Generating MPEG-21 BSDL Descriptions Using Context-Related Attributes

Davy De Schrijver[1], Wesley De Neve[1], Koen De Wolf[1], and Rik Van de Walle[2]
Department of Electronics and Information Systems - Multimedia Lab
[1]Ghent University - IBBT
[2]Ghent University - IBBT - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
Phone: +32 9 331 49 56
email: {davy.deschrijver;wesley.deneve;koen.dewolf;rik.vandewalle}@ugent.be

Abstract

In order to efficiently deal with the heterogeneity in the current and future multimedia ecosystem, it is necessary that content can be adapted in a format-agnostic manner. A first step toward a solution, able to fulfill the just mentioned requirement, is to rely on a scalable video codec and to describe the high-level structure of the resulting bitstreams in such a way that every terminal can understand it, in particular by using XML. This paper describes how such descriptions can be generated by making use of the media format independent BintoBSD tool of the MPEG-21 BSDL standard. However, regarding the current status of BSDL, it is impossible to create a description in real time and to keep the generation speed constant over the complete sequence. In this paper, we describe a number of extensions and algorithmic modifications that make it possible to generate a description of a bitstream in real time and at a constant speed. Our approach results in a significant reduction of the original execution times (up to 99% for the H.264/AVC coding format) and in a constant memory usage.

1 Introduction

The current multimedia ecosystem consists of terminals, networks, and users that are having dissimilar characteristics with regard to the consumption of multimedia content. This results in a need for scalable content in order to tackle this heterogeneity. However, scalable coding alone is not sufficient for creating a framework for Universal Multimedia Access (UMA, [8]), giving content providers the opportunity to create content once and to publish it to every possible device, at any given time. One also needs a content adaptation system. A format-agnostic architecture is the main topic of this paper. To be more specific, we will have a look at a system that allows us to describe the high-level structure of bitstreams in XML (eXtensible Markup Language); in particular, by making use of the MPEG-21 BSDL (Bitstream Syntax Description Language) standard. This standard specifies how XML-based bitstream structure descriptions can be generated in an interoperable way and how a bitstream can be reconstructed from an adapted high-level description in an interoperable way. However, the current BSDL reference software, in particular version 1.2.1, is characterized by a decreasing generation speed and an increasing memory usage during the generation of a description, resulting in unacceptable execution times. For example, to describe a video stream of 13 seconds encoded by an H.264/AVC codec, the BSDL software needs 20 minutes to generate the bitstream description in XML. This is of course unacceptable. The proposal discussed in this paper solves these problems by adding a few extensions to the BSDL standard, and making it possible to generate real time video bitstream descriptions in BSDL for the first time.

The outline of the paper is as follows: in Section 2, we discuss how a bitstream structure description can be generated by relying on the BSDL standard. We also provide a performance analysis of the current reference software, as well as a number of requirements that have to be satisfied to make BSDL usable in practical situations. In Section 3, an overview is given with regard to our solution for the problems as mentioned in Section 2. We introduce a few new BSDL extensions and discuss how these extensions can be used. We also look at the algorithmic modifications that have to be applied to the BSDL software. The performance of our new algorithm is compared with the performance of the old algorithm in Section 4 and a discussion of the obtained results is provided as well. Section 5 concludes this paper.

2 Analysis of the bitstream description generation process

2.1 Bitstream Syntax Description Language

The Bitstream Syntax Description Language (BSDL) is part of the bigger Digital Item Adaptation standard (DIA) of MPEG-21 [2]. DIA defines mechanisms for the adaptation of a Digital Item and the resources that the Digital Item contains. The goal of DIA is to obtain an interoperable transparent multimedia access framework, taking into account the network, terminal and user characteristics. Therefore, it is necessary to describe a bitstream in a standardized manner such that every device in this framework can adapt the content in a transparent way. To realize this scenario, BSDL defines a framework that enables the description of the high-level structure of a bitstream in XML. By a high-level description, we mean that a bitstream is described for example on a frame-per-frame basis and not on a bit-per-bit basis. Based on these Bitstream Syntax Descriptions (BSDs), an adaptation can be executed. In Figure 1, the global operating procedure of the BSDL framework is given. As one can see in the figure, we start from

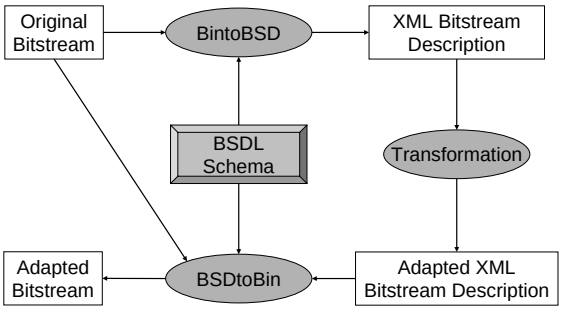

Figure 1. BSDL Framework

a given bitstream (preferably a scalable one) that is encoded with a certain codec. Dependent on the codec used, a BSDL schema is developed that represents the high-level structure of possible bitstreams generated by the codec in question. The structure of a BSDL schema is standardized in the MPEG-21 DIA specification [1] together with the functioning of the BintoBSD tool. Every MPEG-21 DIA compatible terminal can generate a BSD from a bitstream once the schema is known. At the moment that we have the bitstream description in XML, it is possible to adapt the description. This adaptation can use the characteristics from the network or terminal to steer the XML transformation. How the transformation has to be realized is not specified in the standard and it is up to the implementer of the framework. One can for example use of eXtensible Stylesheet Language Transformations (XSLT, [7]); Streaming Transformations for XML (STX, [3]) or an implementation based on an XML API (such as Simple API for XML, SAX, or Document Object Model, DOM). The final step in the framework is the regeneration of the adapted bitstream. Therefore, the BSDtoBin tool is needed, taking as input the adapted description, the BSDL schema and mostly the original bitstream. The functioning of this tool is also standardized within the DIA standard.
This implies that a BSDL framework consists of three (software) implementations: the BintoBSD tool to generate a description; the transformation to execute the adaptation of the XML description; and finally the regeneration of an adapted bitstream by means of the BSDtoBin tool. In this paper, attention is paid to the BintoBSD tool because the current version has problems in terms of unacceptable high execution times and memory consumption. Therefore, this paper presents some extensions and algorithmic modifications on the BSDL specification, solving the detected problems. The other two tools contain less problematical issues and are already discussed in other papers such as [6], [4], or [5].

As already mentioned, the MPEG-21 DIA standard defines how a BSDL schema has to be constructed by using the standardized BSD Language and how the BintoBSD and BSDtoBin tools have to work in an MPEG-21 DIA compliant terminal. BSDL is based on some restrictions to W3C XML Schema (only a limited number of data types is incorporated in BSDL) and on some extensions to W3C XML Schema. These extensions introduce new datatypes such as `bs1:byteRange` (which consists of two non-negative integers whereby the first one refers to a startbyte in the bitstream and the second one represents the length of the range); `bs2:length` (which gives the parser an indication on how many bytes to read); et cetera. Another extension is the introduction of new attributes such as `bs1:ignore` (which indicates that parsers have to ignore this element during the process);`bs2:nOccurs` (which specifies the number of occurrences of an XML particle); `bs2:if` (this makes it possible to make a particle conditional in a BSDL schema); et cetera. It is important to note that most of the BSDL-specific attributes will contain an XPath expression that has to be evaluated in order to obtain the value of the attribute. This evaluation should be done by the BSDL parsers (in particular, by the BintoBSD parser; the BSDtoBin parser does not need this information to generate the correct bitstream from a given description).

2.2 Performance of the description generation process

The functioning of the BintoBSD tool is the core topic of this paper. In the specification, one can find an explanation on how the tool has to work. The first step is the read in of the BSDL schema and the generation of an inter-

nal structure of this schema. An iterative process will traverse the structure to generate the desired description. The BSDL schema contains the `bs2:rootElement` attribute, which gives the parser an indication about what element to use to start the parsing process with. Once this process is started, the description is generated. In this second step, every time that the parser reads a chunk from the bitstream (as indicated by the internal structure of the schema) the corresponding element will be written to the description. This can be a file or another stream such as a URL connection in case of a streaming application. As we have already mentioned, certain BSDL attributes contain an XPath expression that has to be evaluated in order to know the value of the attribute. To give a parser the opportunity to evaluate an arbitrary XPath expression, an internal representation of the already generated description has to be kept in memory (for example in a DOM tree). That means that every element that is created and sent to the output stream, must also be kept in the internal representation. For every evaluation of an XPath expression that is needed, this internal description will be used.
A discussion concerning the performance of the BintoBSD tool can be found in Section 4.2. In the second column of Table 3, one can see the unacceptable high execution times of the software that implements the above algorithm of the BintoBSD tool (as implemented in version 1.2.1 of the MPEG-21 DIA reference software). To understand why these times are so high, we provide a profiling diagram of the software in Figure 2 to give an idea where most of the time is spent during the execution of the tool. This figure

Method	Inherent time	Invocations
com.sun.org.apache.xpath.internal.XPathAPI.eval	2.438 s (94 %)	13.277
org.iso.mpeg.mpeg21.dia.bsdl.XSD.datatypes.AllBinary.readBufFromBitstream	25 s (0 %)	191
org.iso.mpeg.mpeg21.dia.bsdl.utils.ByteArray.<init>	16 s (0 %)	844.909
org.iso.mpeg.mpeg21.dia.bsdl.utils.ByteArray.isEqual	16 s (0 %)	1.689.045
org.iso.mpeg.mpeg21.dia.bsdl.io.BufferedPosInputStream.read	10 s (0 %)	1.689.239
org.iso.mpeg.mpeg21.dia.bsdl.io.BufferedPosInputStream.readAndReset	10 s (0 %)	844.909
org.iso.mpeg.mpeg21.dia.bsdl.utils.ByteArray.size	10 s (0 %)	3.380.018
java.io.BufferedInputStream.read	9.317 ms (0 %)	1.689.239
org.iso.mpeg.mpeg21.dia.bsdl.io.InputBitstream.mark	5.197 ms (0 %)	844.909
org.iso.mpeg.mpeg21.dia.bsdl.io.BufferedPosInputStream.reset	5.192 ms (0 %)	844.909
org.iso.mpeg.mpeg21.dia.bsdl.io.InputBitstream.reset	5.180 ms (0 %)	844.909
org.iso.mpeg.mpeg21.dia.bsdl.io.BufferedPosInputStream.mark	5.142 ms (0 %)	844.909
java.lang.Class.getField	5.018 ms (0 %)	75.680
java.io.BufferedInputStream.reset	3.009 ms (0 %)	844.909
java.io.BufferedInputStream.mark	2.914 ms (0 %)	844.909
java.lang.Object.<init>	2.793 ms (0 %)	876.383
java.io.PrintStream.println	2.302 ms (0 %)	7.797

Figure 2. Profiling information of the BintoBSD tool as implemented by the reference software

is generated by JProfiler[1] 3.3.1 and as one can see in this figure, almost all time is spent to the evaluation of the XPath expressions. This means that these expressions have to be evaluated faster to obtain acceptable execution times. This paper provides a solution for this problem, in particular how the XPath expressions can be evaluated very fast in the context of the BSDL framework.

[1] Java Profiler, JProfiler, can be found on http://www.jprofiler.com/

2.3 Applicability of BSDL

In the literature, one can find several BSDL schemata for bitstreams generated from different codecs. In [6], a BSDL schema is described for the experimental scalable MC-EZBC codec and in [4] the same is done for the new standardized H.264/MPEG-4 Advanced Video Coding specification (commonly referred as H.264/AVC). In these papers, one can find that the generation time of a BSD is unacceptably high when the reference software is used as implementation of the BintoBSD tool. Together with the fact that the speed decreases when the sequence is longer, we should conclude that, at the moment, BSDL is unusable under these circumstances.
In order to be able to apply BSDL in future multimedia frameworks, we have formulated a number of requirements, hereby targeting two types of applications. The first area contains applications whereby the bitstream is saved for a long time (typical applications are streaming scenarios of stored content such as Video on Demand (VoD)). In these applications, the encoding times of the bitstream and generation times of the corresponding descriptions are less important. There are no hard real time constraints as long as the times are acceptable and that the times are having a constant speed over the complete sequence (independent of the length of the sequence). A second field of application contains hard real time requisites, in particular, live content creation and streaming. Examples of applications that belong to this field are video conferences and live TV broadcasting. To make it possible to do an adaptation in this application field by using the BSDL framework, it is necessary to encode the bitstream and to generate the description in real time.
We want that BSDL is usable in both above areas of applications. Therefore, we assume the following requirements:

- It has to be possible to generate a BSDL description in real time.
- The execution times to generate a description should be in proportion to the length of the corresponding sequence. That means that the speed, by which the description is generated, has to be constant over the complete sequence.

3 Algorithmic optimizations of the BintoBSD tool

As explained in Section 2.2, it is necessary to evaluate the occurring XPath expressions in the BSDL schema as fast as possible. An XPath expression can be found as an argument of the following BSDL specific attributes: `bs2:if`; `bs2:ifUnion`; `bs2:nOccurs` and

`bs2:length`. The first two attributes expect a boolean value after the XPath evaluation and the last two expect a number as return value of the evaluation. A more detailed explanation of these attributes can be found in [1]. To speed up the generation of a bitstream description, it is possible to implement software optimizations but only implementation optimizations are not enough to obtain the requirements as formulated in Section 2.3. Therefore, we present an extension mechanism on top of the BSDL specification in the following subsections, together with an algorithmic explanation of the modified BintoBSD parser that deals with the newly introduced extensions.

3.1 Extensions to the BSDL specification

As mentioned earlier, XPath expressions in a BSDL schema are important parts to realize conditional operators and loop constructions. Every time the parser meets an XPath expression that has to be evaluated, the current generated description has to be known by the XPath evaluator. Therefore, the already generated description of the bitstream must be kept in memory to give the evaluator the opportunity to use the current context and to evaluate the XPath expression in a correct manner. Of course, the bigger the description is, the longer the evaluation of the expression will take. Therefore, it is important to keep the (internal) description tree (for example, a DOM tree), on which the expression will be evaluated, as small as possible. A typical bitstream is composed of repetitive constructions, e.g., a video bitstream contains consecutive frames and the parser needs no information about previous frames during the parsing process of the current frame. To keep the internal description context in the memory small, we have defined four new attributes that can appear in a BSDL schema. These new attributes describe how the internal description context has to be managed. The attributes are steering the BintoBSD tool to manage the memory usage as efficient as possible and still give the parser the opportunity to evaluate the XPath expressions in a correct way.

The new BSDL attributes are:

- **startContext:** this attribute contains a *marker* as value. It indicates that the element, to which the attribute belongs, is needed to evaluate a forthcoming XPath expression. When the element is conditional (in other words when it contains a `bs2:if` or a `bs2:ifNext` attribute), the element will only be kept in memory after a positive evaluation of the condition. The marker will be used to refer to the element in a later phase of the parsing process, resulting in the removal of the referred element out of the internal representation (and so out of the system memory).

- **partContext:** this attribute has the same function as the *startContext* attribute but it does not contain a *marker* as value. The attribute can have two values, in particular "Yes" and "No"; the default value of the attribute is "No". When the value is "Yes" this means that the element, to which the attribute belongs, must be kept in the memory (again only after a positive evaluation of a conditional element). Because this attribute contains no marker, it is not possible to remove this element separately out of the memory. The element can only be swapped out the internal representation (and out of the memory) when an ancestor element is removed. This attribute will typically be used when the element is necessary in the location step of an XPath expression. The attribute is useful to keep the memory manageable without an abundant and meaningless usage of markers.

- **stopContext:** this attribute is the counterpart of the *startContext* attribute. It contains a list of markers and the goal of this attribute is to remove redundant elements (and their children) out of the memory. The markers will be used to identify the elements that have to be removed. In contrast to the previous attributes, this attribute is not conditional and it is executed every time that it appears. It is the responsibility of the memory manager to keep the context (the internal description representation) manageable; in other words, the pointer to the current active element should still valid after the removal operation.

- **redefineMarker:** the last attribute can be used to rename an existing marker. The value is composed of two markers separated by a dash character ('-'). The attribute gives the author of a BSDL schema advanced control pertaining to the management of the context. This attribute can be very useful when the context tree contains a common part. That common part can be renamed at the moment that the chosen path is determined. Just like the first two attributes, this attribute is only interpreted when a conditional element evaluates positive.

3.2 Development of an example

To give a better idea about the functionality of the attributes introduced in Section 3.1, we will give an example of a BSDL schema that contains the new attributes. In Figure 3, a simplified BSDL schema is given. In this fragment, only the BSDL elements are given that are relevant for explaining our extension mechanism. Note, this schema is not a valid BSDL schema, therefore it is necessary to include other BSDL or XML schema elements such as `xsd:sequence`, `xsd:complexType`,... In this example, we have represented the global struc-

```
<element name="bitstream" bs0:startContext="bitstream">
   <element name="header" bs0:partContext="Yes">
      <element name="height"/>
      <element name="width"/>
      <element name="color" bs0:partContext="Yes"/>
   </element>
   <element name="frame" bs2:nOccurs="Unbounded"
        bs0:startContext="frame" bs0:stopContext="
        oldFrame">
      <element name ="type" bs0:startContext="type"
           bs0:redefineMarker="frame - oldFrame"/>
      <element name="ifTypeIs1" bs2:if="./type = 1"
           bs0:stopContext="type"/>
      <element name="ifColorIsGrayscale" bs0:if="/
           bitstream/header/color = 2"/>
      <element name="stream" bs0:startContext="stream">
         <element name="length" bs0:partContext="Yes"/>
         <element name="payload" bs0:stopContext="stream
              "/>
      </element>
   </element>
</element>
```

Figure 3. Simplified BSDL-schema example

ture of a video codec containing a header with some global information about the video sequence followed by a number of frame structures, indicated by the `bs2:nOccurs="Unbounded"` attribute. Each frame is composed of the same syntax elements as one can see. The conditional elements are indicted by the BSDL `bs2:if` attribute. The *color* element of the *header* is kept in memory during the complete parsing process, as one can see in the schema. It is indicated by the `bs0:partContext` attribute. The *frame* elements are removed every time a *frame* structure is parsed. A *frame* element is stored in memory by interpreting the `bs0:startContext` attribute and the marker is used as reference. At the same parsing time, the previous parsed frame is removed (indicated by the bs0:stopContext attribute). The bs0:redefineMarker attribute controls the *frame* that has to be removed next time.
In Figure 4, the internal representation of the description is given when the context-related attributes of the example in Figure 3 are eliminated. That is an equivalent representation as the description that the BintoBSD tool generates and it is the internal representation that the reference software uses. As one can see, the longer the video sequence, the bigger the internal representation and the slower the evaluation of the XPath expressions will be. In this figure, the conditional elements are mentioned by a dashed bounding box.
In Figure 5 (a), the internal representation of the context is given after every complete parsing process of a frame by using the context-related attributes of the BSDL schema. As one can see, almost nothing (only the header information) is kept in the memory. In Figure 5 (b), the internal context is given just before the *payload* element is parsed. As one can see, also during the parsing process of a frame, the internal context is smaller than in Figure 4, which has to speed up the evaluation process of the XPath expressions.

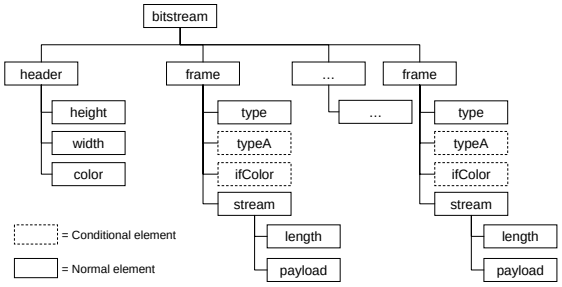

Figure 4. Internal representation of the description without using context-related attributes

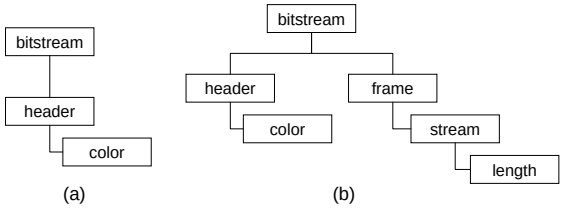

Figure 5. Internal representation of the description by using context attributes: (a) internal description after each frame is parsed; (b) internal description before the *payload* element is parsed

3.3 Algorithm of the modified BintoBSD parser

In Section 3.1, we have introduced the new developed BSDL attributes and in Section 3.2, the impact of these new attributes is explained by using an example on how the internal context evolves. In this subsection, we explain how the BintoBSD tool has to be modified to interpret the new attributes and to manage the memory as efficient as possible.
The algorithm is given in pseudo-code in Figure 6. In this code, we describe only how the memory manager must deal with the new attributes. In the code, the data structure `internal_tree` contains the context on which the XPath expressions have to be evaluated; `lookup_table` is a map, which maps a key (marker) to a value (in particular, a context element) and `deleted_table` is a vector with several references to context elements. The `current_pointer` refers to the active element in the context tree. It is the place from where the evaluations of the XPath expressions must start. As one can see in Figure 6, an element can only be removed from the context tree under the condition that the `current_pointer` is not an element of the removing part of context.

```
for each element in the BSDL schema do
   if (element is a beginElement) do
      if(!(element is conditional) OR ((element is
           conditional) AND (evaluation is true)))) do
         if (element is startContext) do
            add element to internal_tree at
                 current_pointer
            current_pointer = newElement
            add marker and reference to lookup_table
         end if
         if (element is partContext) do
            add element to internal_tree at
                 current_pointer
            current_pointer = newElement
         end if
         if (element is redefineMarker) do
            old_reference = find marker in lookup_table
            remove marker and reference from
                 lookup_table
            add new marker and old_reference to
                 lookup_table
         end if
      end if
      if (element is stopContext) do
         reference = find marker in lookup_table
         remove marker and reference from lookup_table
         if (current_pointer is the same as or child
              from reference) do
            keep reference in internal_tree
            add reference to deleted_table
         else
            remove reference from internal_tree
         end if
      end if
   end if
   else if ((element is an endElement) AND ((element is
        startContext) OR (element is partContext)) do
      if (current_pointer element of deleted_table) do
         temp = parent of current_pointer
         remove current_pointer from internal_tree
         remove current_pointer from deleted_table
         current_pointer = temp
      else
         current_pointer = parent of current_pointer
      end if
   end if
end for
```

Figure 6. Pseudo-code of the modifications of the memory manager of the BintoBSD tool

4 Performance analysis of the modified parser

In this section, we discuss the impact of our new introduced attributes in terms of execution times.

4.1 Materials and methods

To test the performance of our new algorithm, we have extended two different BSDL schemata with our new context-related attributes. The first BSDL schema is a schema for the MC-EZBC codec, the structure of the schema is described in [6]. The codec is an experimental embedded scalable wavelet-based codec. The second schema is developed for the H.264/AVC codec, the structure of this schema is explained in [4]. To describe an encoded bitstream in XML by using the BSDL technology, we have used 4 different bitstreams encoded by the MC-EZBC codec (version of September 2003) and 11 H.264/AVC bitstreams created by relying on the JM 9.4 reference software. The properties of the bitstreams used are given in Table 1. In this table, the first 4 bitstreams are the MC-EZBC bitstreams whereby the number of frames is important. The others 11 bitstreams are the H.264/AVC streams and hereby, the number of slices is more important. This is because in the H.264/AVC specification, the slices are the fundamental building blocks of a bitstream and not the frames.

Table 1. Characteristics of used bitstreams

Name	#frames	#slices / frame	Total #slices	Bitstream Size (bytes)
ezbc_040	40	n/a	n/a	2425432
ezbc_121	121	n/a	n/a	6474100
ezbc_300	300	n/a	n/a	14357852
ezbc_541	541	n/a	n/a	33349445
avc_21_1	21	1	21	89644
avc_21_2	21	2	42	89285
avc_49_1	49	1	49	205217
avc_21_3	21	3	63	89721
avc_199_1	199	1	199	833478
avc_199_2	199	2	398	8333882
avc_399_1	399	1	399	1676736
avc_199_3	199	3	597	834110
avc_599_1	599	1	599	2531328
avc_599_2	599	2	1198	2530329
avc_599_3	599	3	1797	2530286

To execute our tests, we have used version 1.2.1 of the MPEG-21 BSDL reference software. Our algorithm, as discussed in Section 3.3, was integrated into the BintoBSD tool of the reference software. We have compared our implementation of the BintoBSD tool with version 1.2.1 of the reference software and for each generation of a description of a bitstream, we have measured the execution time 5 times. The tables in the following subsection contain the average over the 5 runs.

All the simulations were run on a PC having an Intel Pentium IV CPU, clocked at 2.8GHz with Hyper-Threading and having 1GB of RAM at its disposal. The operating system used was Windows XP Pro (SP2) and Sun Microsystem's Java 2 Runtime Environment (Standard Edition version 1.5.0_02-b09) was running as JVM.

4.2 Results and discussion

First, we discuss the results for the bitstreams as encoded by the MC-EZBC codec. The execution times, as measured by the reference software and our modified BintoBSD tool, together with speed and ratio are given in Table 2. The speed is expressed in `frames/s` and the ratio is calculated

as follows:

$$ratio = 1 - \frac{time\ modified\ BintoBSD\ tool}{time\ reference\ software} \quad (1)$$

In this table, one can see that our implementation is up to 98% faster than the reference software and that it is possible to generate the descriptions in real time. The latter was not possible with an implementation without our modifications. The ratio also gives us an idea about the profit that we obtain with the introduction of the new BSDL attributes. The most interesting conclusion that can be drawn from this table is given in Figure 7. In particular, the speed by which the description is generated is constant, or in other words, the execution time is linear in function of the number of frames in the sequence (see Figure 7). Note that the speed is increasing with the length of the sequence, the reason for this phenomenon is the Just-In-Time (JIT) compiler of the JVM (the longer the sequence will be, the less the influence of the JIT compiler will be and the more constant the speed will be).

Table 2. Results of our simulation for the MC-EZBC sequences

Name	Reference Software		Modified BintoBSD Tool		Ratio
	time (s)	speed	time (s)	speed	(%)
ezbc_040	2.48	16.103	0.71	56.2	72
ezbc_121	11.26	10.740	1.04	116.9	91
ezbc_300	48.59	6.174	1.51	198.1	97
ezbc_541	108.97	4.954	1.98	272.9	98

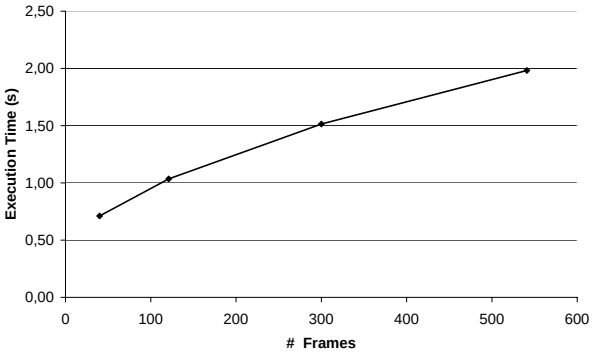

Figure 7. Execution times of the BintoBSD tool for the MC-EZBC bitstreams in function of the number of frames

The same analysis as for the MC-EZBC bitstreams is done for the H.264/AVC encoded streams. The results for this configuration are given in Table 3. The ratio is again calculated as in (1) and the speed is expressed in `slices/s` (because the fundamental elements of an H.264/AVC bitstream are the slices and not the frames, as already mentioned). In this table, one can see that the reference software was unacceptable slow and that the introduction of our new elements has run the execution times drastically down. For the longest sequences, we obtain a ratio of more than 99%. Only the descriptions for the very short sequences (the sequences that contain only 20 frames, which are sequences that are not appear in practical situations) cannot be generated in real time. The reason for the low speed in these cases is the fact that every run has a constant startup time. This startup time involves that the BSDL schema will be loaded and the internal class structure will be generated together with the influence of the JIT-compiler. In case of the H.264/AVC schema, the average loading time of the schema is $0.472s$. When bitstreams are longer (in particular, when the sequence contains more slices), the influence of this constant startup time will be decreased, as can be seen in the table. We observe also that the speed of the reference software decrease as the number of slices increase (that means that there is no linear connection between the execution time and the number of parsed slices) and that the execution time is far from real time.

Table 3. Results of our simulation for the H.264/AVC sequences

Name	Reference Software		Modified BintoBSD Tool		Ratio
	time (s)	speed	time (s)	speed	(%)
avc_21_1	8.09	2.595	1.32	15.92	84.0
avc_21_2	18.70	2.246	1.60	26.30	91.5
avc_49_1	22.55	2.172	1.80	27.22	92.0
avc_21_3	34.00	1.853	1.82	34.58	94.6
avc_199_1	281.55	0.707	3.88	51.26	98.6
avc_199_2	1300.00	0.306	5.70	69.79	99.5
avc_399_1	1164.59	0.342	6.57	60.74	99.4
avc_199_3	3300.00	0.181	7.40	80.68	99.8
avc_599_1	3400.00	0.176	9.18	65.28	99.7
avc_599_2	21000.00	0.057	14.45	82.92	99.9
avc_599_3	50000.00	0.036	19.47	92.29	99.9

To visualize the performance of our modified BintoBSD tool, we have again plotted the execution times of the tool in function of the number of slices in the sequence. This visualization is given in Figure 8. One can see that the curve has a linear course, certainly for the three longest sequences (in particular, the sequences that contain 599 frames with 1, 2 or 3 slices a frame respectively). This linear connection indicates an almost constant speed, as also can be seen in Table 3. The jumps in the graph are a remarkable phenomenon in Figure 8. The two jumps have the same origin and we will explain the cause based on the last jump. The two measurements of sequences that are responsible for the second jump are `avc_599_1` and `avc_199_3`. These sequences contains almost as much slices, in particular 599

and 597 slices respectively. Therefore, we should expect that the execution times are approximately the same but this is not true as one see in Table 3 and Figure 8. Notwithstanding the fact that the XML descriptions of the bitstreams (generated by BintoBSD) are having about the same sizes (1232 Kbytes and 1235 Kbytes, which is not represented in the tables), which is expectable because the slices are the fundamental elements of a bitstreams, the times are not the same. The extra time needed to generate the description for the `avc_599_1` is spending on I/O operations. In Table 1, one can see that the `avc_599_1` contains three times more bytes than `avc_199_3` and the BintoBSD tool must, of course, parse the complete bitstream which generates the extra I/O time for the `avc_599_1` sequence, which results in a the higher execution time as can be seen in Table 3.

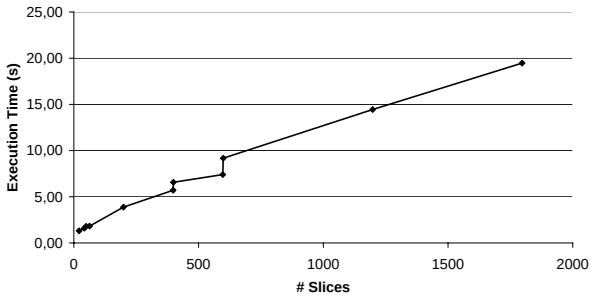

Figure 8. Execution times of the BintoBSD tool for the H.264/AVC bitstreams in function of the number of slices

Finally, we want to verify if our postulated requirements of Section 2.3 are satisfied. For the MC-EZBC, it is clear that the generation speed of a description is constant over the complete sequence and that it is no problem to generate a description in real time (up to 250 frames / s). The same conclusion can be made for the complex H.264/AVC bitstreams. In this situation, the speed is also constant and it is possible to parse a bitstream in real time (up to 3 slices a frame for sequences that are long enough, this means in situations whereby the startup time is almost completely eliminated). Therefore, we can conclude that the requirements are satisfied and we note that the execution times should go further down after a complete software optimization.

5 Conclusions

In this paper, the MPEG-21 DIA BSDL standard was discussed. This specification provides a framework for adapting multimedia content in a transparent and generic way, hereby making use of XML-based descriptions of the high-level structure of (scalable) bitstreams. We have postulated a number of requirements to which the description generation process has to comply. After having profiled the current version of the MPEG-21 BSDL reference software, we could conclude that the requirements in question were not met and that BSDL could not be used in for example streaming use cases. Therefore, we have introduced a number of extensions to the BSDL standard, giving implementations of the standard the possibility to evaluate XPath expressions in a very efficient and fast manner. The algorithmic modifications of the parser were discussed and implemented in version 1.2.1 of the reference software. After having tested our new algorithm, we could observe that up to 99% less time was needed to produce the same bitstream description. This paper also shows for the first time that it is possible to generate an XML description of an H.264/AVC bitstream in real time by using BSDL technology.

6 Acknowledgements

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] Information technology – Multimedia framework – Part 7: Digital Item Adaptation. ISO/IEC JTC 1, October 2004.

[2] M. Amielh and S. Devillers. Bitstream Syntax Description Language: Aplication of XML-Schema to Multimedia Content Adaptation. *WWW2002: The Eleventh International World Wide Web Conference (Honolulu, Hawaii)*, May 2002.

[3] P. Cimprich. Streaming Transformations for XML (STX) Version 1.0 Working draft. *http://stx.sourceforge.net/documents/spec-stx-20040701.html*, July 2004.

[4] W. De Neve, S. Lerouge, P. Lambert, and R. Van de Walle. A performance evaluation of MPEG-21 BSDL in the context of H.264/AVC. *Proceedings of SPIE annual meeting 2004: Signal and Image Processing and Sensors*, August 2004.

[5] D. De Schrijver. High Level XML Description of a Scalable Video Codec. *Abstracts of the Fifth FTW PhD Symposium*, December 2004.

[6] D. De Schrijver, W. Van Lancker, and R. Van de Walle. Performance Of a Scalable Bitstream Adaptation Process Based On High Level XML Descriptions. *Proceedings of the 6th WIAMIS Conference*, April 2005.

[7] M. Kay. *XSLT Programmer's Reference, 2nd Edition*. Wrox Press Ltd., Birmingham, UK, 2001.

[8] A. Vetro, C. Christopoulos, and T. Ebrahimi. Universal Multimedia Access. *IEEE Signal Processing magazine*, March 2003.