percent and memory usage in MB for processing several streams

	Number of Streams				
	1	**2**	**3**	**4**	**5**
BSAC (CPU in %)	6,38	16,1	27,03	33,77	47,69
EZBC (CPU in %)	1,21	5	6,23	9,95	13
SVC (CPU in %)	0,62	2,67	2,87	4,79	5,62
BSAC (Memory in MB)	14,12	14,92	15,94	16,69	17,73
EZBC (Memory in MB)	15,01	17,4	20,53	21,1	22,06
SVC (Memory in MB)	15,76	16,76	17,83	18,47	19,19

4. Conclusion and Future Work

In this paper we showed that static, server-centric DIA-based adaptation can be efficiently extended towards dynamic and distributed application scenarios in a backwards-compatible way. To this end, we introduced a new language which enables the streaming of an XML document by the composition and timing of well-formed XML fragments which can be consumed as such. This enabled us to implement a media codec agnostic adaptation node, which may be located anywhere along the content delivery path. We then evaluated this prototype and showed that it is efficient enough to support the concurrent adaptation of several media streams on a standard computer.

The current synchronization mechanism relies on a one-to-one relationship between media AUs and PUs. Since PUs are usually much smaller than media AUs this leads to processing (e.g., CPU) overhead. A more flexible mechanism will be investigated.

5. References

[1] ISO/IEC 21000-7:2004: Digital Item Adaptation

[2] A. Vetro, "MPEG-21 Digital Item Adaptation: Enabling Universal Multimedia Access", IEEE Multimedia, pp. 84-87, January 2004

[3] ISO/IEC 2100-7:2004/FPDAmd 2: Dynamic and Distributed Adaptation, 2006

[4] H. Schwarz, D. Marpe, T. Wiegand, "Overview of the Scalable H.264/MPEG4-AVC Extension", ICIP, October 2006

[5] H. Purnhagen, "An Overview of MPEG-4 Audio Version 2", AES, September 1999

[6] S.-T. Hsiang, J. W. Woods, "Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling", DCC, May 2000

[7] C. Timmerer, I. Kofler, J. Liegl, H. Hellwagner, "An evaluation of Existing metadata compression and encoding technologies for MPEG-21 applications", ISM, 2005

[8] M. Ransburg, C. Timmerer, H. Hellwagner, “Transport Mechanisms for Metadata-driven Distributed Multimedia Adaptation”, MSAN, 2005