# COMPARISON OF XML SERIALIZATIONS: COST BENEFITS VERSUS COMPLEXITY

*Robbie De Sutter*[†] · *Sam Lerouge*[†] · *Peter De Neve*[†] · *Christian Timmerer*[‡] · *Hermann Hellwagner*[‡] · *Rik Van de Walle*[†]

[†] Dept. of Electronics and Information Systems, Multimedia Lab, Ghent University — IBBT
{robbie.desutter, sam.lerouge, peter.deneve, rik.vandewalle}@ugent.be

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


Department of Information Technology (ITEC)
Klagenfurt University
Technical Report No. TR/ITEC/06/1.05
July 2006

Multimedia Systems
DOI 10.1007/s00530-006-0044-y

REGULAR PAPER

# Comparison of XML serializations: cost benefits versus complexity

**Robbie De Sutter · Sam Lerouge · Peter De Neve · Christian Timmerer · Hermann Hellwagner · Rik Van de Walle**



**Abstract** More and more data are structured, stored, and sent over a network using the Extensible Markup Language (XML) language. There are, however, concerns about the verbosity of XML in such a way that it may restrain further adoption of the language, especially when exchanging XML-based data over heterogeneous networks, and when it is used within constrained (mobile) devices. Therefore, alternative (binary) serialization formats of the XML data become relevant in order to reduce this overhead. However, using binary-encoded XML should not introduce interoperability issues with existing applications nor add additional complexity to new applications. On top of that, it should have a clear cost reduction over the current plain-text serialization format. A first technology is developed within the ISO/IEC Moving Picture Experts Group, namely the Binary MPEG Format for XML. It provides good compression efficiency, ability to (partially) update existing XML trees, and facilitates random access into, and manipulation of, the binary-encoded bit stream. Another technique is based on the Abstract Syntax Notation One specification with the Packed Encoding Rules created by the ITU-T. This paper evaluates both techniques as alternative XML serialization formats and introduces a solution for the interoperability concerns. This solution and the alternative serialization formats are validated against two real-life use cases in terms of processing speed and cost reduction. The efficiency of the alternative serialization formats are compared to a classic plain text compression technique, in particular ZIP compression.

**Keywords** XML serialization formats · MPEG-B · ASN.1 · Information encoding · Data interchange formats · Multimedia applications

R. De Sutter (✉) · S. Lerouge · P. De Neve · R. Van de Walle
Department of Electronics and Information Systems — Multimedia Lab,
Ghent University — IBBT,
Sint-Pietersnieuwstraat 41,
9000 Ghent, Belgium
e-mail: robbie.desutter@ugent.be

S. Lerouge
e-mail: sam.lerouge@ugent.be

P. De Neve
e-mail: peter.deneve@ugent.be

R. Van de Walle
e-mail: rik.vandewalle@ugent.be

C. Timmerer · H. Hellwagner
Department of Information Technology (ITEC),
Klagenfurt University,
Universitätsstraße 65–67,
9020 Klagenfurt, Austria
e-mail: christian.timmerer@itec.uni-klu.ac.at

H. Hellwagner
e-mail: hermann.hellwagner@itec.uni-klu.ac.at

## 1 Introduction

The *Extensible Markup Language* (XML) is gaining momentum. More and more XML-based data are structured, stored, and sent over networks. Due to this, the disadvantages of the language should no longer be ignored because they may restrict the adoption of XML in future applications. XML represents its data in plain-text encoding (e.g., UTF-8), thus guaranteeing platform-independent processing thereof, as long as the platform can handle the used content encoding

format. However, this kind of serialization introduces a lot of overhead, also known as the *verbosity of XML*. The overhead is a potential disadvantage, especially when using XML in constrained environments, such as mobile devices, where memory, processing power, and network bandwidth are restricted. One could argue that these devices are becoming more and more powerful and such limitations will become obsolete. In practice, however, these network-enabled devices are becoming smaller and smaller and usage of XML-based data is increasing, therefore similar constraints will apply to future devices as much as they apply to the devices we are using today. Furthermore, if we want to exploit the full potential of any device when handling XML, we should address the overhead.

The verboseness issue can be solved by an alternative serialization of the XML data, i.e., through binary encoding. The ISO/IEC JTC 1/SC 29/WG 11 group—better known as the *Moving Picture Experts Group* (MPEG)—standardized such a binary serialization as part of the MPEG-7 Systems standard, namely *Binary format for Metadata* (BiM) in 2001 [8,19]. Initially, BiM was intended as an alternative and compact serialization of XML-based MPEG-7 descriptions. According to [19] the BiM approach can also be applied to XML files in general as long as there is an available *Description Definition Language* (DDL) [9] Schema or an XML Schema. The DDL adopted the XML Schema specification with some multimedia-related extensions. Hence, XML Schema is a subset of DDL. Recently, the BiM technology has been relocated to a new MPEG standard formally known as MPEG-B Part 1 — *Binary MPEG format for XML* [11].

Besides the binary XML efforts within the MPEG group, ISO/IEC has put some joint efforts with ITU-T towards an alternative XML serialization with the *Abstract Syntax Notation One* (ASN.1) technology [13]. Originally, it was intended to describe the structure and type of any kind of structured data for representing, encoding, and decoding of this data in a generic fashion. Mapping rules between XML Schemas and ASN.1 Schemas are defined [12,15], and, for ASN.1 instances, efficient binary encoding schemes such as *Packed Encoding Rules* (PER) are available [14]. Recently, tools became available to convert XML Schemas to ASN.1 Schemas. These ASN.1 Schemas can be used to automatically generate source code that can be compiled into an application that can apply a binary encoding scheme, like PER.

It is true that the verboseness issue of XML could be addressed by simply applying well-known plain-text compression algorithms such as the Lempel/Ziv algorithm [28,27] as used in WinZIP or GnuZIP. However, in particular BiM provides additional benefits besides achieving compression ratios comparable to those plain-text compression schemes [23]. The most important benefits are the fact that parsing and manipulating the binary-encoded XML data can occur directly in the binary domain, streaming of the data is supported, and optimized techniques to update and modify XML trees are standardized. Note that XML documents do not natively support streaming, i.e., a transmitted XML document is only well-formed if the closing tag of the root element is also received.

For alternative XML serialization formats to be well adopted, regardless of the exact type of serialization, it should (1) avoid interoperability issues with existing applications, (2) not add any additional degree of complexity for the application developers, and (3) offer a demonstrable cost reduction. As for (1), interoperability is ensured by using open standards—MPEG-B Part 1 is an ISO standard [11] and ASN.1 is an ITU.T specification [13]. To avoid the complexity concerns as mentioned in (2), it is important to realize that application developers handle (plain-text) XML data using an XML parser. As such, it is desirable that developers continue to use their preferred type of parser if they want to handle binary-encoded XML data. Moreover, they need not be aware of the fact they are handling binary-encoded or plain-text XML. It is the parser's responsibility to handle the XML data correctly. The parser is an intermediate layer between the application and the XML source data. As for (3), the cost reduction is achieved by the fact that fewer bytes need to be transferred over networks (reduced bandwidth). Most network operators charge their users (in the broadest sense of the word, thus end users, content providers, and so on) per time unit or per byte transferred. By reducing the byte volume, a measurable saving can be accomplished. Also the network operators benefit from the reduced byte volume as they are capable of handling more users without additional expensive infrastructure.

The remainder of this paper is organized as follows. Section 2 covers the various existing efforts being made to address the verboseness issue of XML. In Sect. 3, we elaborate on the technical details of MPEG-B and of ASN.1. In Sect. 4, we introduce a novel kind of XML parser, namely a parser that is serialization-type agnostic. Section 5 discusses two real-life use cases to evaluate the usefulness and the economical advantages of alternative binary-encoded XML data. The materials and methods for the evaluation are discussed in Sect. 6 and the results are discussed in Sect. 7. Finally, Sect. 8 concludes this paper.

## 2 Related work

Apart from MPEG, other standardization bodies are investigating the need and usefulness of an alternative XML serialization. In this section, we briefly give an overview of these efforts.

The driving force behind XML, the *World Wide Web Consortium* (W3C), founded a task force in 2004 to investigate the usefulness and desirability of an alternative serialization format. This resulted in a first working draft of relevant use cases and applications that could benefit from a (binary) serialization. This document is regularly updated and can be found on the Website of the W3C [3]. The report is used to determine if it is possible to select one specific kind of serialization for all use cases. Amongst others, MPEG-B is listed as a candidate solution. Furthermore, this task force has recently released a new working draft describing measurement aspects, methods, caveats, test data, and test scenarios for evaluating the potential benefits of an alternative serialization for XML [26]. Other related W3C activities include the efficient transportation of non-XML-based data within XML-based data only, e.g., *SOAP messages with attachments* [1], *XML-binary Optimized Packing* [7], and *Resource Representation SOAP Header Block* [17].

Also, the Web service community has recognized the verboseness issue and is currently developing alternative XML serialization schemes known as *Fast Infoset* [22] and *Fast Web Services* [21]. The latter is built upon ASN.1 as described in the previous section. The former uses 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 is again a burden. Performance results comparing these two approaches with other binary XML encoding schemes are not available at the time of writing.

Finally, we want to mention two proprietary solutions, namely XMill [18] and XMLPPM [2]. The former exploits the self-describing nature of XML for compression by leveraging existing compression algorithms and tools like ZLIB (the library function version of GZIP) and some simple data type specific compressors. The latter is a compression tool for XML documents that combines the well-known Prediction by Partial Match (PPM) and the Multiplexed Hierarchical Modeling algorithms. Currently these alternative XML serialization solutions are not yet stable for real-life applications, but progress is being made. In [23] a comparison of MPEG-B BiM to XMill and XMLPPM is made in terms of compression efficiency. The results demonstrate that BiM is superior to these solutions.

## 3 Technical overview

### 3.1 Binary MPEG format for XML

The Binary MPEG format for XML (BiM) was initially designed to binary encode only MPEG-7 descriptions, which are XML based. However, BiM can also cope with other kinds of XML-based data as long as they are based on a DDL or an XML Schema and the respective DDL or XML Schema definition is available.

BiM is an XML Schema aware encoding scheme for XML documents [19], i.e., it uses information from the XML Schema to create an efficient alternative serialization of XML documents within the binary domain. This knowledge enables the removal of structural redundancy (e.g., white space, element and attribute names), thus achieving high compression ratios with respect to the document structure. Furthermore, element and attribute names as well as data are encoded using dedicated coders based on the data type (integer, float, string) which further increases the compression ratio. The resulting compression ratio is comparable to those achieved by traditional plain-text compression algorithms [19]. The advantages of BiM over traditional plain-text compression algorithms are the support of parsing the XML data in the binary domain (thus without mapping to plain-text XML), its streaming capabilities, and dynamic and partial updating of existing XML trees.

To achieve the latter, BiM can divide an XML tree into different parts. One part is encoded into an *access unit* and contains an optional *schema update unit* and one or more *fragment update units*. As a decoder needs to know the set of utilized XML Schemas in order to decode the XML data, the schema update unit makes it possible to modify the initially retrieved set of schemas. More interesting are the fragment update units, which in turn consist of three parts: a *fragment update command*, a *fragment update context*, and a *fragment update payload*. The fragment update command specifies the decoder action for the corresponding fragment which can be either add, delete, replace, or reset, i.e., BiM provides partial updates of an XML document. The fragment update context is used to uniquely determine the location of the fragment to be updated in the XML document by, for example, an XPath expression. Finally, the fragment update payload contains the actual encoded XML data of the update. Figure. 1 illustrates how an XML document is divided into access units and streamed over

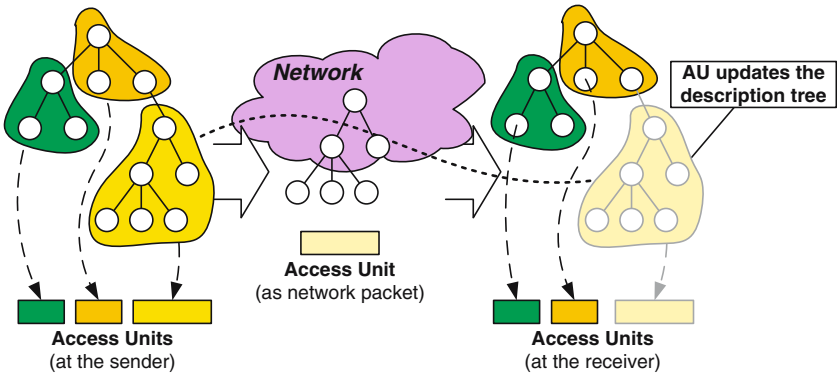


**Fig. 1** Streaming XML documents over the network using access units (AU)

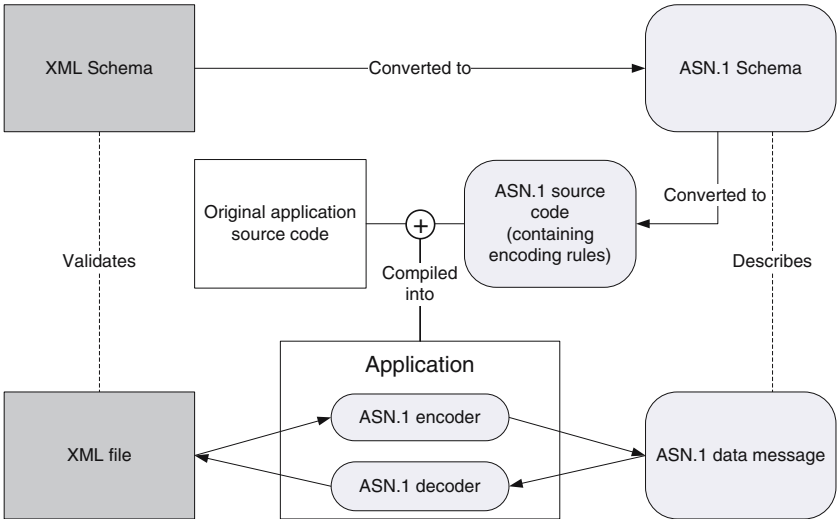


**Fig. 2** Handling XML documents by an ASN.1 application

the network. In particular, it shows how a sub-tree of the whole XML document is transmitted over the network and added to the XML tree at the receiver side (cf. dotted line).

### 3.2 Abstract Syntax Notation One

The *International Telecommunication Union* (ITU), and more specifically the *ITU Telecommunication Standardization Sector* (ITU.T), together with the ISO/IEC started development of the *Abstract Syntax Notation One* (ASN.1) standard in 1984 [13]. Originally, it was intended to describe the structure and type of any kind of organized data for representing, encoding, and decoding of the data in a generic fashion.

ASN.1 stores information in an *ASN.1 data message*, which is structured according to an *ASN.1 Schema* and serialized by an *ASN.1 encoding rule*. The schema takes care of the abstract notation of the data by dividing data messages into smaller and simpler basic structures such as integers, booleans, and strings and by combining these basic structures into bigger structured types. The main task of an encoding rule is to remove any redundant structural information from the data message and to make sure that the receiver of the message uses the correct character set or decoder. If a program is aware of the ASN.1 Schema and the selected ASN.1 encoding rule, it can process (read and write) correctly formed data messages. In [15] mapping rules between an XML Schema and an ASN.1 Schema are defined. As such, ASN.1 Schema can be seen as an alternative to XML Schema.

Based on the ASN.1 Schema, different tools are available to automatically generate source code that can be compiled into other applications. These tools also generate source code for the different encoding rules so for each encoding rule two methods are available: an encoding method and a decoding method. The former accepts a valid XML file and returns the ASN.1 data message; the latter accepts an ASN.1 data message and returns a valid XML file. As such, it is trivial to process and generate ASN.1 data messages compliant for a given ASN.1 Schema from any application. A schematic overview of the ASN.1 procedure to handle XML data can be found in Fig. 2.

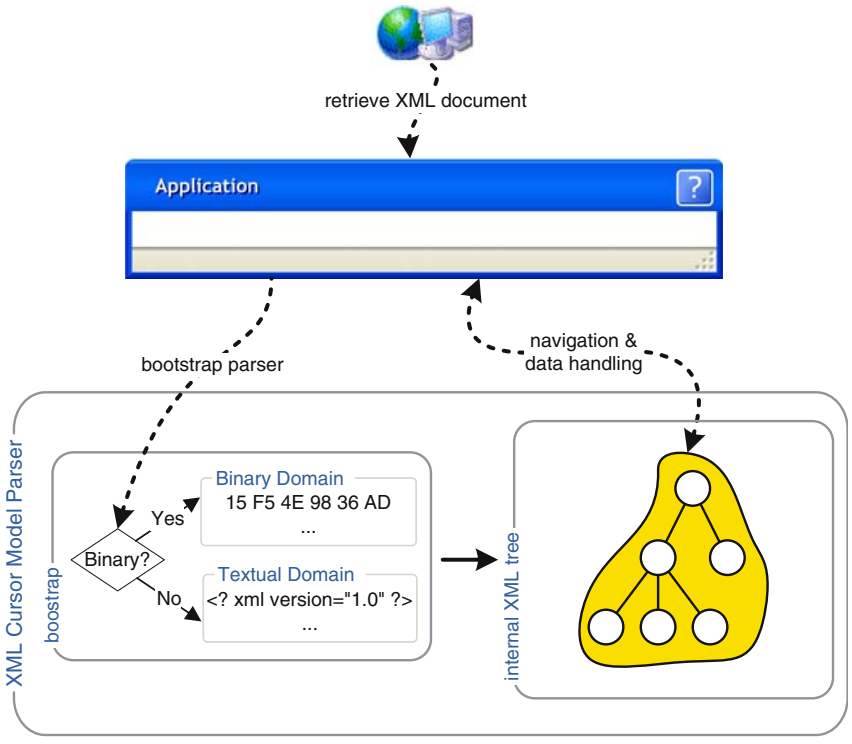


**Fig. 3** Architectural overview of the XML Cursor Model Parser capable of handling plain-text XML and binary-encoded XML

## 4 Serialization-type agnostic XML parser

### 4.1 XML parser models

In almost all use cases, applications process an XML data stream indirectly using an XML parser. Application developers can choose from a wide range of different parsers, where each parser has its own possibilities, characteristics, and fields of applications. When observing the currently available parsers, we can categorize them into five distinct classes [6]:

1. The *Tree Model* (e.g., W3C DOM parser[1]). This is the first parser model. It exploits the tree structure characteristics of an XML document.
2. The *Push Model* (e.g., SAX[2]). The parser generates an event for each XML token (e.g., a start tag, an attribute, etc.).
3. The *Pull Model* (e.g., XMLPull[3]). The parser is instructed by an application when to read the next XML token.
4. The *Cursor Model* (e.g., .NET XPathNavigator[4]). The model is based on the database views concept, i.e., a view on the XML data is created through an XPath expression.
5. The *Mapping Model* (e.g., .NET XmlSerializer[5]). The parser maps the XML content onto an object-oriented data structure.

### 4.2 Serialization-type agnostic XML parser architecture

Based on the conclusions in [6], we developed an XML Cursor Model Parser capable of handling both plain-text as well as binary-encoded XML data (Fig. 3). As such, an application using the parser is unaware of the serialization format of the XML data. Hence, application developers create applications that support BiM-encoded or ASN.1-PER-encoded XML data transparently.

In practice, an application retrieves an XML document from a source and sends it to the bootstrap method of the parser. The bootstrap method determines the serialization format of the XML data either trivially, by looking at the extension of the filename, or more advanced by using the MIME-type information or by inspecting the first bytes of the stream. Once the bootstrap method has identified the serialization type, it can handle the data appropriately.

The parsing of BiM-encoded data occurs directly by the serialization-type agnostic parser. Indeed, the

[1] W3C Recommendation, Document Object Level 3 Core Specification, available at http://www.w3.org/TR/DOM-Level-3-Core/

[2] Simple API for XML, available at http://sax.sourceforge.net.

[3] Common API for XML, available at http://www.xmlpull.org.

[4] Microsoft .NET XPathNavigator, available at http://msdn.microsoft.com.

[5] Microsoft .NET XmlSerializer, available at http://msdn.microsoft.com.

internal XML tree is constructed simultaneously with the processing of the BiM -serialized data. This is accomplished by integrating the BiM reference software (January 2005 version, [10]) with the parser. As a result, we do not need to decode the data first before the creation of the internal XML tree can start.

In the case of ASN.1-PER-encoded data, the specific ASN.1-PER decoding software is first used, which results in plain-text XML data. After the decoding step, the parser processes the plain-text XML data.

If the content-encoding format of the XML data is plain text, it is only necessary to create the internal XML tree for which existing XML parsers, as described in the previous section, can be used.

## 5 Use cases

In the remainder of this paper, we want to evaluate the usefulness of an alternative XML serialization in real-life applications. We investigate the complexity concerns, execution time issues, and cost-effectiveness. This section describes two real-life applications, their current problems, and how alternative serializations can address these issues.

### 5.1 Use case 1: usage environment description notification

*Universal Multimedia Access* (UMA) [20,24] refers to a paradigm where audiovisual data are produced once and consumed on any kind of end-user device, anywhere and anytime. UMA dynamically adapts a multimedia resource so that the modified resource meets the requirements of the target application or device. In order to achieve an optimal content adaptation, the UMA-enabled adaptation engine requires a description of the usage environment. However, this usage environment is not static and changes thereof can occur during consumption of the audiovisual data, for example the available bandwidth can change. When a modification occurs, the adaptation engine has to be informed such that the adaptation scheme is updated accordingly [5].

Due to its update capability, BiM is an ideal candidate for dealing with the volatile nature of the usage environment. BiM is capable of encoding only the information that has changed and that needs to be transmitted over the network. This eliminates the need to send the full usage environment information whenever a change thereof occurs. For the network operator this has an economical advantage. The required bandwidth reduction implies that more users can use the same existing network infrastructure.

Another issue arises when applying the UMA paradigm to constrained devices, e.g., a PDA or a cellular phone. These devices are not only limited in memory capacity and processing capabilities, but they have a constrained network connection, for example a *General Packet Radio Service* (GPRS) connection, which has an upload transfer speed that is several factors slower than download speed [16] and for which the end user has to pay a fee based on the number of bytes transferred.

Again, BiM is an ideal candidate to address this issue. BiM reduces the number of bytes to be sent in two ways: by binary encoding the information and by only sending the updated information. As a result, the BiM approach reduces the fee the end user pays, hence a cost reduction.

### 5.2 Use case 2: RSS feeds

*Really Simple Syndication* or *Rich Site Summary* (RSS) is an XML application that enables users to be informed whenever an update occurs to an Internet news source. In this paper, we refer to version 2.0 of the RSS specification[6]. This specification defines a container structure: an RSS document—the *RSS feed*—contains one or more *channels*, e.g., a weather information channel. A Channel groups multiple *Items*. One Item is a particular news piece on the topic of the given Channel, e.g., weather information for a particular city.

Recently, RSS became a popular tool as a means to disperse *podcasts*. Indeed, a podcast publishes multimedia data (such as audio and video) by using RSS feeds, such that the feed subscribers automatically receive new content. This is accomplished by adding the optional enclosure element to an RSS Item, which contains an URL to the actual location of the audio-visual data stream. As such, it is comparable to the technique to add multimedia content to a Web page.

The *RSS viewer*, i.e., the client application, retrieves the RSS feed from a server on a regular basis, e.g., by downloading the RSS document every day. Note that in this use case the XML data are transmitted from the server to the client whereas in the first use case it is the other way round, i.e., the usage environment description is sent from the client towards the server. When one or more new Items are available, the RSS viewer will inform the user about these by, for example, showing the titles of the new Items on his or her display. The RSS publisher updates the RSS feed by adding new Items or removing obsolete Items at the server. Usually the Channel information is not modified.

[6] The RSS 2.0 Specification is discussed at http://blogs.law.harvard.edu/tech/rss.

Current issues with the RSS application are obvious. The RSS viewer regularly downloads the RSS data file anew in order to check for added Items. If the RSS data file was not changed since the last retrieval, this download is unnecessary and the bandwidth waste is obvious. But even if new Items were added to the RSS file, bandwidth is wasted, as the new Items are usually only a relatively small part of the complete XML file. Again, for an end-user, this overhead is expensive, especially when using an Internet connection which has to be paid on a per byte basis. For content providers, the overhead can also become a big concern when their RSS feed is popular. The overhead when millions of users are subscribed to a certain feed may become significant. Note that content providers also pay a fee for the used bandwidth. In other words, the overhead introduces a cost for the end-user as well as for the content provider.

## 6 Materials and methods

For the evaluation of the first use case, we have selected the MPEG-21 *Digital Item Adaptation* (DIA) standard amongst various standards to describe the usage environment [4,25]. The main part of the DIA standard comprises the *Usage Environment Description* (UED) format which provides means to describe and structure the usage environment containing information about the (end) user, the terminal, the network, and the natural environment.

We created a conceivable usage environment description compliant to MPEG-21 DIA UED. This usage environment description is sent to an adaptation engine located on a server or on a proxy device within the network. Upon receiving the description, the adaptation engine bootstraps the XML Cursor Model Parser. Thereafter, three kinds of updates are transmitted from the device to the server. First a small-sized update is sent which modifies information about the bandwidth of the network. Next, a medium-sized update is transmitted changing the terminal's display information. And finally, a large update is sent which changes the user and the natural environment information.

For the second use case, we emulated a typical usage of RSS. An RSS viewer retrieves the RSS data from a content provider daily during the period of 1 month. In practice, we used the MSDN[7] RSS feed of the month November 2004 containing a total of 53 Items scattered over the weekdays of the month.

In order to evaluate the usefulness of MPEG-B to solve the issues of the two use cases, we compare four different kinds of XML serialization types:

1. *Plain text*. The classical way of serializing XML data is by storing the data as UTF-8 or UTF-16 encoded plain text. In these tests, we use UTF-8 encoding.
2. *BiM encoding*. The MPEG-B Binary MPEG format for XML, as described in Sect. 3.1, is applied to the XML data. Handling (parsing) the BiM-encoded XML data occurs in the binary domain, thus without decoding the BiM data to plain text.
3. *ZIP compression*. For this serialization type, we apply ZIP compression to the plain-text XML data files. Before parsing the ZIP-compressed XML data, the compressed data are decompressed to plain text. This serialization type requires that the ZIP-compressed data are completely available at the receiver before decompressing and parsing can start. For the tests, we make use of the internal ZIP handling package of Java 1.5 which is based on the ZLIB specification[8] and PKWARE's Application Note on the .ZIP file format.[9]
4. *ASN.1-PER*. This serialization type applies the Packed Encoding Rules for the Abstract Syntax Notation One to the plain-text XML data, as described in Sect. 3.2. In order to apply this serialization type, an RSS and a DIA-UED PER-encoder/decoder were created. To do this, first the ASN.1 Schemas were automatically generated[10] for the XML Schemas. Next, Java source code, which includes the PER capabilities, was generated from these ASN.1 Schemas using the OSS Nokalva ASN.1 tools[11] and was combined with a newly developed application, resulting in a new software module. These software modules accept an XML file (valid to the particular RSS or DIA-UED XML Schemas) as input and return the ASN.1-PER encoded version and vice versa. Note, there are two independent ASN.1-PER software modules: one module for the Usage Environment Description Notification use case (based on the DIA-UED XML Schemas) and one module for the RSS feeds (based on the RSS XML Schema).

[7] The Microsoft Developer Network (MSDN) Website can be found at http://msdn.microsoft.com/.

[8] The ZLIB specification can be found in the RFC 1950 document.

[9] More information on the ZIP file format can be found at http://www.pkware.com/.

[10] XML Schema to ASN.1 Schema conversion can be done on the ASN.1 dedicated website at http://asn1.elibel.tm.fr/

[11] OSS Nokalva ASN.1 tools can be found at the Website http://www.oss.com/.

**Table 1** Results for use case 1: usage environment description notification — full mode

| Full mode | Plain text | | BiM encoded | | | ZIP compressed | | | ASN.1-PER | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Byte size | Parse (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) |
| Full UED | 8,805 | 2.9 | 939 | 276.1 | 628.2 | 2,175 | 5.7 | 3.8 | 1,808 | 9.4 | 83.1 |
| Small update | 8,810 | 2.8 | 937 | 275.3 | 626.7 | 2,179 | 5.6 | 4.2 | 1,806 | 9.4 | 49.2 |
| Medium update | 8,574 | 2.9 | 933 | 275.2 | 627.2 | 2,162 | 8.6 | 4.3 | 1,785 | 9.4 | 56.9 |
| Large update | 13,847 | 4.2 | 1,046 | 279.2 | 654.3 | 2,790 | 7.4 | 5.0 | 2,450 | 13.1 | 61.2 |

All four serialization types are used in two modes: a *full mode* and an *update mode*. The full mode is the classical way of exchanging XML-based information, namely by transmitting the complete and well-formed XML file valid to the corresponding XML Schemas. In practice, for the first use case the updates are applied to the usage environment description on the client device and the resulting description is used as input for the serialization type, for the second use case, a valid RSS feed containing Items published up to the "current day" of the month. The update mode only transmits the modifications, in other words, the small, medium, and big modifications for the first use case and the Items that were published between two successive days for the second use case. Note that only the BiM serialization type can natively handle updates to previously received XML information through its Access Units (as discussed in Sect. 3.1). For all other serialization types, a proprietary and therefore non-interoperable construction is used to handle the update information. Indeed, the serialization-type agnostic XML parser has hard-coded rules to process these updates. As such, this solution is only for the BiM serialization-type generic, interoperable, and applicable for commercial and enterprise applications. Nevertheless, the update mode for the plain text, ZIP compressed, and ASN.1-PER serialization types is useful for a comparison.

For each use case, each serialization type and each mode, we measured the byte size of the data that must be transmitted, the time required to parse the data, and if applicable, the time required to prepare the XML data before transmission. The latter implies using the BiM encoder for the BiM serialization type, compressing the XML files for the ZIP-compression serialization type, and executing the ASN.1-PER encoding software for the ASN.1-PER type. The time required to parse the data also includes the time required to decompress the ZIP-compressed XML data and the time required to decode the PER-encoded data using the ASN.1-PER decoding software.

We executed ten runs of the tests without exiting the Java Runtime Environment. The numbers show the average thereof whereby at most one outlier per test was discarded. The latter is necessary to minimize the influence of a run of the Java garbage collector and delay introduced by I/O. The standard deviation of the remaining numbers indicates that the resulting average is significant.

All measurements were performed on an Intel Centrino Pentium M 1.1 Ghz processor running Windows XP Pro SP2 and Java 1.5.0. For the runtime measurements we use the *System.nanoTime()* method of Java 1.5.0.

## 7 Results and discussion

### 7.1 Results for use case 1: usage environment description notification

The results for the first use case are shown in Tables 1 and 2. The tables show for each of the four serialization types the byte size of the data, the time required to parse this file, and the time required to prepare the XML data before transmission (also called the *encoding* of the data).

In case of the full mode (Table 1), the results clearly show that the time required to parse BiM-encoded XML data is about 90 times higher than plain-text XML data while at the same time the byte size is about 10 times smaller. ZIP compression doubles the parsing time compared to the plain-text serialization type and achieves a compression ratio of 4:1. ASN.1-PER needs about triple execution time for a compression ratio of 5:1.

In case of the update mode (Table 2), the BiM compression efficiency decreases to 6:1, but the parsing time increases; BiM needs about 250 times longer to parse the data. Also ZIP compression is less efficient, namely 3:1, and requires four times more time to parse. Contrary to the other binary serialization types, ASN.1-PER improves its compression efficiency, namely from 5:1 for the full mode to 10:1 for the update mode, without a time penalty.

The time to encode the XML data for the different kinds of serialization reveals similar results: BiM is by far the slowest serialization type, then ASN.1-PER, and finally ZIP compression.

**Table 2** Results for use case 1: usage environment description notification — update mode

| Update mode | Plain text | | BiM encoded | | | ZIP compressed | | | ASN.1-PER | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Byte size | Parse (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) |
| Full UED | 8,805 | 3.1 | 939 | 277.2 | 631.4 | 2,175 | 6.8 | 4.6 | 1,808 | 10.2 | 83.3 |
| Small update | 710 | 0.6 | 305 | 264.5 | 680.6 | 524 | 3.1 | 2.8 | 45 | 1.9 | 9.4 |
| Medium update | 808 | 0.7 | 697 | 266.7 | 680.3 | 526 | 6.2 | 2.3 | 55 | 2.3 | 7.4 |
| Large update | 10,393 | 3.3 | 1,110 | 386.7 | 1,049.8 | 2,107 | 6.9 | 3.5 | 1,655 | 10.8 | 48.7 |

**Table 3** GPRS price for use case 1 for the different serialization types and modes [GPRS price of a Belgian telecom operator (0.5€/100 Kb)]

| | Byte size | GPRS price (€) | Cost reduction (%) |
|---|---|---|---|
| Full Plain text | 40,036 | 0.19 | |
| Full BiM encoded | 3,855 | 0.01 | 90.37 |
| Full ZIP compressed | 9,306 | 0.04 | 76.75 |
| Full ASN.1-PER | 7,849 | 0.03 | 80.39 |
| Plain-text updates | 20,716 | 0.10 | 48.25 |
| BiM-encoded updates | 3,051 | 0.01 | 92.37 |
| ZIP-compressed updates | 5,332 | 0.02 | 86.68 |
| ASN.1-PER updates | 3,563 | 0.01 | 91.10 |

The slow parsing of the BiM-encoded XML data and the slow BiM encoding of plain-text XML data can be explained by the usage of the MPEG reference software for BiM encoding/decoding which is not optimized in terms of runtime. Commercial implementations of the MPEG-B standard should provide a more optimized solution; unfortunately, such implementations were not available for this evaluation. Also, the XML Schema for MPEG-21 DIA UED is very complex and comprehensive. Analysis of this schema is not straightforward and introduces a time penalty for parsing and encoding. Optimized implementations of the standard will probably provide a caching mechanism for analyzed XML Schemas. Furthermore, as the MPEG-21 DIA UED XML Schema is standardized, it is envisaged that applications or devices will use a hard-coded version thereof avoiding the analysis of the XML Schema phase. The ASN.1-PER serialization type avoids this pitfall as the analysis phase of the XML Schema is performed during creation of the Java source code. ASN.1-PER requires over 200 Java classes to model the MPEG-21 DIA UED XML Schemas, in other words over 200 complex type XML elements are used in the UED XML Schemas. The ASN.1-PER tools require about 400 ms to create these classes.

In Table 3, Figs. 4, and 5, the cumulated byte sizes for the different serialization types are listed, these are the total number of bytes that are sent when the three updates are consecutively applied to the initial usage environment description. Additionally, Table 3 shows the costs when sending this amount of data over a GPRS connection. The cost reduction when using binary-encoded XML data is the highest when all functionalities provided by BiM are exploited, in other words, for BiM-encoded serialization in update mode.

### 7.2 Results for use case 2: RSS feeds

For the second use case the results are listed in Tables 4 and 5. These tables show for each of the four serialization types the byte size of the data, the time required to parse this file, and the time required to encode the XML data before transmission.

In case of the full mode (Table 4), the results show that the time required to parse BiM-encoded XML data is about 20 times slower than plain-text XML data while at the same time the byte size is about 5 times smaller. The compression efficiency is lower than for the first use case because the XML data contains more text values whereas in the UED test case there is more overhead by the XML structure. This fact also results in a lower compression efficiency for the ASN.1-PER serialization type, namely 1.5:1. Only ZIP compression retains its compression ratio of 4:1. The parsing and the encoding of the XML data for the different serialization types are proportional to the byte size. The parsing of the BiM-encoded XML data is faster in comparison to the first use case due to the simple XML Schema for RSS feeds. Nevertheless, this parsing time remains high compared to the other serialization types. The ZIP compression serialization type is, again, the fastest of the three binary serialization types.

For the values in Table 5 (the update mode) and for the larger byte size values in Table 4 (the full mode), the times encoding and parsing the BiM serialization type is higher than in the first use case. This can be

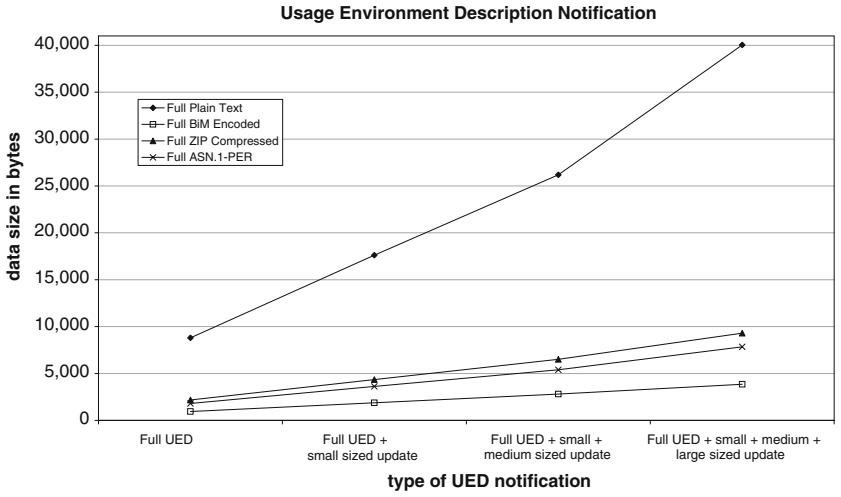


**Fig. 4** Cumulated byte sizes UED notifications—full mode

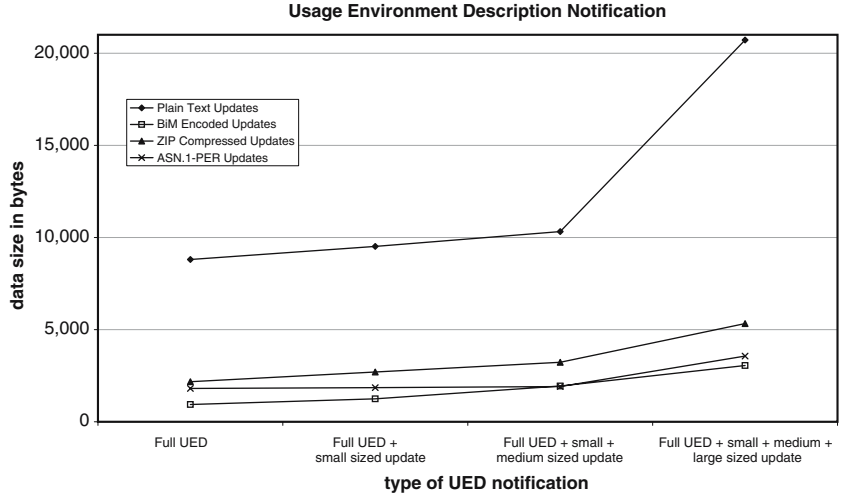


**Fig. 5** Cumulated byte sizes UED notifications—update mode

explained as for every new Item in the RSS feed, a new fragment update unit is created. The parsing and encoding of multiple fragment update units into one access unit delays the processing speed. Once again, we want to emphasize that we make use of the non-optimized reference software and that a more optimized implementation of the BiM specification, which handles multiple fragment update units better, should reduce the executing time for parsing as well as encoding. The results also show that the gap of the compression efficiency for the different serialization types is closing. In fact, ZIP compression outperforms BiM encoding with respect to parsing and encoding, mainly due to performance issues of the BiM reference software, but regarding byte size BiM is superior in all cases with a few exceptions though. In particular, if many new RSS Items need to be transmitted, ZIP provides a slightly better compression ratio than BiM. This indicates that the overhead for creating a new fragment update unit for every new Item is large, not only in terms of execution time, but also in terms of bytes.

The cumulated byte sizes for the different serialization types are listed in Table 6 and depicted in Figs. 6 and 7. We compare the used bandwidth for this use case after month with regards to the different serialization types. We also provide the price when sending this amount of data over a GPRS connection.

Once again, the cost reduction is the highest for the BiM-encoded serialization type in update mode. However, ZIP compression is only slightly worse than BiM encoding for both the full and update mode, but the latter mode implies application domain-specific handling of the data.

## 8 Conclusions

As XML is nowadays used to store more and more data, the verbosity of the format is a disadvantage that

**Table 4** Results for use case 2: daily retrieval of an RSS feed — full mode

| Full mode | Plain text | | BiM encoded | | | ZIP compressed | | | ASN.1-PER | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Day | Byte size | Parse (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) |
| 1 | 611 | 0.6 | 226 | 65.1 | 236.8 | 503 | 4.7 | 26.7 | 280 | 1.9 | 21.2 |
| 2 | 1,217 | 0.8 | 437 | 65.7 | 245.6 | 762 | 3.6 | 27.0 | 687 | 2.4 | 23.4 |
| 3 | 1,748 | 1.0 | 522 | 66.1 | 248.1 | 845 | 4.5 | 27.7 | 1,018 | 3.0 | 26.2 |
| 4 | 3,207 | 1.5 | 895 | 66.9 | 261.8 | 1,224 | 4.7 | 29.6 | 2,015 | 4.1 | 31.2 |
| 5 | 4,013 | 1.8 | 1,049 | 67.5 | 273.0 | 1,372 | 7.1 | 30.6 | 2,564 | 4.8 | 34.5 |
| 6 | 4,013 | 1.8 | 1,049 | 67.9 | 272.4 | 1,372 | 4.7 | 30.7 | 2,564 | 4.8 | 33.4 |
| 7 | 4,013 | 1.8 | 1,049 | 67.4 | 272.3 | 1,372 | 4.8 | 31.0 | 2,564 | 4.8 | 35.0 |
| 8 | 5,372 | 2.2 | 1,403 | 68.2 | 278.6 | 1,724 | 6.2 | 33.1 | 3,495 | 6.0 | 41.5 |
| 9 | 5,372 | 2.2 | 1,403 | 68.1 | 278.9 | 1,724 | 5.3 | 34.7 | 3,495 | 6.0 | 41.2 |
| 10 | 8,783 | 3.4 | 2,120 | 70.1 | 296.4 | 2,403 | 6.7 | 37.1 | 5,905 | 8.8 | 54.6 |
| 11 | 9,703 | 3.7 | 2,402 | 70.6 | 301.3 | 2,691 | 7.4 | 38.0 | 6,599 | 9.6 | 58.8 |
| 12 | 11,417 | 4.3 | 2,698 | 71.7 | 371.6 | 2,973 | 7.7 | 40.9 | 7,741 | 11.0 | 66.8 |
| 13 | 11,417 | 4.3 | 2,698 | 71.6 | 372.2 | 2,973 | 8.4 | 41.7 | 7,741 | 10.9 | 65.4 |
| 14 | 11,417 | 4.3 | 2,698 | 71.6 | 371.8 | 2,973 | 7.4 | 40.8 | 7,741 | 10.9 | 64.8 |
| 15 | 12,398 | 4.6 | 2,856 | 72.9 | 439.4 | 3,121 | 8.1 | 42.0 | 8,377 | 11.8 | 70.4 |
| 16 | 15,311 | 5.6 | 3,306 | 74.4 | 894.8 | 3,514 | 8.7 | 45.4 | 10,281 | 14.1 | 79.8 |
| 17 | 17,905 | 6.5 | 3,732 | 75.6 | 1,042.7 | 3,938 | 11.1 | 50.3 | 12,157 | 16.1 | 90.5 |
| 18 | 25,526 | 9.1 | 4,822 | 90.4 | 1,040.9 | 4,916 | 13.1 | 58.9 | 17,092 | 23.0 | 120.1 |
| 19 | 27,285 | 9.8 | 5,064 | 97.4 | 1,277.9 | 5,137 | 14.5 | 61.7 | 18,249 | 24.5 | 127.6 |
| 20 | 27,285 | 9.7 | 5,064 | 97.7 | 1,277.8 | 5,137 | 13.0 | 61.1 | 18,249 | 24.6 | 126.3 |
| 21 | 27,285 | 9.7 | 5,064 | 96.8 | 1,276.9 | 5,137 | 14.5 | 61.3 | 18,249 | 24.6 | 126.3 |
| 22 | 29,497 | 10.5 | 5,533 | 98.7 | 1,291.6 | 5,615 | 14.1 | 64.7 | 19,776 | 25.8 | 134.4 |
| 23 | 30,212 | 10.8 | 5,656 | 98.5 | 1,295.8 | 5,744 | 15.6 | 65.2 | 20,263 | 26.5 | 137.4 |
| 24 | 30,212 | 10.7 | 5,656 | 99.3 | 1,298.4 | 5,744 | 15.0 | 64.8 | 20,263 | 26.6 | 138.2 |
| 25 | 30,952 | 11.0 | 5,886 | 99.2 | 1,305.4 | 5,977 | 16.5 | 66.2 | 20,831 | 27.1 | 138.7 |
| 26 | 31,606 | 11.2 | 6,006 | 99.7 | 1,304.4 | 6,078 | 14.8 | 67.4 | 21,286 | 27.7 | 138.7 |
| 27 | 31,606 | 11.2 | 6,006 | 99.6 | 1,304.7 | 6,078 | 16.9 | 66.6 | 21,286 | 27.6 | 137.7 |
| 28 | 31,606 | 11.2 | 6,006 | 99.5 | 1,306.6 | 6,078 | 14.9 | 66.6 | 21,286 | 27.5 | 137.3 |
| 29 | 33,946 | 12.0 | 6,396 | 100.7 | 1,315.8 | 6,457 | 16.7 | 69.7 | 22,971 | 29.5 | 145.0 |
| 30 | 39,669 | 13.9 | 7,291 | 103.4 | 1,746.6 | 7,303 | 18.0 | 77.1 | 27,075 | 33.9 | 165.3 |

for some Internet-based applications can no longer be ignored. Creating an alternative (binary) serialization of the plain-text data can provide a good solution for this problem.

In this paper, we analyzed two standardized alternative XML serialization formats to address the verbosity issue: MPEG-B, the Binary MPEG format for XML, and ASN.1-PER, the Abstract Syntax Notation One with Packed Encoding Rules. Overhead reduction is achieved (1) by a compact representation of the plain-text XML data and (2) in MPEG-B, by standardizing a method to update (parts of) an XML tree.

We introduced a new kind of XML parser that is serialization-type agnostic. As such, applications can use this parser to handle XML-based data without being aware of the actual content encoding format. This parser shields the application developer from any additional complexity in order to support MPEG-B, ASN.1-PER, and ZIP compression. On top of that, application developers create applications supporting alternative serialization formats transparently. Thus, theusage of this parser enables transparent access to XML-based (meta)data by shielding users, i.e., application developers, from its encoding format which is aligned with the principles of UMA.

We evaluated the usefulness of MPEG-B, ASN.1, and the developed parser for two real-life applications: a UMA-enabled application where a client device can inform a server about its usage environment and an RSS application. The MPEG-B BiM serialization and the ASN.1-PER encoding were compared to a classical plain text compression technique, namely ZIP compression. These three binary serialization types and the traditional plain-text serialization are used in two modes, namely a full mode that handles complete and valid XML files and an update mode that only processes the differences.

The results show that BiM encoding of XML data reduces the required bandwidth and the associated costs by more than 92% for the first application and nearly 96% for the second application. This is achieved mainly thanks to the standardized update functionality of MPEG-B. The ZIP-compression and ASN.1-PER serialization types achieve a cost reduction in the update mode, which is a proprietary solution for these two

**Table 5** Results for use case 2: daily retrieval of an RSS feed — update mode

| Update mode | Plain text | | BiM encoded | | | ZIP compressed | | | ASN.1-PER | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Day | Byte size | Parse (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) | Byte size | Parse (ms) | Encode (ms) |
| 1 | 611 | 0.6 | 226 | 65.1 | 236.8 | 548 | 4.7 | 26.7 | 280 | 1.9 | 21.2 |
| 2 | 1,433 | 0.9 | 364 | 66.1 | 1,862.7 | 819 | 3.2 | 26.7 | 707 | 3.5 | 31.1 |
| 3 | 1,358 | 0.9 | 330 | 66.0 | 1,864.5 | 787 | 3.8 | 26.5 | 631 | 3.5 | 36.4 |
| 4 | 2,286 | 1.2 | 818 | 119.7 | 3,287.9 | 1,061 | 5.0 | 27.6 | 1,297 | 4.3 | 39.4 |
| 5 | 1,633 | 1.0 | 425 | 66.5 | 1,860.4 | 875 | 4.4 | 26.9 | 849 | 3.6 | 30.3 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 2,186 | 1.2 | 785 | 120.1 | 3,281.0 | 1,044 | 3.5 | 27.4 | 1,231 | 4.2 | 33.1 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 4,238 | 2.0 | 1,969 | 282.8 | 8,087.1 | 1,485 | 4.8 | 31.1 | 2,710 | 6.4 | 46.1 |
| 11 | 1,747 | 1.0 | 519 | 67.3 | 1,882.8 | 984 | 4.2 | 27.8 | 994 | 3.7 | 37.6 |
| 12 | 2,541 | 1.3 | 862 | 122.2 | 3,329.1 | 1,064 | 3.9 | 29.0 | 1,442 | 4.6 | 35.2 |
| 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | 1,808 | 1.0 | 427 | 67.5 | 1,865.0 | 884 | 3.8 | 27.1 | 936 | 3.8 | 31.8 |
| 16 | 3,740 | 1.7 | 1,338 | 178.2 | 4,926.8 | 1,211 | 4.5 | 29.6 | 2,204 | 5.8 | 41.1 |
| 17 | 3,421 | 1.6 | 1,389 | 178.5 | 5,012.9 | 1,314 | 4.5 | 29.5 | 2,176 | 5.3 | 40.2 |
| 18 | 8,448 | 3.3 | 4,208 | 681.6 | 19,220.9 | 1,809 | 6.0 | 35.9 | 5,235 | 11.1 | 64.4 |
| 19 | 2,586 | 1.3 | 861 | 126.3 | 3,351.8 | 1,064 | 4.6 | 28.4 | 1,457 | 4.6 | 35.7 |
| 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | 3,039 | 1.5 | 1,266 | 183.3 | 5,000.2 | 1,299 | 4.2 | 28.2 | 1,827 | 5.3 | 45.4 |
| 23 | 1,542 | 1.0 | 405 | 69.8 | 1,713.5 | 860 | 3.8 | 26.5 | 787 | 3.8 | 30.1 |
| 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | 1,567 | 1.1 | 469 | 69.6 | 1,745.8 | 925 | 4.0 | 26.8 | 868 | 3.6 | 30.5 |
| 26 | 1,481 | 1.0 | 396 | 69.7 | 1,684.2 | 848 | 3.5 | 27.0 | 755 | 3.5 | 30.2 |
| 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 29 | 3,167 | 1.5 | 1,287 | 184.8 | 5,081.7 | 1,202 | 4.4 | 28.5 | 1,985 | 5.3 | 39.7 |
| 30 | 6,550 | 2.6 | 3,168 | 475.2 | 13,007.9 | 1,775 | 5.1 | 33.2 | 4,404 | 8.5 | 58.1 |

**Table 6** GPRS price for use case 2 after 1 month

| | Byte size | GPRS price (€) | Cost reduction (%) |
|---|---|---|---|
| Full Plain text | 524,604 | 2.56 | |
| Full BiM encoded | 104,993 | 0.51 | 79.99 |
| Full ZIP compressed | 110,885 | 0.54 | 78.86 |
| Full ASN.1-PER | 352,100 | 1.72 | 32.88 |
| Plain-text updates | 55,382 | 0.27 | 89.40 |
| BiM-encoded updates | 21,512 | 0.11 | 95.90 |
| ZIP-compressed updates | 21,858 | 0.11 | 95.83 |
| ASN.1-PER updates | 32,775 | 0.16 | 93.75 |

GPRS price of a Belgian telecom operator (0.5€/ 100 Kb)

serialization types, close to the BiM serialization type. Nevertheless, MPEG-B BiM has a better compression ratio; BiM-serialized data can be processed during decoding; and BiM natively supports the update mode. These characteristics make BiM a good alternative serialization type with regards to compression efficiency and usability.

However, the tests also clearly demonstrate the very slow parsing and encoding of MPEG-B BiM-encoded XML data. This is partially explained due to the usage of the MPEG reference software implementation of MPEG-B, which is not optimized for speed. It is expected that (commercial or open source) implementations will exhibit a significantly better runtime behavior because the MPEG-B BiM specification does not have any inherent limitation that would prevent this. Currently, however, there are no such implementations available. Hence, BiM is currently not a recommended solution, as long as optimized encoders and decoders are not available. This is especially true for constrained

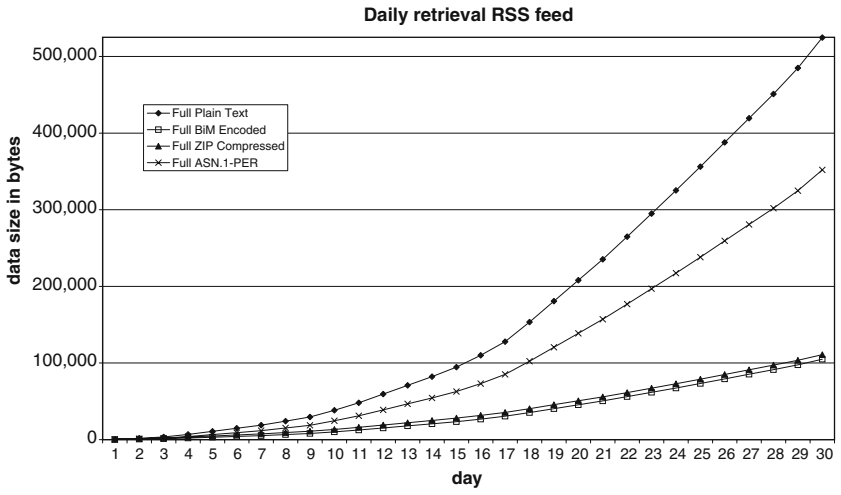


**Fig. 6** Cumulative byte size of daily retrieval of the RSS feed–full mode

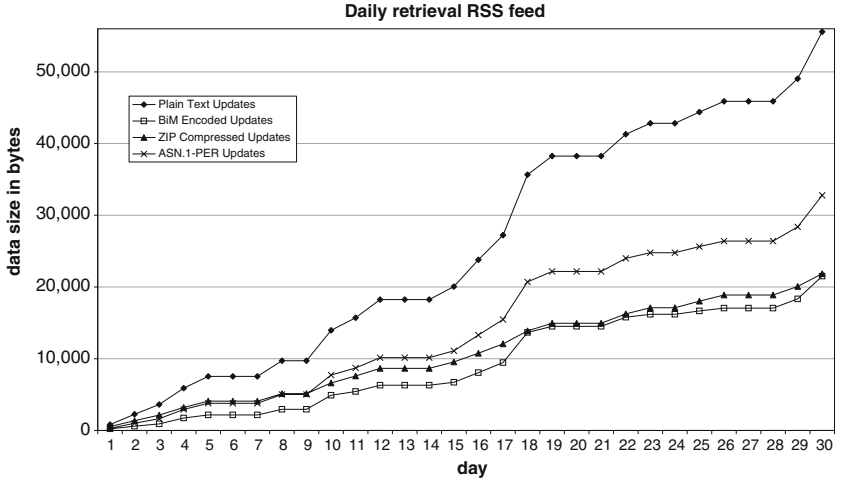


**Fig. 7** Cumulative byte size of daily retrieval of the RSS feed–update mode

devices as the additional processing time, and hence power consumption, nullifies BiM's advantages with regard to compression efficiency and native update capability.

The Abstract Syntax Notation One with Packed Encoding Rules does not live up to the expectation with regard to efficiency. Although it is much more efficient in processing data compared to the MPEG-B BiM technology, it is slower than ZIP compression and parsing. On top of that, ASN.1-PER can only handle XML-based data valid to a pre-determined and fixed XML Schema as the encoder/decoder software is schema specific, i.e., every XML Schema needs its own software module.

Finally, alternative XML serialization formats can address the verboseness and the non-existing update capabilities of XML trees. By letting XML parsers handle these additional serialization formats, the complexity of handling XML is not increased for the application developers. In fact, they do not need to be aware of the actual content encoding format. End users with mobile devices may benefit from such alternative serialization formats by means of paying less for actually sending or retrieving the same data.

**Acknowledgments** The research activities that have been described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), the Belgian Federal Science Policy Office (BFSPO), and the European Union.

## References

1. Barton, J.J., Thatte, S., Nielsen, H.F.: SOAP messages with attachments. W3C note (2000)
2. Cheney, J.: Compressing XML with multiplexed hierarchical PPM models. In: Proceeding of IEEE Data Compression Conference, Utah, USA (2001) , pp. 163–172

3. Cokus, M., Pericas-Geertsen, S.: XML binary characterization use cases. W3C working draft (2005)
4. De Sutter, R., De Keukelaere, F., Van de Walle, R.: Evaluation of usage environment description tools. In: Proceedings of the International Conference on Internet Computing, pp Las Vegas, USA 66–72 (2004)
5. De Sutter, R., Lerouge, S., Bekaert, J., Rogge, B., Van De Ville, D., Van de Walle, R.: Dynamic adaptation of multimedia data for mobile applications. In: Proceedings of the SPIE ITCom Internet Multimedia Management Systems III, Boston, USA vol. 4862, pp. 240–248 (2002)
6. De Sutter, R., Timmerer, C., Hellwagner, H., Van de Walle, R.: Evaluation of models for parsing binary encoded XML-based metadata. In: Proceedings of the IEEE International Symposium on Intelligent Signal Processing and Communication Systems, Seoul, Korea pp. 419–424 (2004)
7. Gudgin, M., Mendelsohn, N., Nottingham, M., Ruellan, H.: XML-binary Optimized Packaging. W3C recommendation (2005)
8. Heuer, J., Thienot, C., Wollborn, M.:Binary format. Introduction to MPEG-7: multimedia Content Description Language, pp. 61–80. Wiley, Newyork (2002)
9. Hunter, J.: An Overview of the MPEG-7 Description Definition Language (DDL). IEEE Trans Circuits Syst. Video Technol. **11**(6), 765–772 (2001)
10. ISO/IEC: Information technology — MPEG-7 — part 1: reference software status and workplan. Report no. ISO/IEC JTC1/SC29/WG11 MPEG/N6973 (2005)
11. ISO/IEC: Information technology MPEG-B part 1: Binary MPEG Format for XML. Report no. 23001-1:2006 (2006)
12. ITU-T, ISO/IEC: Encoding using XML or basic ASN.1 value notation. Report no. ITU-T Rec. X.693 (2001), ISO/IEC 8825-4:2001 (2001)
13. ITU-T, ISO/IEC: Abstract Syntax Notation One (ASN.1) Specification of Basic Notation. Report no. ITU-T Rec. X.680 (2002), ISO/IEC 8824-1:2002 (2002)
14. ITU-T, ISO/IEC: Specification of Packed Encoding Rules (PER). Report no. ITU-T Rec. X.691 (2002), ISO/IEC 8825-2:2002 (2002)
15. ITU-T, ISO/IEC: Mapping W3C XML Schema definitions into ASN.1. Report no. ITU-T Rec. X.694 (2004), ISO/IEC 8825-5:2004 (2004)
16. Kalden, R., Meirick, T., Meyer, M.: Wireless Internet Access based on GPRS. IEEE Pers. Commun. **7**(2), 8–18 (2000)
17. Karmarkar, A., Gudgin, M., Lafon, Y.: Resource representation SOAP header block. W3C recommendation (2005)
18. Liefke, H., Suciu, D.: XMill: An efficient compressor for XML Data. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, USA pp. 153–164. (2000)
19. Niedermeier, U., Heuer, J., Hutter, A., Stechele, W., Kaup, A.: An MPEG-7 tool for compression and streaming of XML data. In: Proceedings of the IEEE International Conference on Multimedia and Expo, Lausanne, Switzerland vol. 1, pp. 521–524. (2002)
20. Perkis, A., Abdeljaoued, Y., Christopoulos, C., Ebrahimi, T., Chicharo, J.: Universal multimedia access from wired and wireless Systems. Circuits Syst. and Signal Proces (Special issue on Multimedia Communications) **20**(3-4), 387–402 (2001)
21. Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M., Pelegri-Llopart, E.: Fast Web services. Sun developer network technical article (2003). http://java.sun.com
22. Sandoz, P., Triglia, A., Pericas-Geertsen, S.: Fast Infoset. Sun developer network technical article (2004). http://java.sun.com
23. Timmerer, C., Kofler, I., Liegl, J., Hellwagner, H.: An evaluation of Existing metadata compression and encoding technologies for MPEG-21 applications. In: Proceedings of the Seventh IEEE International Symposium on Multimedia (ISM'05), Irvine, USA pp. 534–539 (2005)
24. Vetro, A., Christopoulos, C., Ebrahimi, T.: Universal multimedia access. IEEE Signal Process (Special issue) **20**(2), 16 (2003). Guest editors
25. Vetro, A., Timmerer, C.: Overview of the digital item adaptation standard. IEEE Trans Multimed (Special Issue on MPEG-21) **7**(3), 435–445 (2005)
26. Williams, S.D., Haggar, P.: XML binary characterization measurement Methodologies. W3C working draft (2005)
27. Ziv, J., Lempel, A.: Compression of individual sequences via variable rate coding. IEEE Trans Inf Theory **24**(5), 530–535 (1978)
28. Ziv, J., Lempel, A.: A Universal Algorithm for Sequential Data Compression. IEEE Trans Inf Theory **23**(3), 337–343 (1978)

## Authors' biographies

**R. De Sutter** received master degree in computer science from Ghent University, Belgium, in 1999. He joined the Multimedia Lab in 2001 where he is currently working toward Ph.D. degree. His research interests include video coding technologies, usage context modeling and negotiation, and content adaptation.

**S. Lerouge** received his master degree in computer science from Ghent University, Belgium, in 2001. Since then, he started working towards Ph. D. degree in the Multimedia Lab, which he obtained in 2005. His research focuses on applications that use scalable video coding, in particular the maximization of the visual quality in constrained environments.

**P. De Neve** was born in Ghent, Belgium, in 1970. He received his M.Sc. and Ph.D. degrees in Engineering from the Ghent University, Belgium, in 1995 and 2000, respectively. He is working as an assistant professor at the Multimedia Lab. His major research interests include color image processing, video coding, coding and description of multimedia data.

**C. Timmerer** received the Dipl.-Ing. degree in Applied Informatics at University of Klagenfurt, Department of Information Technology (ITEC). He joined the University of Klagenfurt in 1999 and is currently a University Assistant and chairs the IT administration group of the Department of Information Technology. His research interests include coding-format agnostic resource adaptation, transport of multimedia content, multimedia adaptation in constrained and streaming environments, and distributed multimedia adaptation.

**H. Hellwagner** received his Dipl.-Ing. degree (in Informatics) and Ph.D. degree (Dr. techn.) in 1983 and 1988, respectively, both from the University Linz, Austria. From 1989 to 1994, he was senior researcher and team/project manager at Siemens AG, Corporate R&D, Munich, Germany. From 1995 to 1998, he was associate professor of parallel computer architecture at Technische Universität München (TUM). Since late 1998, he has been a full professor of computer science at the Department of Information Technology (ITEC) at the University Klagenfurt, Austria. His current research areas are distributed multimedia systems, multimedia communications, and Internet QoS.

**R. Van de Walle** received his M.Sc. and Ph.D. degrees in Engineering from Ghent University, Belgium, in 1994 and 1998, respectively. After a visiting scholarship at the University of Arizona (Tucson, USA), he returned to Ghent University, where he became full professor of multimedia systems and applications, and head of the Multimedia Lab. His current research interests include multimedia content delivery, presentation and archiving, coding and description of multimedia data, content adaptation, and interactive (mobile) multimedia applications.