Evaluation of Models for Parsing Binary Encoded XML-based Metadata

Robbie De Sutter[1], Christian Timmerer[2], Hermann Hellwagner[2], and Rik Van de Walle[1]
[1] Multimedia Lab
Ghent University – IBBT, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
E-mail: {robbie.desutter;rik.vandewalle}@ugent.be
[2] Dept. of Information Technology (ITEC)
Universität Klagenfurt, Universitätsstraße 65-67, A-9020 Klagenfurt, Austria
E-mail: {christian.timmerer;hermann.hellwagner}@itec.uni-klu.ac.at

Abstract: In multimedia applications, XML is being increasingly used to represent metadata; examples are MPEG-7 multimedia description schemes and MPEG-21 usage environment descriptions. As with the media data, the size of, or the overhead induced by, the XML metadata is important, particularly when used on constrained mobile devices. Therefore, compression (binary encoding) of the XML data becomes relevant to reduce this overhead. Within the MPEG-7 standardization effort, a Binary Format for Metadata (BiM) was developed, providing good compression efficiency and facilitating random access into, and manipulation of, the binary encoded bit stream.

In order to support processing of metadata streams in the binary domain and making this task for client applications as simple as possible, we are developing a universal parser for handling both plain text and binary encoded XML-based metadata. The parser exposes a single interface making it transparent for the application whether a plain text or a binary XML document is being processed.

As part of this effort, this paper provides a detailed study of five existing XML parser models and evaluates their applicability to serve as a model for parsing binary XML data, encoded using the BiM codec. Additionally, the parser models are investigated against important usage scenarios enabled by BiM, such as dynamic updates of XML data. From the five models, two are rejected and one is only applicable for domain specific applications. Of the remaining two, one model is proposed as preferred model because of different advantages over the other model.

1. Introduction

Within the last decade, research in the area of multimedia communication was more or less driven by *Universal Multimedia Access* (UMA) which aims to enable seamless access to a rich variety of multimedia content through heterogeneous networks and end-user devices [1]. To support the concepts of UMA, various metadata standards emerged for associating additional information, i.e., metadata, to the content [2] as well as for describing the capabilities and characteristics of terminals and networks [3]. The preferred format of metadata is the *Extensible Markup Language* (XML). Due to the fact that XML is based on plain text, the metadata can be easily used by several applications, ranging from educational services to geographical information systems and surveillance, for instance. However, this represents just the tip of the iceberg [4]. The recently finalized part 7 of the MPEG-21 Multimedia Framework [5], better known as MPEG-21 Digital Item Adaptation (DIA) [3], is a perfect example. DIA standardizes how to describe the usage environment as well as how to describe the structure of multimedia coding formats. The latter makes it possible to easily adapt multimedia resources in a coding-format independent way by manipulating the corresponding XML descriptions [6].

In practice, however, using XML has some drawbacks. The most import drawback is that the XML language is inherently verbose. For bandwidth-constrained devices such as cell phones, or in streaming applications, this can be a burden. Even when General Packet Radio Services (GPRS) or other higher-speed wireless networks are being used - where bandwidth is no longer a tight constraint - it must be taken into account that the user may be charged for the transferred data volume. Thus, the metadata overhead needs to be decreased in order to reduce the user's cost which contributes to a positive user experience. As a side effect, this may improve the performance of the device by decreasing the amount of data it has to process.

Within the World Wide Web Consortium (W3C) and the Moving Picture Experts Group (MPEG), this verbosity issue has been recognized. While the former is investigating this issue in a recently established working group [7], the latter has already standardized a method to encode XML data in binary form, known as *Binary Format for Metadata* (BiM) [8]. The method was originally intended to binary encode MPEG-7 metadata; however, due to its generic design most other XML data can be binary encoded by BiM as well. For instance, in the course of the MPEG-21 development, BiM will be improved to provide optimal support for metadata developed within this standardization activity.

Although binary encoding of XML data solves the verbosity problem, this technique also eliminates one of the key benefits of XML: its plain text property and thus human readability. However, XML merely struc-

Acknowledgment. The research activities described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Fund for Scientific Research-Flanders (FWO-Flanders), the Belgian Federal Science Policy Office (BFSPO), and the European Union.

tures the information and usually the applications become aware of the semantics carried within the XML document, by reading, interpreting, and manipulating the XML data. For this purpose, applications make use of an XML parser which should not be restricted to handling plain text XML data only.

In this paper, we look at the requirements to create a parser that is capable of handling regular plain text XML and binary encoded XML (see Fig. 1). Therefore we provide a detailed study of existing XML parser models and evaluate their usefulness and applicability to serve as a model for parsing binary encoded XML data using the BiM codec. Additionally, the parser models will be investigated against usage scenarios enabled by BiM.

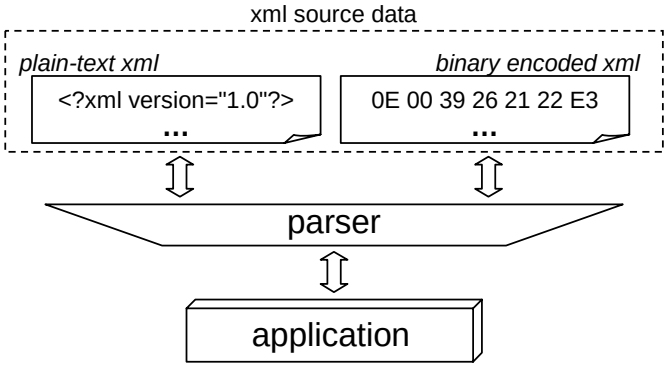

Figure 1. A client application handles plain-text or binarized XML through a parser.

The remainder of this paper is organized as follows. Section 2 describes the typical operations which an XML parser should support. Subsequently, we discuss five common XML parser models and their characteristics in Section 3. In Section 4 we discuss BiM and the validation of the parser models regarding their applicability to create a compliant parser for binary encoded XML data. Finally, the paper is concluded in Section 5.

2. Common XML Parser Functionalities

While the available XML parsers differ in the details of the functionality provided, they expose some common basic operation sets.

Hereafter a *client application* is the application that is using a parser. Further we define an *XML token*, or simply *token*, as the combination of XML markup and XML element content, both defined in [9].

2.1 Bootstrap

The *bootstrap* is the entry point of the parsing operation. In order to allow different implementations of the same parser model, an interface offering the bootstrap operations must be defined. The minimal operations are:

- Resolve and open the physical source XML data stream such as a file, an HTTP connection, and so on.
- Detect the encoding of the XML data stream. This can be accomplished by using the information in the XML prolog or by using a character encoding detection algorithm such as suggested in Appendix F of [9].
- Prepare the stream for navigation and token operations as discussed in subsequent subsections.

Specific implementations of a parser model can offer additional features such as enabling XML validation, performing normalization, supporting entity replacements, etc.

2.2 Navigation

The term *navigation* stands for the ability to move through the XML data stream to reach a specific token.

A. Simple forward navigation

Every parser model must offer simple forward navigation, either explicit or implicit. Simply forward navigation allows the client application to move through the XML stream one token at a time. This basic operation is usually extended – and possibly concealed – by more advanced forward navigation methods such as: *go to the next (start) tag*, *go to the first child element*, *go to the last sibling*, and similar operations.

B. Simple backward navigation

The basic operation of this kind of navigation is to move back one token. This basic operation is usually extended by more advanced backward navigation methods. Typical examples are: *move to the parent node*, *move to previous sibling*, and *move to the root node*. Simple backward navigation is offered by some parser models.

C. Random navigation

This is the most flexible way for navigating through the XML data stream. It allows the client application to "jump" directly to certain parts in the XML stream. For example, the client application can: *jump to an element with a specific ID attribute value*, *jump to a token with a certain fully qualified name*, and *address a token by an XPath*[1] *expression*. It depends on the parser model if this form of navigation is available.

2.3 Token Operations

By applying the navigation operations, the client application can select a single token in the XML data stream. We call this token the *current token*. The next step is to act upon the current token by, e.g., consuming or manipulating it.

A. Token Consumption

All parser models must allow token consumption, also called *read* functionality. In this context, "read" means to investigate the current token and harvest the value of the given token.

The main operation is the retrieval of the token type, i.e., identify the current token as an element (a start tag or an end tag), an attribute, text data, a CDATA

[1] W3C Recommendation, XML XPath Language, available at http://www.w3.org/TR/xpath

section, a comment tag, a processing instruction, or ignorable white space. Each token exposes its value. Furthermore, the start tag, the end tag, and the attribute tokens also expose their name and namespace context[2]. Advanced parsers can provide additional information, such as the number of attributes and the number of children in case of a start tag.

B. Token Manipulation

Some parser models allow the *manipulation* of the XML data stream by adding new tokens or by deleting selected tokens. Changing the current token can be seen as a delete operation followed by an add operation. The add operation has multiple provisions. For example, it is not allowed to add a processing instruction to an attribute token. The delete operation on an element token must be seen as the removal of the current element and all child elements, if any.

After all token manipulations are executed, the XML data should be at least well-formed.

2.4 Auxiliary Operations

The auxiliary operations are not necessary for the parsing of an XML data stream and as such are not part of the parser models. However, they are provided by most parser implementations as an aid for the client application, such as localization of the current token in the physical stream, token comparison, namespace prefix lookup, duplication of a token, and so on.

3. Survey of Existing XML Parser Models

If an application processes an XML data stream, it accesses the XML data indirectly by using an XML parser as shown in Fig. 1. Application developers can choose from a wide range of different parsers, each parser having its own possibilities, characteristics, and fields of application. All these parsers are built according to a particular XML processing model. When studying the available parsers, five distinct models can be identified.

This section describes each model in detail, starting from the oldest and original model developed by W3C, namely the Tree Model. The other models described are the Push and Pull Models, the Cursor Model, and the Mapping Model. For each model, an example of a parser and the typical model characteristics are listed. This information is also summarized in Table 1. Other available tools to process XML are briefly discussed in the final subsection.

3.1 The Tree Model

This is the original and thus oldest model whereby the tree structure characteristic of an XML document is exploited. The XML tree is reconstructed in memory in such a way that it closely reflects the XML data model [10]. The parser grants the client applications access to the in-memory tree and offers navigation throughout the tree.

Example parser: Document Object Model (DOM)[3]

Characteristics:

- The complete navigation operation set is available.
- Token consumption and token manipulation.
- The client application fully controls the parser.
- Parsing the XML data is slow as the complete data stream must be read in order to build the in-memory tree structure before it is available to the client application. As such, the XML data stream must be completely received, e.g., the data must be entirely downloaded from a HTTP connection, before the client application can act upon the data.
- High memory requirements because the complete XML data stream is mimicked in memory. For memory constrained environments such as a cell phone, these high requirements can be problematic.

3.2 The Push Model

The second model that emerged after the Tree Model was the Push Model. Its main goal was to address the shortcomings of the Tree Model. The Push Model states that a compliant parser reads the data stream and for each XML token it encounters it generates an event. 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.

Example parser: Simple API for XML (SAX)[4]

Characteristics:

- Simple forward navigation only.
- Token consumption only.
- The parser is in control, the client application does not know when and if an event will be thrown. As such, implementing the client application is by some seen as difficult and unnatural.
- Very fast parsing.
- Very low memory requirements because the XML data is not kept in memory.

3.3 The Pull Model

Because the event oriented programming model is seen by some as a disadvantage, the Pull Model was developed. Pull Model compliant parsers read only one single token after being instructed by the client application to do so. Information about the token can be requested from the parser by the client application. As such, the information is pulled from the parser by the client application.

Example parser: XMLPull[5]

Characteristics:

- Simple forward navigation only.
- Token consumption only.

[2]The *namespace context* comprises the used namespace, an overview of all available namespaces for the given token and the mapping of the namespace prefixes to the actual namespace.

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

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

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

Table 1. Comparing XML parser models

	Tree Model	Push Model	Pull Model	Cursor Model	Mapping Model
Navigation					
forward navigation	yes	yes	yes	yes	yes
backward navigation	yes	no	no	yes	yes
random navigation	yes	no	no	yes	yes
Token operations					
token consumption	yes	yes	yes	yes	yes
token manipulation	yes	no	no	optional	yes
Processing speed					
parsing	slow	fast	fast	fast	slow
token consumption	fast	medium	medium	slow–fast	fast
token manipulation	fast	n/a	n/a	fast	fast
Memory requirements	high	low	low	low–high	medium

• The client application controls and instructs the parser when to act upon the data stream.
• Very fast parsing.
• Low memory requirements as only the current XML token resides in memory.

3.4 The Cursor Model

The Cursor Model is very similar to the Pull Model, but allows random access throughout the XML data stream by directly addressing XML tokens using an XPath expression. The random access makes it possible to target a specific part of the data and as such create a view on the data. These views can be further examined by the client application one token at a time, like using a Pull Model parser. As such, the Pull Model can be seen as a forward only and token consumption only version of the Cursor Model.

Example parser: .NET XPathNavigator[6]

Characteristics:

• The complete navigation operation set is available.
• Token consumption and optionally token manipulation.
• The client application controls the parser.
• Very fast parsing. However, depending on the XPath expression, execution can vary in time and is parser implementation dependent.
• Depending on the specific parser implementation, this model has very low to high memory requirements.

It is important to note that a specific Cursor Model parser implementation must make a tradeoff between processing speed and memory requirements. Fast processing and especially fast XPath navigation will require to load the complete XML document into memory. Enabling write access automatically implies that the very low memory requirements are practically unreachable.

3.5 The Mapping Model

The fifth model differs from previous models as it focuses more on the XML content than on the XML semantics. First, the model allows the creation of object oriented classes based on the XML data structure. A Document Type Definition (DTD), schema language, or XML data stream analysis can be used to generate the classes. Next, the XML data, and more specifically the XML content, is mapped to instances of the created object oriented classes. The client application can use the instantiated object oriented classes directly just as any other class instances.

Example parser: .NET XmlSerializer[7]

Characteristics:

• The complete navigation operation set is available.
• Token consumption and token manipulation.
• The client application uses the instantiated classes directly without the need to steer the parser.
• The creation of the classes adds an additional time-cost factor to the parsing of the XML data.
• This model has medium memory requirements as the complete XML data is stored in memory. However because the classes are optimized to the characteristics of the XML structure and used data types, it requires less memory than the Tree Model.

It must be mentioned that this model has several issues that prevent a complete correct mapping of the XML data model, such as the impracticality of handling mixed content and preserving element sequence. Due to these issues and the additional time-cost factor for class creation, the mapping model is mostly used for application domain specific applications whereby the structure of the XML data is known in advance.

4. Parsing Binary Encoded XML

4.1 Introduction

Our aim is to develop a universal parser capable of handling plain text XML as well as binary encoded XML. A typical usage scenario is as follows. A client application receives an XML document (in plain text or binary encoded) and uses a parser to handle the document. The parser is compliant to one of the five models

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

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

as described above and exposes its operation set to the client application. The client application handles the XML data through the exposed interface, unaware if the parser is handling the XML document in binary or plain text format.

4.2 MPEG-7 Binary Format for Metadata

This subsection is intended to provide basic information about the BiM decoder. The reader is referred to [11] for a detailed and complete description.

XML is binary encoded into a sequence of so-called *Access Units* (AUs). Each AU is a standalone entity that can be decoded by a BiM decoder. It contains *schema information* and *Fragment Update Units* (FUUs). The schema information is used to configure the decoder which decodes the FUUs sequentially. Each FUU contains the *update command* (i.e., add, replace, delete or reset), the *update context* (i.e., identifying the location in the XML document for applying the update command) and the *update payload* (i.e., the actual data). Note that the update context uses XPath syntax which identifies the exact location, i.e., one node where the update command will be applied. The decoding results in a *description tree*, comparable to an XML tree, and valid against the received schema information.

4.3 Usage Scenarios

In order to evaluate the parser models for handling BiM documents, we consider the following two scenarios: traditional XML handling and handling dynamic updates of XML data.

In the first scenario, i.e., traditional XML handling, the complete XML document is known and will become available for the parser. This also includes XML documents being downloaded from a network connection. This scenario is common in current XML processing where random access to the information in the XML document is desired, e.g., for transforming the whole XML document into another one or for searching the metadata describing a multimedia repository.

The second scenario takes dynamic updates of an XML document into account which is an integral part of the functionality provided by MPEG-7 BiM. BiM allows the stepwise construction of the description tree by the incorporation of dynamic changes, i.e., updates, into the existing XML description tree. These updates may become available during the actual parsing process.

The usefulness of the second scenario is illustrated by the following example. Suppose a cell phone describes its usage environment and sends this information to a server in order to receive an optimized video stream (see also [12]). During the consumption of the video stream the usage environment may change, e.g., by connecting the cell phone to a power outlet. Consequently, the usage environment information needs to be updated and communicated to the server. The first possibility is to send the complete updated usage environment information again. This is however not advisable for the same reasons as discussed in Section 1. A more sophisticated possibility is to send only the updated information to the server using a predefined – and thus application specific – message format. Due to interoperability reasons, this is also not an ideal solution. Therefore, a standardized solution for the second scenario, e.g., dynamic updating of XML documents using MPEG-7 BiM, is required which allows the cell phone to send only its updated usage environment information to the server in an optimal and application independent manner.

4.4 Evaluating the XML Parser Models

In the following, the XML parser models as introduced in Section 3 are discussed according to their applicability for the aforementioned usage scenarios.

A. Scenario 1

For this scenario, all models can be used. The Push and Pull Models are best suited due to fast processing with the lowest memory requirements because the binary encoded data does not need to be kept in memory and can be discarded after processing. Nevertheless, there are some prerequisites: it is necessary that the bit stream only contains one AU and one FUU or AUs and sequences of FUUs that only add information to the description tree in a depth-first order. If a bit stream does not fulfill these prerequisites, it is possible that a FUU modifies the information stored in previous FUUs. If this is the case, it is necessary to create the description tree in memory to fully support the FUU update commands. It is also necessary that the BiM parser needs to process all AUs and FUUs such that all FUUs which may possibly update the information stored in the current description tree are dealt with. Because of this, the Push and Pull models forfeit their advantage concerning the memory requirement. A possible workaround for this issue is to preprocess the BiM bit stream in such a way that it meets the prerequisites as mentioned above. Note that this workaround is only useful if the same BiM bit stream is used multiple times. The identification whether or not a given bit stream satisfies these prerequisites is not straightforward.

All other parser models retain their typical characteristics. As such the Cursor Model has a slight advantage over the Tree Model as it is slightly more memory efficient. The Mapping Model can be used for application domain specific applications.

B. Scenario 2

For this scenario, parser models offering only simple forward navigation cannot be used due to the possibility that updates in the middle of parsing an XML document may occur. This means that the Push and Pull Models are not suitable for this scenario. When new AUs are received, they can update information that has already been processed by the parser. Parsers with only forward navigation cannot handle these changes because the client application has no access to the information which should be updated. Even if the client application is informed that the information was changed, it is

necessary to restart the parsing operation.

The Mapping Model seems to provide the best solution because the newly received AUs can manipulate the instantiated classes directly in a fast and straightforward way. Thus the client application is immediately aware of the modified information. For the sake of completeness, however, it must be noted that BiM allows to update and to add new (fragments of) XML Schema elements through the schema updates. While some programming languages allow to create and to modify classes at runtime, these constructs are not optimized (neither in memory usage nor in processing speed) and difficult to use by a client application. As such, this solution is advised if all XML Schema elements are known in advance (during development of the parser) such that optimized parsers can be created. This is especially applicable for standardized and normative schemas.

Finally, the Cursor Model has a slight advantage over the Tree Model as it has lower memory requirements. Furthermore, the fragment update context uses the XPath syntax to select the update location for which the internal XPath resolver of the Cursor Model could be re-used. Moreover, the same XPath expression can be further used to signal the client application that updates are available without any additional processing steps.

5. Conclusion

As XML is more and more being used in multimedia applications to represent the complementary metadata, the verbosity of the format is becoming a concern, especially in constrained environments. Using MPEG-7 BiM makes it possible to encode XML data into a binary bit stream with good compression while retaining random access and manipulation of the XML data.

In this paper we investigated the prerequisites in order to create a universal parser capable of handling both plain text and binary encoded XML-based metadata. Therefore five XML parser models were investigated, namely the Tree Model, the Push Model, the Pull Model, the Cursor Model, and the Mapping Model. We evaluated the models with respect to their capability to handle BiM bit streams in two usage scenarios: traditional XML handling and handling dynamic updates of XML data.

This research proves that, while the Push and Pull parser models are fast and have the lowest memory requirements, these models are not appropriate to parse BiM bit streams. For the traditional XML handling scenario they tend to loose their low memory advantage and they are not suitable for the second scenario due to their forward only navigation capability. The Mapping Model proves to be a very good model, however only for predefined domain specific applications. Finally, the Cursor Model is to be preferred over the Tree Model, firstly, due to its better memory requirements, and secondly, because of the inherent XPath support of the model.

It is therefore clear that the way to go for creating a universal parser capable of handling plain text and binary encoded XML data is by creating a parser compliant to the Cursor Model.

References

[1] R. Mohan; J. R. Smith; C.-S. Li, "Adapting Multimedia Internet Content for Universal Access," in *IEEE Trans. Multimedia*, Vol. 1, No. 1, 1999 pp. 104-114.

[2] B.S. Manjunath; P. Salembier; T. Sikora; Eds., "Introduction to MPEG-7: Multimedia Content Description Language," John Wiley & Sons, ISBN: 0471486787, 2002.

[3] A. Vetro; C. Timmerer; S. Devillers, "Digital Item Adaptation," in *The MPEG-21 Book*, John Wiley & Sons, 2004.

[4] Frank Nack; Adam T. Lindsay, "Everything you want to know about MPEG-7: Part1," in *IEEE MultiMedia*, October 1999, pp. 64-73.

[5] I. Burnett; R. Van de Walle; K. Hill; J. Bormans; F. Pereira, "MPEG-21: Goals and Achievements", in *IEEE Multimedia*, 2003, pp. 60-70.

[6] C. Timmerer; G. Panis; H. Kosch; J. Heuer; H. Hellwagner; A. Hutter, "Coding format independent multimedia content adaptation using XML," in *Proceedings of SPIE ITCom 2003 on Internet Multimedia Management Systems IV*, vol. 5242, September 2003, pp. 92-103.

[7] M. Cokus; S. Pericas-Geertsen, "XML Binary Characterization Use Cases," Technical Report World Wide Web Consortium (W3C), Available: http://www.w3.org/TR/2004/WD-xbc-use-cases-20040728/, July 2004.

[8] "Information Technology - Multimedia Content Description Interface - Part 1: Systems FDAM Amd/1," Technical Report MPEG, ISO/IEC JTC1/SC29/WG11 N6326, July 2004.

[9] T. Bray; J. Paoli; C. M. Sperberg-McQueen; E. Maler; F. Yergeau, "Extensible Markupt Language (XML) 1.0 (Third Edition)," Technical Report World Wide Web Consortium (W3C), Recommendation, Available: http://www.w3.org/TR/2004/REC-xml-20040204/, February 2004.

[10] B. Bos, "The XML Data Model," Technical Report World Wide Web Consortium (W3C), Available: http://www.w3.org/XML/datamodel.html.

[11] U. Niedermeier; J. Heuer; A. Hutter; W. Stechele; A. Kaup, "An MPEG-7 tool for compression and streaming of XML data," in *The 2002 IEEE International Conference on Multimedia and Expo (ICME)*, vol. 1, August 2002, pp. 521-524.

[12] R. De Sutter; S. Lerouge; W. De Neve; P. Lambert; R. Van de Walle, "Advanced mobile multimedia applications using MPEG-21 and time-dependent metadata," in *Proceedings of SPIE ITCom 2003 on Internet Multimedia Management Systems IV*, vol. 5241, September 2003, pp. 147-156.