Efficient Processing of MPEG-21 Metadata in the Binary Domain

Christian Timmerer[a], Thomas Frank[a], Hermann Hellwagner[a], Jörg Heuer[b], and Andreas Hutter[b]

[a] Klagenfurt University, Department of Information Technology , A-9020 Klagenfurt, Austria
[b] Siemens AG, Corporate Technology IC 2, D-81730 Munich, Germany

Department of Information Technology (ITEC)
Klagenfurt University
Technical Report No. TR/ITEC/05/1.10
August 2005

Efficient Processing of MPEG-21 Metadata in the Binary Domain

Christian Timmerer[*a], Thomas Frank[a], Hermann Hellwagner[a], Jörg Heuer[b], and Andreas Hutter[b]
[a] Klagenfurt University, Department of Information Technology , A-9020 Klagenfurt, Austria
[b] Siemens AG, Corporate Technology IC 2, D-81730 Munich, Germany

ABSTRACT

XML-based metadata is widely adopted across the different communities and plenty of commercial and open source tools for processing and transforming are available on the market. However, all of these tools have one thing in common: they operate on plain text encoded metadata which may become a burden in constrained and streaming environments, i.e., when metadata needs to be processed together with multimedia content on the fly. In this paper we present an efficient approach for transforming such kind of metadata which are encoded using MPEG's Binary Format for Metadata (BiM) without additional en-/decoding overheads, i.e., within the binary domain. Therefore, we have developed an event-based push parser for BiM encoded metadata which transforms the metadata by a limited set of processing instructions – based on traditional XML transformation techniques – operating on bit patterns instead of cost-intensive string comparisons.

Keywords: universal multimedia access, multimedia adaptation, compressed-domain metadata processing, bitstream adaptation in constrained and streaming environments, MPEG-21, Digital Item Adaptation, generic Bitstream Syntax Description

1. INTRODUCTION AND MOTIVATION

Today’s multimedia communication is inconceivable without the support of metadata to help managing the multitude of possible usage scenarios that are enabled by the high variety of devices accessing multimedia content through a plethora of heterogeneous networks. Furthermore, the diverse set of coding formats, user characteristics, and user preferences needs to be taken into account in order to enable Universal Multimedia Access (UMA)[1,2] for the end user. For UMA, metadata is playing a key role in supporting seamless access to any type of (multimedia) content anywhere and anytime. UMA has become a driving concept behind a significant amount of research and standardization activities. One of MPEG’s (Moving Picture Experts Group) most recent responses to the UMA challenges is MPEG-21 Digital Item Adaptation (DIA).

MPEG-21 provides a multimedia framework enabling the interaction of Users by means of so-called Digital Items, i.e., the Digital Item is the object of interaction[3]. Therefore, MPEG-21 introduces key concepts like User (please note the upper case “U”) and Digital Item. A Digital Item is the fundamental unit of transaction within the MPEG-21 framework and aggregates multimedia resources together with metadata, licences, identifiers, intellectual property management and protection (IPMP) information, and methods. Users interacting with Digital Items include individuals as well as communities or organizations. It is important to note that Users are not restricted to humans; a User within the MPEG-21 framework may also include intelligent software modules such as agents. Hence, in this paper the term “user” refers to a (human) end user and “User” refers to the broader notion as defined in MPEG-21.

The part of MPEG-21 that is essential for UMA is part 7, entitled Digital Item Adaptation (DIA)[4]. DIA aims to achieve interoperable transparent access to (distributed) advanced multimedia content by shielding users from network and terminal installation, configuration, management, and implementation issues. In order to achieve this goal, the adaptation of Digital Items is required. Therefore, DIA provides tools, i.e., normative description elements based on XML, supporting the adaptation process of Digital Items – including audio-visual multimedia resources – in a device and coding format independent way. Additionally, it has been demonstrated that these tools can be used within constrained and streaming environments[5]. In particular, DIA specifies – among others – an XML-based metadata format enabling the description of the high-level multimedia bitstream syntax in a generic way, i.e., in terms of packets, layers, headers, etc. Such a description is referred to as generic Bitstream Syntax Description (gBSD) which can be transformed according to the parameters extracted from the context in which the multimedia content is intended to be

[*] christian.timmerer@itec.uni-klu.ac.at; phone +43 (463) 2700 3621; fax +43 (463) 2700 3699; www.itec.uni-klu.ac.at

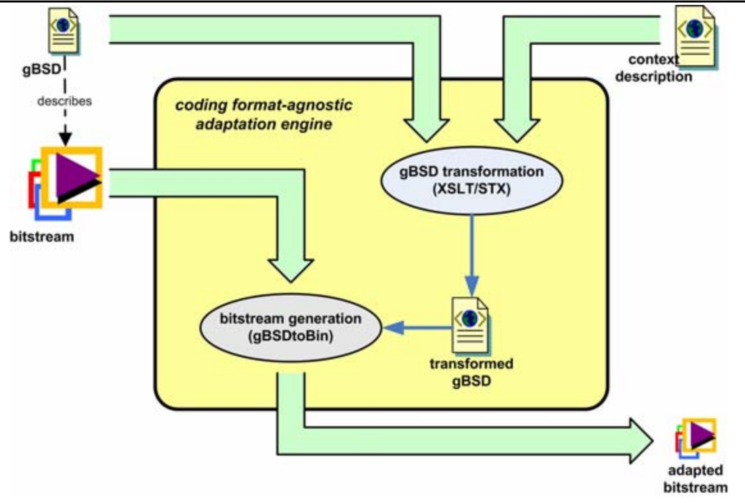
gBSD
describes
bitstream
coding format-agnostic adaptation engine
context description
gBSD transformation (XSLT/STX)
bitstream generation (gBSDtoBin)
transformed gBSD
adapted bitstream

Figure 1 — High-level architecture of a coding format agnostic adaptation engine.

consumed. Note that the context description is also an integral part of DIA including four major categories, namely terminal capabilities and network conditions/capabilities as well as user and natural environment characteristics. The transformed gBSD controls a generic processor – the behaviour is defined by the MPEG-21 DIA standard – which adapts a given bitstream without having knowledge about the coding format of the original multimedia bitstream. As such, this approach enables the construction of coding format agnostic multimedia adaptation engines. The high-level architecture of such an engine is depicted in Figure 1.

Multimedia content is usually encoded using highly efficient compression techniques and real-time processing thereof can be achieved as well. Most coding schemes provide certain scalability dimensions, e.g., spatial, temporal, or signal-to-noise ratio, enabling adaptation by means of truncating the bitstream, in some cases followed by minor update operations. Additionally, truly scalable coding techniques[6] are becoming available. All these operations do not require the decoding of the bitstream.

Metadata, on the other hand, is mainly XML-based and encoded in plain text. When metadata is transported together with the multimedia content, bandwidth requirements are increasing which is unacceptable in constrained and streaming environments. Furthermore, end users do not understand why they have to pay for metadata they do not see even though these are required for an improved multimedia experience. Consequently, the metadata overhead needs to be reduced by means of appropriate encoding schemes and processing thereof needs to be as efficient as for the corresponding media data, i.e., within the compressed/encoded (binary) domain. A first step towards processing metadata within the binary domain is based on intelligent fragmentation techniques[7] which utilize MPEG's Binary Format for Metadata (BiM) exploiting special encoder configuration settings to enable the filtering and updating of XML fragments within the binary domain. However, this approach has several drawbacks, e.g., for complex transformations the compression performance decreases due to the required fragmentation as briefly described in Section 2.

In this paper we present enhanced binary encoding strategies that efficiently support the metadata filtering and processing in the binary domain while preserving efficiency in terms of metadata overhead. A BiM payload parser is suggested that does not rely on a specific fragmentation of the XML data. To achieve this, BiM needs to be extended by binary encoding optimizations like tokenization of string values and skipping information[8]. Based on this parser, a BiM transformer is responsible for the adaptation of the binary metadata. XML transformation templates that are currently used in the uncompressed domain are translated into processing instructions and are passed to the BiM transformer.

The remainder of this paper is organized as follows. Section 2 describes related work and Section 3 provides background information on MPEG's approach of binary encoding of metadata and its native support for binary filtering. Enhanced binary encoding strategies are presented in Section 4. The new event-based binary XML transformation technique based on so-called processing instructions is described in Sections 5 and 6, respectively. Experimental results are presented in Section 7 and Section 8 concludes the paper.

2. RELATED WORK

Traditional XML transformation tools have their focus on transforming only plain text encoded XML descriptions. XSLT[9] is its most popular representative which has the status of a W3C recommendation which currently advances to version 2.0 and many implementations and tools are available. One major drawback of XSLT is that the complete

XML description must be in memory before being processed which is a burden in streaming scenarios. STX[10] which is based on SAX (Simple API for XML) events seems to be a promising candidate to overcome this burden but still operates on plain text XML. Tools like FXT (Functional XML Transformation)[11], XDuce[12], and HaXml[13] are not that established but operate on plain text as well.

Further related work can be found in another paper[8] which provides an evaluation of several binary XML encoding optimizations, i.e., alternative serialization format, tokenization, and skip-to pointers which are also introduced in this paper. However, this evaluation concentrates on parsing performance only and has shown that such optimization facilitate a performance gain up to a factor of six (for parsing only) – depending on the document structure and the required information – compared to the fasted XML parser they were aware of (i.e., xpat[14]).

Besides the binary XML efforts within MPEG, ISO/IEC has put some joint efforts with ITU-T towards an alternative XML serialization within the ASN.1 (Abstract Syntax Notation One) group. Therefore, mapping rules between XML schemas and ASN.1 schemas are defined[15,16] and for ASN.1 instances efficient binary encoding schemes such as Packed Encoding Rules (PER)[17] or Encoding Control Notation (ECN)[18] are available. In practice, however, no common API capable of handling both types of data (i.e., XML and ASN.1) is available and transformation between the two representations is uneconomical.

In addition to the above mentioned efforts, the Web service community has also recognized the verbosity of XML as an issue and is currently developing alternative XML serialization schemes known as Fast Infoset[19] and Fast Web Services[20] which are built upon ASN.1 as described above. Therefore, an indexing mechanism which associates an index to each XML element enabling its usage for further occurrences of the same XML element, i.e., highly repetitive content will benefit from this approach. However, for small and complex XML documents the index table would introduce additional redundancy and is not applicable in streaming scenarios.

3. BACKGROUND

A promising candidate for binary encoding of XML-based metadata which (partially) fulfils the aforementioned requirements is MPEG's Binary Format for Metadata (BiM)[21,22,23]. Following several experimental investigations within the MPEG-21standardization process, BiM has been selected as the binary format for MPEG-21 metadata and is now specified in part 16 of MPEG-21[24]. BiM has been recently moved from MPEG-7 to the first part of a new MPEG standard named MPEG-B which is dedicated to generic media tools which are heavily used across MPEG standards.

BiM is an XML Schema aware encoding scheme for XML documents, i.e., it uses information from the XML Schema to create an efficient (compressed) serialization of XML documents within the binary domain. This schema knowledge enables the removal of structural redundancy, e.g., element and attribute names known from the XML Schema definition, which achieves high compression ratios with respect to the document structure. Furthermore, element and attribute names as well as data are encoded using dedicated codecs mapped to selected data types (e.g., integer, float, string) which further increases the compression ratio. Additionally, one of the main features of BiM is that it provides streaming capabilities for XML-based data which is not natively supported by XML. Therefore, BiM enables the fragmentation of the XML tree into access units (AUs) containing one or more fragment update units (FUUs). Each FUU includes the FU command, FU context, and FU payload which are described briefly in the following:

- The FU command specifies the decoder action for the corresponding fragment which can be either add, delete, replace, or reset, i.e., BiM also provides partial updates of an XML document on the receiver side.
- The FU context is used to uniquely determine the location of the signalled fragment in the XML document.
- The FU payload contains the actual XML fragment data for the node addressed by the FU context.

BiM provides a degree of native support for binary filtering at the FUU level by exploiting the structure of the FU context which is represented as a path. This path is divided into a list of node names and a list of position codes separated by a unique path termination code. The representation of the context supports the filtering for certain context paths independent of their position by bit pattern matches of the node name representations. For example, an XML path expression like `/el1/el2/el3` can be matched against the context path of an FUU by using only the first part of the FU context, i.e., regardless of its actual position within the complete XML document.

For transformations of metadata as illustrated in Figure 1 not only the context of fragments is relevant but also values of attributes and elements which trigger transformation operations. Thus, a coding scheme[7] can be configured in which each FUU is actually split in at least two FUUs: in a first FUU which only contains values for testing conditions of transformation operations and in a second FUU (or even more) which contains values which are transformed. This fragmentation is increased to support complex condition tests and operations though. The gain of flexibility in the transformation on the other hand results in a drawback of efficiency: redundancy is introduced in form of context information and the usage of the Zlib encoding scheme[25] – which is part of BiM version 2[23] –considerably decreases the compression ratio for small fragments. Thus, in this paper we describe a more comprehensive approach based on enhanced binary encoding strategies and event-based binary XML transformations which helps to overcome these issues.

4. ENHANCED BINARY ENCODING STRATEGIES

MPEG's BiM already supports various encoding strategies and provides an extension mechanism which enables the integration of external or third party encoding schemes. Some of them have been proven and validated as very useful and are now an integral part of the standard, e.g., the usage of the Zlib encoding scheme for string values has been adopted within the advanced optimization schemes of BiM version 2. However, it is optional on the encoder side to configure the use of Zlib.

In this section we present how enhanced binary encoding strategies such as tokenization of string values and introduction of skipping information are applied to BiM. Note that MPEG-21 metadata such as gBSDs could extremely benefit form such enhancements due to its data characteristics, i.e., recurring label and marker tokens. Furthermore, it has been already demonstrated that tokenization and skipping information increases the parsing speed of XML documents up to a factor of six[8] but not specifically in the context of BiM.

4.1. Tokenization

The main obstacle for a fragmentation independent processing of BiM encoded XML content is the way of coding string values. BiM provides two default codecs for string values in the two versions of the standard. In BiM version 1 string values are encoded uncompressed – UTF-8 string preceded by its size in bytes – between structural information and values with different data types (e.g., boolean or integer values) as depicted in Figure 2.

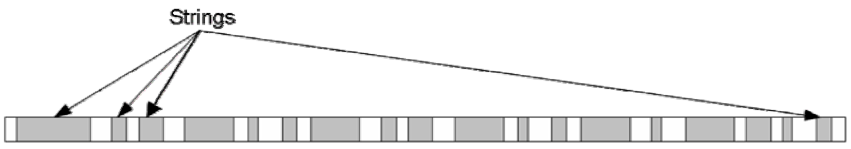

Figure 2 — Coding of string values in BiM version 1[22].

BiM version 2 has adopted the Zlib encoding scheme which normally provides increased compression efficiency – except for small fragments as mentioned above. During encoding, string values are gathered in a fixed-size buffer which is compressed when the buffer is full or end-of-payload is reached. The resulting compressed chunk of data is placed at the expected position of the first string within the resulting bitstream. Figure 2 shows a BiM bitstream compliant to version 1, i.e., without the Zlib codec. String values (in grey) are distributed over the entire bitstream. Figure 3 illustrates the two phases of the Zlib encoding process of BiM version 2. The first phase, i.e., buffering, gathers the string values into a fixed-size buffer. The second phase, i.e., compression, applies the Zlib algorithm to the buffer from phase one which may contain several string values. The compressed Zlib buffer can be found at the position of the first string in this buffer. This position is exploited during the decoding process, i.e., all string values of this Zlib chunk are decompressed into the Zlib buffer. As such, string values are becoming available to the application for consumption through the Zlib buffer. If the Zlib buffer is empty, the next chunk is decompressed into the buffer.

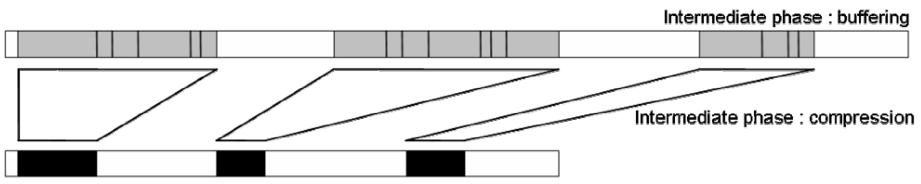
Intermediate phase : buffering
Intermediate phase : compression

Figure 3 — Coding of string values in BiM version 2[23].

This Zlib codec prohibits the processing of BiM data within the binary domain without decompressing and decoding all Zlib chunks. The position of the next Zlib chunk in the BiM stream is not known until the last string of the current decompressed chunk is consumed. Even if none of these string values is needed all chunks have to be read and decoded to be able to read the remainder of the bitstream correctly.

However, BiM provides the possibility to introduce special codecs for defined XML types. In order to be able to support processing BiM data in the binary domain, a new codec for attribute and element content of string type using tokenization has to be introduced. Tokenization is a powerful concept for compression since recurring strings need not be repeated within the binary data. Instead of encoding the string value itself several times, a much smaller token identifier can be used to point to the value at each occurrence. The application performance can also be improved by using tokenization, i.e., expensive string comparisons can be replaced by just comparing the token values – typically a byte or an integer.

The proposed way for a mapping of string values and tokens is a string table that has to be built up for each fragment during the encoding phase. Strings have to be inserted into the table and the corresponding token has to be encoded. If a new string is already part of the table, it is not inserted again. The token of the already inserted string will be encoded. The table – which can be Zlib compressed – can be written to the beginning of the payload (see Figure 4) or just before the first occurrence of a token value within the payload. As such, it can be treated as a special BiM type codec without the need of an extension of the BiM format.

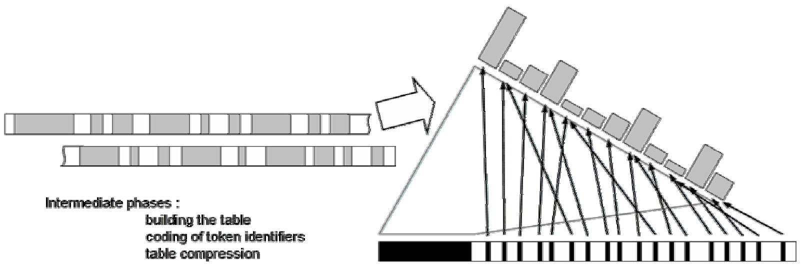

Figure 4 — Coding of string values using tokenization.

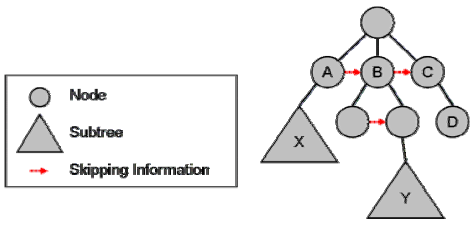

Figure 5 – Usage of skipping information.

The decoder reads the table and decompresses it, if needed. The rest of the payload can be processed in the binary domain. If it is known that string values are of no relevance for further processing the document, the decompression of the table can be skipped or the decompression process can be deferred to the moment when the first string value is needed.

4.2. Skipping Information

An enhancement in terms of processing performance is adding pointers from one node within the tree to its next sibling node. By adding this skipping information a random access within the payload can be simulated and a full depth-first traversal of the whole tree is not required. This can be helpful if only parts of the document are of relevance for the further processing and especially if this is the case with deeply nested data. Figure 5 shows the usage of this skipping information. If the required information can be found in node D and in node A and B it can be decided that there is no relevant information in the corresponding sub-trees, these nodes can be skipped and the processing can be continued at node C. A traversal of the whole tree including sub-trees X and Y is not needed.

This sibling offset is already part of BiM but its existence needs to be signalled by a special flag at the beginning of each payload. With skipping information, the encoder has to be configured in a way that this length information has to be mandatory to enable the processing application to use this enhancement.

5. EVENT-BASED BINARY XML PARSING

In general, event-based XML parsers implement the so-called push model where the parser reads the data stream and, for each XML token it encounters, an event is generated. The event contains implicit and explicit information about the token that was read. By using an event model, the parser pushes the information to the client application. A well-known and commonly used exponent of this parsing model is the simple application programming interface for XML, better known as SAX[26]. A parser compliant to the de-facto standard SAX enables simple access to the XML data through its content handler which receives the events from the underlying parser. Such events indicate the start/end of the document and elements respectively, attributes are collected in a special attribute list. In practice, however, these events provide only the string representation of the element or attribute names which are not suitable for binary encoded XML content. Furthermore, the current SAX specification does not take into account BiM specifics such as fragmentation into access or fragment update units. Therefore, we have developed the *BiM API for XML (BAX)* which basically relies on the same principles as SAX but provides full support for binary encoded XML data compliant to the BiM standard. In addition to the usual SAX events, a couple of new event types have been introduced taking into account the streaming functionality of BiM, i.e., events indicating the start/end of access units and fragment update units have been defined. It is important to note that all events provide data in binary form according to the BiM standard, i.e., no translation to the string representation is performed by the parser, which allows for fast XML processing and transformation within the binary domain by using simple bit manipulation techniques. Additionally, our parser supports the enhanced binary encoding techniques as described in Section 4.

The following BAX events are currently defined and briefly described in the subsequent paragraphs: start/end access unit (startAu, endAu), start/end fragment update unit (startFuu, endFuu), context path element (pathElement), start/end payload (startPayload, endPayload), start/end element (startElement, endElement), and content (content).

Start/end access unit (startAu, endAu). These events indicate the start and the end of BiM access units, respectively. Note that the *startAu* event is the first method being invoked before any other method within this interface. As such, these events become quite similar to the start/end document event from SAX. The *startAu* event contains one parameter providing the number of fragment update units within this access unit.

Start/end fragment update unit (startFuu, endFuu). These events are triggered due to the start and the end of a fragment update unit, respectively. The *startFuu* method contains three parameters, namely the length of the FUU, the FUU command, and the remaining parts of the context path (e.g., position codes) to which the FUU belongs. Note that the element nodes within the context path are provided by (several) *pathElement* events. Additionally, the *startFuu* method returns a Boolean value indicating whether this FUU can be skipped or not. This decision may be performed, for example, based on the context path (e.g., this element and its children are not required by the application) or FUU command (e.g., remove).

Context path element (pathElement). The path element event indicates an element within the context path of the fragment update unit according to the BiM standard. For elements within the FUU context path, no *startElement* and *endElement* will be received, but one *pathElement* event for each node in the context path. The parameters of this event identify the tokens representing the element name as well as the type name if the current element contains a type substitution indicated by an 'xsi:type' attribute. If no type substitution has been performed, a NULL pointer is passed instead of the token which indicates the type from the schema. The third parameter contains the binary information belonging to this element, e.g., the binary type substitution information. As for the *startFuu*, the *pathElement* returns a Boolean value indicating whether the current FUU is required or can be skipped.

Start/end payload (startPayload, endPayload). These events indicate the beginning and end of the payload, respectively. The *startPayload* event provides one parameter defining the decoding mode as specified in the BiM standard, i.e., some flags containing information about the length encoding, general settings for type substitution, etc.

Start/end element (startElement, endElement). The *startElement* determines the start of an element and provides additional information through its parameters. The first two parameters provide the binary tokens indicating the element name and, if needed, the substituted type name according to the schema. Subsequently, the next parameter provides structural information such as flags as well as decoding information for the choices and loops in the final state machine. Additionally, this parameter provides information about schema updates, and substitution groups. The length of the element is given by the fourth parameter. Finally, the fifth parameter contains the list of attributes ordered by name. Each attribute consists of an id, a type id, and its binary value. The id is the schema token that identifies the attribute name. The type id identifies the name of the type. The binary value is the BiM-encoded value of the attribute. Note that optional attributes are indicated by a NULL pointer if they are not encoded, i.e., not present.

Content (content). This event provides the binary content of an element. The schema token that identifies the type name is given by the first parameter and the actual BiM encoded content by the second parameter.

6. PROCESSING INSTRUCTIONS

MPEG-21 DIA based adaptation as introduced in Section 1 performs the bitstream adaptation via the XML domain by first transforming the corresponding XML document describing this bitstream. In most cases the adaptation of scalable bitstream formats can be accomplished by removing or truncating bitstream segments corresponding to specific scalability layers followed by possible minor editing operation such as updating certain parameters in the bitstream, e.g., the number of remaining layers. In some cases the insertion of bitstream segments – after prior removal of some segments – is required, e.g., in order to be compliant with video buffer verifier models[27]. Thus, the transformation of XML documents describing scalable bitstream formats can be classified into three major categories, namely remove, update, and insert operations:

- The *remove* operation deletes a certain element from an XML document. As a consequence, the corresponding bitstream segment is removed as well during the actual bitstream adaptation process. The element being removed is usually identified by an appropriate XML path expression within the XML transformation style sheet. Note that the remove operation on an element also applies to the child elements and attributes.

- The *update* operation alters the value of a certain parameter within the bitstream. The value which needs to be updated is identified by means of an XML path expression and the new parameter value is either provided by an XML path expression or a constant value.

- The *insert* operation includes a new XML element at the position identified by an XML path expression.

Note that the actual specification of these basic transformation instructions is rather abstract such that bindings to common XML transformation languages (e.g., XSLT, STX) can be defined. This is required due to syntactic and semantic differences of some components within these languages, e.g., XSLT uses XPath whereas STX has defined its own XML path language named STXPath which basically differs in the semantics of the context definition. The XSLT binding of the transformation instructions is shown in Document 1 – the STX binding is similar but is not shown here due to space limitations.

```
<xsl:stylesheet version=”1.0” xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.itec.uni-klu.ac.at/2005/stb" xmlns:stb="http://www.itec.uni-klu.ac.at/2005/stb">
  <xsl:template name="stb:remove">    <!-- #### REMOVE operation #### -->
    <xsl:param name="node"/>
  </xsl:template>
  <xsl:template name="stb:update">    <!-- #### REMOVE operation #### -->
    <xsl:param name="node"/>
    <xsl:param name="value"/>
    <xsl:value-of select="$value"/>
  </xsl:template>
  <xsl:template name="stb:insert">    <!-- #### REMOVE operation #### -->
    <xsl:param name="curNode"/>
    <xsl:param name="insNode"/>
    <xsl:copy-of select="$insNode"/>
  </xsl:template>
</xsl:stylesheet>
```

Document 1 — Generic XSLT templates for common gBSD transformation operations.

The basic operations `stb:remove`, `stb:update`, and `stb:insert` are defined in their own namespace – preceded with the `stb` prefix (streaming transformations for BiM) – which can be exploited by a customized style sheet processor providing additional information to subsequent processing steps as described in the following.

Traditional XML transformation techniques such as XSLT or STX perform the actual transformation by copying the remaining nodes or events from the source to the result structure. The removed elements simply disappear from the resulting set of nodes or events. In practice, however, the set of nodes or events being transformed is usually smaller than the set of remaining nodes or events resulting in a considerable performance gain of the overall adaptation process. The high-level adaptation architecture of such a streaming XML transformer is depicted in Figure 6. The BiM source stream is parsed with the BAX parser as described in Section 5 providing the source events as an input to the XML transformer. The style sheet itself is parsed into an internal memory structure and the templates (including the transformation instructions) are passed to the event transformer. The output of the event transformer is twofold. First, the result events are generated corresponding to the aforementioned remaining events which may be used for other purposes, e.g., for multi-step adaptations[28]. Second, transformed events are generated which may be used by a gBSDtoBin-like processor performing the actual bitstream adaptation. As such, the transformed events provide intrinsic support for combined processing in the XML transformer and the gBSDtoBin processor.

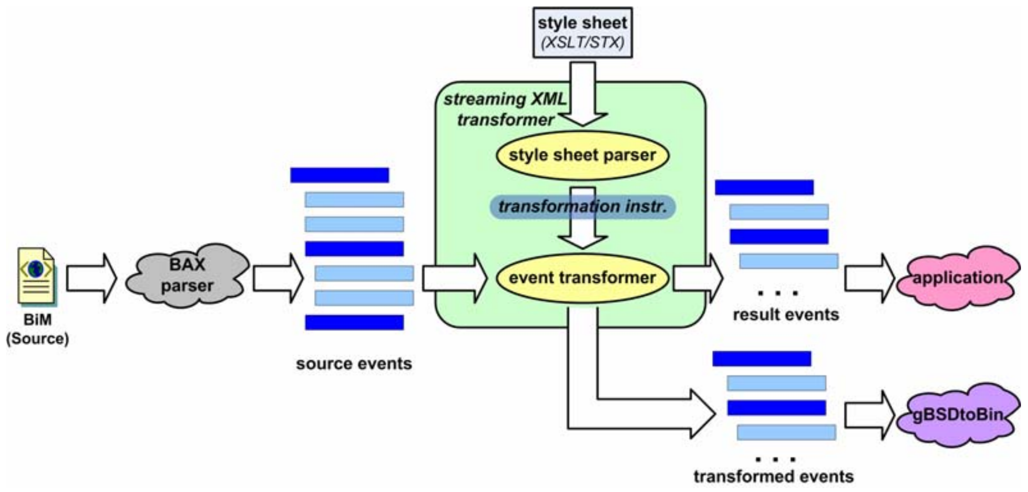

Figure 6 — Architecture of Streaming Transformer for BiM.

7. EXPERIMENTAL RESULTS

We conducted a series of experiments to measure and compare the performance of the different binary encoding strategies and XML transformation tools. In particular, we measured the compression ratio of the encoding strategies as formulated in Section 4 and compared the results to Zlib-enabled BiM encoding. Additionally, the XML transformation performance has been investigated in detail. All tests were performed on a 2.8 GHz P4 Mobile machine with 512 MB main memory and Windows XP service pack 2 installed.

For the compression ratio we have compared the original plain-text XML size with several encoding strategies, namely uncompressed strings (i.e., without Zlib), with Zlib encoding scheme enabled, our token-based string codec encoded as fixed-size 8 bit, and our token-based string codec encoded as vluimsbf5 (variable length unsigned integer most significant bit first 5bit, i.e., a variable length codec for unsigned integers defined within BiM). For the transformation performance we have considered MSXML version 4.0 Service Pack 2, Xalan-C version 1.9.0 (utilizing Xerces-C v2.6.0), and our streaming transformations for BiM (STB) processor version 0.0.1 which utilizes our BiM API for XML version 0.0.1.

The test data set comprises generic Bitstream Syntax Descriptions (gBSDs) describing audio (jm_gbsd.xml), video (akiyo_gbsd.xml, foreman_gbsd.xml), and image resources (city_gbsd.xml, shanghai_gbsd.xml). For audio the MPEG-4 Bit Sliced Arithmetic Coding (BSAC) codec is chosen providing fine-grained scalability through enhancement layers. The BSAC gBSD is described at a frame and group of frames (GOF) level (i.e., 10 and 25 frames per GOF respectively). For video we used MPEG-4 Visual Elementary Streams (VES) encoded at the Advanced Simple Profile which includes B-frames. The VES gBSDs are provided at the same granularity as the BSAC gBSDs. Finally, images are encoded using the JPEG 2000 wavelet-based compression algorithm. The JPEG 2000 images are completely described by the gBSD, i.e., without providing the gBSD data on a fragment basis. The test data is taken from the current MPEG-21 reference software[29] and the gBSD describing the *akyio* MPEG-4 visual elementary stream is excerpted in Document 2.

```
<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">
  <dia:DescriptionMetadata>
    <dia:ClassificationSchemeAlias alias="M4V" href="urn:mpeg:mpeg4:visual:cs:syntacticalLabels"/>
  </dia:DescriptionMetadata>
  <dia:Description xsi:type="gBSDType" addressUnit="byte" addressMode="Absolute"
       bs1:bitstreamURI="content/akiyo.cmp">
    <gBSDUnit syntacticalLabel=":M4V:VO" start="0" length="4"/>
    <gBSDUnit syntacticalLabel=":M4V:VOL" start="4" length="14"/>
    <gBSDUnit start="18" length="5830" marker="ICRAParentalRatingViolenceCS-6">
      <gBSDUnit syntacticalLabel=":M4V:I_VOP" start="18" length="4641"/>
      <gBSDUnit syntacticalLabel=":M4V:P_VOP" start="4659" length="98"/>
      <gBSDUnit syntacticalLabel=":M4V:B_VOP" start="4757" length="16"/>
      <gBSDUnit syntacticalLabel=":M4V:B_VOP" start="4773" length="23"/>
      <!--... and so on ...-->
    </gBSDUnit>
    <!--... and so on ...-->
  </dia:Description>
</dia:DIA>
```

Document 2 — Excerpt of a gBSD describing an MPEG-4 visual elementary stream.

The results of the experiments are depicted in Figure 7 to Figure 10. Figure 7 shows the compression efficiency of different string encoding strategies compared to the original plain text XML file. Using the string–token codec introduced in Subsection 4.1 is almost as efficient for encoding of video and image descriptions as the BiM version 2 (i.e., with Zlib codec enabled). In case of audio descriptions an increase in terms of compression efficiency can be noticed. The reason for this rather huge increase is a difference in the size of the string buffers that are Zlib compressed. The whole string table of the string-token codec is Zlib compressed as one chunk, where the BiM Zlib codec uses several chunks. The reason for differences in using 8 bit fixed length tokens or the string-token codec with variable length is the number of strings in the mapping table of the string-token codec. Regularly recurring string values can be heavily found in the MPEG-4 VES gBSDs *akiyo_gbsd.xml* and *foreman_gbsd.xml.* This leads to string tables with very few entries. Both above mentioned gBSDs contain not more than 16 different entries, i.e., each token can be coded in one 5 bit chunk. In some cases the compression ratio of the string-token codec is slightly fewer than BiM version 2 with Zlib enabled (e.g., for akiyo1.xml) which can be explained due to the small number of entries in the string table which is Zlib-encoded by default. However, the compression ratio increases if the string table is provided uncompressed. The decision whether the string table should be Zlib encoded or not could be part of the encoding process based on the number or size of the entries in the string table. The encoding format of the string table could be signalled through a flag at the beginning of the string table.

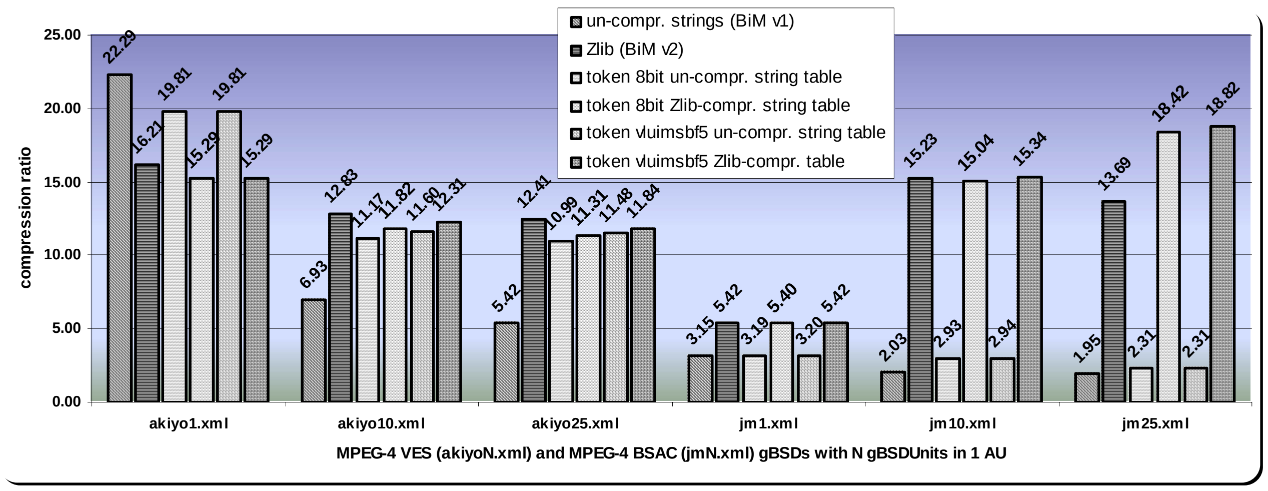

Figure 7 — Compression ratio using different string encoding strategies.

An enhanced processing performance has been expected by coding of skipping information. Figure 8 shows the increase of binary size caused by coding all element lengths which is quite high. For MPEG-4 VES descriptions (akiyo and foreman) about 15% of the total size is needed for length encoding, for JPEG 2000 and BSAC gBSD files the overhead for length encoding is about 25%. In many cases gBSDs are normally not very deeply nested. Therefore, an enhancement in terms of processing speed could not be noticed by using skipping information in the transformation process. Nevertheless, BiM facilitates dynamically (de-)activating of this length encoding mode on a fragment update unit basis which will be investigated as part of our future work.

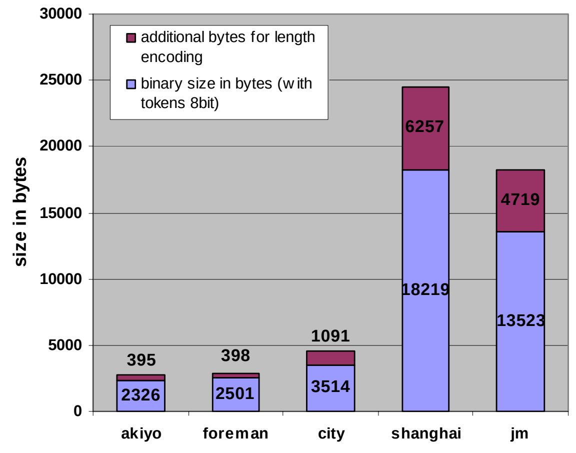

Figure 8 — Increase of binary size by adding skipping information.

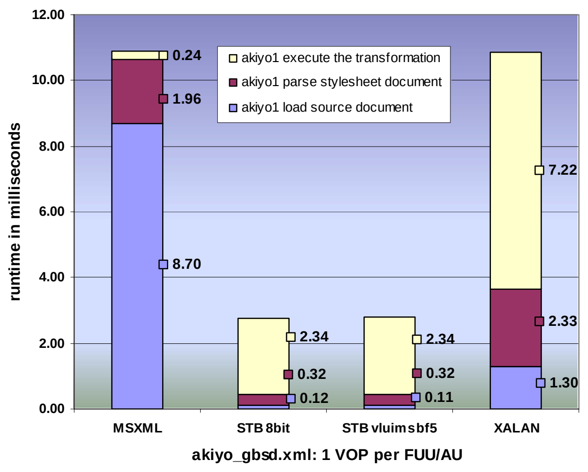

Figure 9 — Transformation runtime comparison per VOP.

For runtime comparisons the use case of B-Frame dropping in MPEG-4 videos has been selected. Therefore the transformer has to delete all `gBSDUnit` elements (see Document 2) that contain an attribute `syntacticalLabel` with the value `:M4V:B_VOP`. The removal itself is done by calling the `stb:remove` template as described in Section 6.

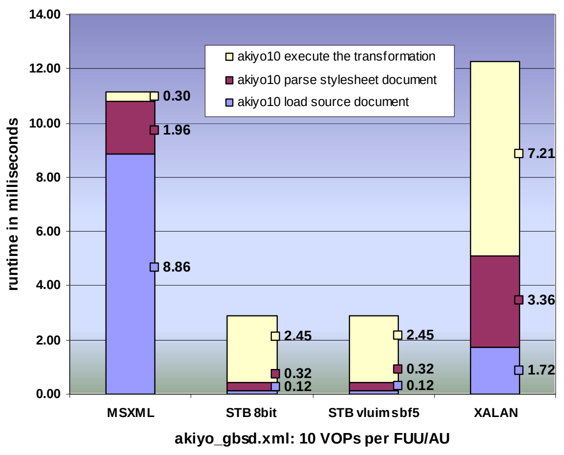

Figure 10 — Transformation runtime comparison per GOV with 10 VOPs per GOV.

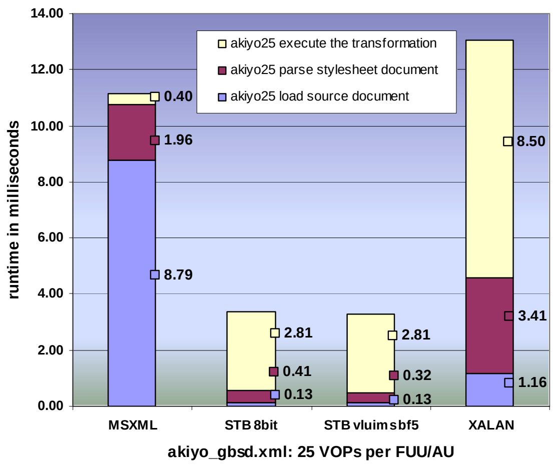

Figure 11 — Transformation runtime comparison per GOV with 25 VOPs per GOV.

In terms of the total runtime the winner is clearly STB as illustrated in Figure 9, Figure 10, and Figure 11. The differences in the loading time of the source document of MSXML, XALAN and the two STB versions are due to the architecture of the transform applications. MSXML and XALAN are DOM-based and need more time to build up the source trees whereas our STB is event-based and only has to read the source file in this phase without any further processing. However, STB needs more time to execute the transformation because creating the events is assigned to this phase of the total transformation process.

For the second MPEG-4 gBSD (*foreman*) similar results could be achieved. The only difference is a larger resulting BiM file that needs more processing time to be serialized.

8. CONCLUSION AND FUTURE WORK

In this paper we have presented a comprehensive approach for transforming MPEG-21 metadata – notably gBSDs which might become very large depending on the granularity of the description – in the binary domain which eliminates the drawbacks from previous work[7] while preserving efficiency in terms of compression ratio and transformation time. Our work is based on an event-based push parser for BiM encoded metadata and processing instructions operating on binary values instead of cost-intensive string comparisons.

In future work, we will further develop and evaluate this approach with regard to the combined processing of gBSD transformation and bitstream generation using gBSDtoBin as indicated in Section 6.

ACKNOWLEDGMENTS

Part of this work is supported by the European Commission in the context of the DANAE (Dynamic and distributed Adaptation of scalable multimedia coNtent in a context Aware Environment) project (IST-1-507113). Further information is available at http://danae.rd.francetelecom.com.

REFERENCES

1. R. Mohan, J. R. Smith, and C.-S. Li, "Adapting Multimedia Internet Content for Universal Access", *IEEE Transactions on Multimedia*, vol. 1, no. 1, pp. 104-114, Jan.-Mar. 1999.
2. A. Vetro, C. Christopoulos, and T. Ebrahami, eds., *IEEE Signal Processing Magazine, special issue on Universal Multimedia Access*, vol. 20, no. 2, March 2003.
3. F. Pereira, J. R. Smith, and A. Vetro, eds., *IEEE Transactions on Multimedia, special section on MPEG-21*, vol. 7, no. 3, June 2005.

4. ISO/IEC 21000-7:2004, *Information Technology – Multimedia Framework – Part 7: Digital Item Adaptation*, October 2004.
5. C. Timmerer, G. Panis, and E. Delfosse, "Piece-wise Multimedia Content Adaptation in Streaming and Constrained Environments (invited paper)", *6th Int'l Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2005)*, Montreux, Switzerland, April 2005.
6. J.-R. Ohm, "Advances in Scalable Video Coding", *Proceedings of the IEEE*, vol. 93, no. 1, pp. 42-56, January 2005.
7. C. Timmerer, P. Lederer, and H. Kosch, "Towards Transforming MPEG-21 Metadata within the Binary Domain", *4th Int'l Workshop on Content-Based Multimedia Indexing (CBMI 2005)*, Riga, Latvia, June 2005.
8. R. J. Bayardo, D. Gruhl, V. Josifovski, J. Myllymaki, "An Evaluation of Binary XML Encoding Optimizations for Fast Stream Based XML Processing", *13th Int'l Word Wide Web Conf.*, New York, USA, pp. 345-354, May, 2004.
9. World Wide Web Consortium (W3C), XSL Transformations (XSLT) Version 1.0, W3C Recommendation, Nov. 1999. http://www.w3.org/TR/xslt.
10. Streaming Transformations for XML (STX) Version 1.0, Working Draft, July 2004. http://stx.sourceforge.net/.
11. A. Berlea, H. Seidl, "Fxt - A Transformation Tool for XML Documents", *Int'l XML Conf. & Exposition*, Dec. 2001.
12. H. Hosoya and B. C. Pierce, "XDuce: A typed XML processing language", *ACM Transactions on Internet Technology*, vol. 3, no. 2, pp. 117-148, May 2003.
13. M. Wallace and C. Runciman, "Haskell and XML: Generic Combinators or Type-Based Translation?", *Int'l Conf. on Functional Programming*, pp. 148-259, September 1999.
14. C. Cooper, Using expat, xml.com, 1999. http://www.xml.com/pub/a/1999/09/expat/index.html.
15. ITU-T and ISO/IEC, "Encoding Using XML or Basic ASN.1 Value Notation", *ITU-T Rec. X.693 (2001) | ISO/IEC 8825-4:2001*, 2001.
16. ITU-T and ISO/IEC, "Mapping W3C XML Schema Definitions into ASN.1", *ITU-T Rec. X.694 (2004) | ISO/IEC 8825-5:2004*, 2004.
17. ITU-T and ISO/IEC, "Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)", *ITU-T Rec. X.691 (2002) | ISO/IEC 8825-2:2002*, 2002.
18. ITU-T and ISO/IEC, "Information technology – ASN.1 encoding rules: Specification of Encoding Control Notation (ECN)", *ITU-T Rec. X.692 | ISO/IEC 8825-3*, 2002.
19. P. Sandoz, A. Triglia, and S. Pericas-Geertsen, "Fast Infoset", *Sun Developer Network Technical Article*, June 2004, available at http://java.sun.com/developer/technicalArticles/xml/fastinfoset/.
20. P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E. Pelegri-Llopart, "Fast Web Services", *Sun Developer Network Technical Article*, August 2003, available at http://java.sun.com/developer/technicalArticles/WebServices/fastWS/.
21. O. Avaro and P. Salembier, "MPEG-7 Systems: Overview", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 6, pp. 760-764, June 2001.
22. ISO/IEC 15938-1:2002, *Information technology — Multimedia content description interface — Part 1: Systems*, 2002.
23. ISO/IEC 15938-1:2002/Amd 1:2005, *Information technology — Multimedia content description interface — Part 1: Systems, AMENDMENT 1: Systems extensions*, 2005.
24. C. Seyrat, ed., "ISO/IEC 21000-16 Information Technology – Multimedia Framework – Part 16: Binary Format", *Final Draft International Standard (FDIS), ISO/IEC JTC 1/SC 29/WG 11 N7247*, Busan, Korea, April 2005.
25. P. Deutsch, J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", *RFC1950*, May 1996, available at http://www.ietf.org/rfc/rfc1950.txt.
26. The official website for SAX, http://www.saxproject.org.
27. P. Nunes, F. Pereira, "MPEG-4 compliant video encoding: analysis and rate control strategies", *34th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, CA, USA, pp. 54-58, November 2000.
28. D. Jannach, K. Leopold, H. Hellwagner, and C. Timmerer, "A Knowledge Based Approach for Multi-step Media Adaptation", *5th Int'l Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, Instituto Superior Técnico, Lisboa, Portugal, April 21-23, 2004.
29. F. De Keukelaere, G. Drury, C. Timmerer, and X. Wang, eds., "International Standard, Information technology — Multimedia framework (MPEG-21) — Part 8: Reference Software", *ISO/IEC 21000-8, Final Draft International Standard (FDIS), ISO/IEC JTC 1/SC 29/WG 11 N7206*, Busan, Korea, April 2005.