BFlavor: an Optimized XML-based Framework for Multimedia Content Customization

Davy Van Deursen*, Wesley De Neve*, Davy De Schrijver*, and Rik Van de Walle+

Department of Electronics and Information Systems - Multimedia Lab
*Ghent University - IBBT
+Ghent University - IBBT - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
e-mail: davy.vandeursen@ugent.be

Abstract. During recent years, several languages have been developed that allow to automatically create descriptions containing information about the high-level structure of binary multimedia resources. Such an approach makes it possible to tackle the diversity of the current networks, terminals, and multimedia formats in a transparent way. This paper discusses the fundamentals of BFlavor, a novel language for describing the structure of binary multimedia resources. BFlavor has been developed to combine the strengths of MPEG-21 BSDL and XFlavor and to avoid their weaknesses. The three different scalability axes of the H.263+ coding format (i.e., temporal, spatial, and Signal-to-Noise Ratio scalability) are exploited by making use of MPEG-21 BSDL, XFlavor, and BFlavor. We show through experiments that BFlavor maintains the constant memory consumption of XFlavor, generates compact descriptions, and outperforms XFlavor and MPEG-21 BSDL in terms of execution times.

Index Terms -BFlavor, content adaptation, H.263+, MPEG-21 BSDL, XFlavor

1 Introduction

Today, in the world of multimedia, there is a huge heterogeneity in terminals, networks, and multimedia formats used. These differences have to be taken into account when multimedia resources have to be delivered to different devices. Scalable coding makes it possible to tackle this tremendous diversity. It enables the extraction of multiple (lower quality) versions of the same multimedia resource without the need of a complete encode-decode process. This is in line with the vision of the Universal Multimedia Access (UMA) philosophy: a multimedia resource only needs to be created once, after which it can be published to all possible terminals using all possible different networks. It is important to realise that an efficient solution for the heterogeneity does not only imply the usage of scalable coding, but also the usage of a complementary content adaptation system. One could build a content adaptation system operating directly onto the bitstream. However, such an implementation will be error prone and the adaptation system will only be capable to support one bitstream format. Therefore, a generic approach is needed for the adaptation of (scalable) bitstreams. Such an architecture has to support multiple multimedia formats and it should be possible to extend it with new multimedia formats. These adaptive multimedia systems can be realised by relying on textual descriptions of the high-level structure of scalable bitstreams (usually in XML format). In this paper, a new language is introduced for the creation of these textual descriptions.

The outline of this paper is as follows. In Section 2, the concept of bitstream structure description languages is elaborated. Two existing languages, in particular MPEG-21 BSDL and XFlavor are discussed. Section 3 introduces BFlavor (BSDL + XFlavor), our new bitstream structure description language. In Section 4, this new approach is validated by presenting some experimental results, hereby targeting the H.263+ coding format. Finally, conclusions are made in Section 5.

2 Bitstream Structure Description Languages

2.1 Overall Approach

A possible architecture for the customization of (scalable) bitstreams is shown in Fig. 1. Such an adaptation system can be realized by relying on Bitstream Syntax Descriptions (BSDs). A bitstream structure description language makes it possible to describe the structure of a specific bitstream format. A text-based BSD is generated by a `bitstream-to-BSD` parser which interprets the structure of the bitstream. The BSD typically contains information about the high-level structure

(information about packets, headers, or layers of data) of a (scalable) bitstream. Such a BSD can be seen as an intermediate format that acts as an abstraction layer for the bitstream. The most elegant manner to store the text-based BSD is by using the ubiquitous XML specification. The XML formalism does not only allow repurposing many existing tools for manipulating XML-based BSDs, it also allows a straightforward integration with other metadata standards, such as the MPEG-7 specification. According to a given set of constraints, the BSD is transformed, resulting in a customized BSD. Because of the high-level nature of the BSD, only a limited knowledge about the structure of a bitstream is required to perform this transformation. Finally, the customized BSD is used by a `BSD-to-bitstream` parser to generate an adapted bitstream. The major advantage of a BSD-driven approach for the customization of (scalable) bitstreams is its generic architecture. To support a new multimedia format, only a description of the structure of a bitstream format in the specific language has to be created while the same generic parsers can still be used.

2.2 MPEG-21 BSDL

The MPEG-21 Bitstream Syntax Description Language (MPEG-21 BSDL) is a tool of part 7 (Digital Item Adaptation, DIA) of the MPEG-21 specification. It is built on top of the World Wide Web Consortium's (W3C) XML Schema Language and is able to describe the structure of a (scalable) bitstream in XML format [1]. The primary motivation behind the development of MPEG-21 BSDL is to assist in customizing scalable bitstreams [2]. The Bitstream Syntax Schema (BS Schema), which contains the structure of a certain media format, is used by the BintoBSD Parser to generate a BSD for a given (scalable) bitstream. After the BSD is transformed, an adapted bitstream is created by using the BSDtoBin Parser, which takes as input the BS Schema, the customized BSD, and optionally the original bitstream.

2.3 XFlavor

The Formal Language for Audio-Visual Object Representation, extended with XML features (XFlavor) is a declarative C++-like language to describe the syntax of a bitstream on a bit-per-bit basis. XFlavor was initially designed to simplify and speed up the development of software that processes audio-visual bitstreams by automatically generating a parser for these bitstreams. By extending this automatically generated parser with XML features, it was possible to translate the syntax of a bitstream in XML format [3]. The XFlavor code, which contains a description of the syntax of a certain media format, is translated by the Flavorc translator to Java or C++ classes. This set of classes is compiled to a coding format-specific parser. XFlavor comes with the Bitgen tool for creating an adapted bitstream, hereby guided by the customized BSD. Note that the Flavorc translator is also able to generate an XML Schema which can be used to validate the BSD that is created by the automatically generated parser.

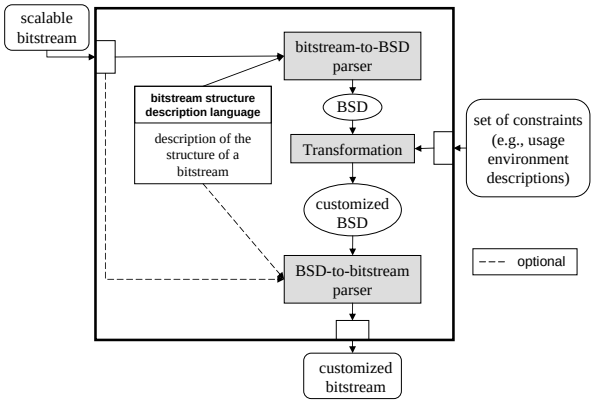

Fig. 1. A BSD-driven content adaptation framework

3 BFlavor: Harmonizing MPEG-21 BSDL and XFlavor

Although MPEG-21 BSDL and XFlavor can be used as stand-alone tools [4], a harmonized approach can combine the strengths of the two technologies. In XFlavor, the XML schema is only used for validation purposes. The bitstream generator (i.e., Bitgen) only uses information from the BSD and thus is independent of the XFlavor code. Hence, the complete bitstream data are actually embedded in the BSD, resulting in potentially huge descriptions. On the contrary, MPEG-21 BSDL makes use of a specific datatype to point to a data range in the original bitstream when it is too verbose to be included in the description (i.e., by making use of the language construct `bs1:byteRange`). This results in BSDs containing only the high-level structure of the bitstream. The strengths of XFlavor are the fast execution speed

```
class Picture{
   Base_layer b_layer;
   Enhancement_layer e_layer;
}

class Bitstream{
   while(1)
      Picture picture;
}
```

Fig. 2. BFlavor code example

```
<!-- before transformation -->
<Bitstream>
   <Picture>
      <Base_layer><!-- ... --></Base_layer>
      <Enhancement_layer><!-- ... --></Enhancement_layer>
   </Picture>
   <Picture>
      <Base_layer><!-- ... --></Base_layer>
      <Enhancement_layer><!-- ... --></Enhancement_layer>
   </Picture>
</Bitstream>

<!--after transformation -->
<Bitstream>
   <Picture>
      <Base_layer><!-- ... --></Base_layer>
   </Picture>
   <Picture>
      <Base_layer><!-- ... --></Base_layer>
   </Picture>
</Bitstream>
```

Fig. 3. An example of a BSD before and after a transformation

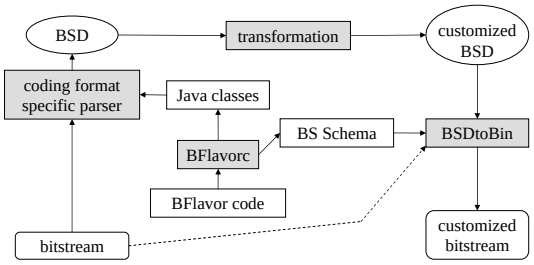

Fig. 4. The BFlavor adaptation chain

and the low and constant memory consumption of the coding format-specific parser, while the BintoBSD Parser of MPEG-21 BSDL struggles with an unacceptable execution speed and increasing memory consumption (see Section 4) caused by an inefficient XPath evaluation process. This is due to the fact that the entire description of the bitstream structure is kept in system memory in order to allow the evaluation of arbitrary XPath 1.0 expressions.

BFlavor, our novel bitstream structure description language, bridges the gap between MPEG-21 BSDL and XFlavor. It is developed to combine the strengths of MPEG-21 BSDL and XFlavor, i.e., to generate a compact high-level BSD at a fast execution speed and with a constant memory consumption. It is built on top of XFlavor by defining a number of restrictions and extensions (such as MPEG-21 BSDL is build on top of W3C XML Schema). By using the automatically generated parser of BFlavor, it is possible to generate BSDs that can be further processed by the BSDtoBin Parser of MPEG-21 BSDL. Note that one could also modify the MPEG-21 BSDL language specification in order to achieve a feasible execution speed and a constant memory consumption [5]. The performance of this approach, together with the performance of MPEG-21 BSDL and XFlavor, will be compared to BFlavor's performance in Section 4.

3.1 Working of BFlavor

The BFlavor adaptation chain is illustrated in Fig. 4. The BFlavor code describes the structure of a specific bitstream format. In Fig. 2, an example is given of a BFlavor code fragment. It describes `Picture`s that contain a `Base_layer` and an `Enhancement_layer`. The BFlavorc translator uses this code to generate Java source classes that can be compiled to a coding format-specific parser. So far, the XFlavor approach has been followed. From this point, a switch is made to the MPEG-21 BSDL approach. The coding format-specific parser generates a BSD which can be further processed by the efficient and format-agnostic BSDtoBin Parser of MPEG-21 BSDL. An example of a such a BSD is illustrated in Fig. 3. The BSDtoBin Parser needs a BS Schema. This schema contains the structure of a specific bitstream format, analogous to the BFlavor code. Therefore, the BFlavorc translator is also able to generate a BS Schema from the BFlavor code. Thus, after the transformation of the BSD, the BSDtoBin Parser uses the customized BSD, the generated BS Schema, and the original bitstream to generate a customized bitstream. As a result of our approach, it is possible to generate BSDs with the fast execution speed and the low memory consumption of the coding format-specific parser of XFlavor, while the generated BSDs are compact because they only contain a description of the high-level structure of the (scalable) bitstream.

3.2 Implementation details

In Section 3.1, the general working of BFlavor was illustrated. In order to realize the construction of BFlavor (i.e., the generation of a parser producing BSDs usable by the BSDtoBin Parser of MPEG-21 BSDL), a few restrictions and extensions have to be defined on top of XFlavor. This section will

discuss these restrictions and extensions, as well as the methodology needed to implement them.

Restrictions and extensions on top of XFlavor Section 3.1 has shown that the Flavorc translator has to be able to generate Java source classes and a BS Schema using the BFlavor code. This is only possible when a number of restrictions are defined for this BFlavor code. Because the BFlavor code is a C++-like code, it contains variables. These variables can be parsable (i.e., they retrieve their value from the bitstream) or non-parsable (i.e., regular C++ variables). The parsable variables are similar to the `xsd:element` language construct in MPEG-21 BSDL. Non-parsable variables cannot be translated to an MPEG-21 BSDL language construct which implies a prohibition on the use of non-parsable variables in the BFlavor code. Other language constructions in XFlavor which cannot be translated to MPEG-21 language constructs are the `map` and `bac` constructions as well as multi-dimensional arrays. Hence, only one-dimensional arrays are allowed in BFlavor as they can be translated to elements with a `bs2:nOccurs` attribute. A last restriction is the use of the built-in operators of XFlavor. Only the `nextbits()` operator can be translated to an MPEG-21 BSDL element (i.e., an element with the `bs2:ifNext` attribute). This implies that the other built-in operators (i.e., `nextcode()`, `numbits()`, `lengthof()`, `isidof()`, and `skipbits()`) are prohibited in BFlavor.

XFlavor also needs some extensions to be able to generate Java source classes and a BS Schema. A first extension is the possibility to include information about the root element, namespace, and target namespace in the BFlavor code. A second extension is related to datatypes. In MPEG-21 BSDL, one can have elements and types, while XFlavor only has classes. As such, it must be possible to signal in the XFlavor code that a class represents a type or an element. There are three options to signal an XFlavor class as a type: the class definition has to be translated to a `complexType` element; the class definition has to be translated to a `simpleType` element, hereby providing information about what procedural object to use (i.e., by making use of the `bs0:implementation` attribute in MPEG-21 BSDL); or the class definition has to be translated to a `simpleType` element, hereby providing information about the length of a start or end code. A disadvantage of XFlavor was the lack of possibility

```
%targetns{H_263%targetns}
%ns{h263%ns}
%root{Byte_stream%root}

%align{%align}
class Stuffing{}

%payload{0000FC-0000FF%payload}
%payload{000080-000083%payload}
class GOB_Payload{}

class Picture{
   bit(22) picture_start_code = 0b00000000000000000100000;
   bit(8) temporal_reference;
   Ptype ptype;
   if(ptype.source_format == 7)
      Plus_Header plus_header;
   bit(5) pquant;
   if(ptype.source_format != 7)
      bit(1) cpm;
   if(ptype.source_format != 7 && cpm == 1)
      bit(2) psbi;
   if(ptype.optional_pbframes_mode == 1){
      bit(3) temporal_reference_b;
      bit(2) dbquant;
   }
   do{
      bit(1) pei;
      if(pei == 1) bit(8) psupp;
   }while(pei == 1);
   Stuffing estuff;
   GOB_Payload payload;
}

class Bitstream{
   while(1)
      Picture picture;
}
```

Fig. 5. Partial BFlavor code for the H.263+ coding format

```
<Bitstream>
   <!-- ... -->
   <Picture>
      <picture_start_code>32</picture_start_code>
      <temporal_reference>0</temporal_reference>
      <Ptype><!-- ... --></Ptype>
      <Plus_Header><!-- ... --></Plus_Header>
      <pquant>13</pquant>
      <do_pei><pei>0</pei></do_pei>
      <estuff>15</estuff>
      <payload>10 3899</payload>
   </Picture>
   <!-- ... -->
</Bitstream>
```

Fig. 6. Example of BSD for a H.263+ bitstream

to refer to the original bitstream. BFlavor solves this problem by defining a payload datatype that can be translated to a `simpleType` element with restriction to the `bs1:byteRange` datatype of MPEG-21 BSDL. The last extension is a datatype which is able to make the bitstream byte-aligned. This is mapped to the non-normative `bs0:fillByte` construct in MPEG-21 BSDL.

Methodology The extensions are implemented by using the verbatim codes of XFlavor. These codes are enclosed within the verbatim delimiters %`x`{ and %`x`} in the XFlavor code. The symbol `x` hereby denotes the type of the verbatim code. In Fig. 5, a partial BFlavor fragment for the H.263+ coding format is illustrated. Specific verbatim codes are used to signal the root element, namespace,

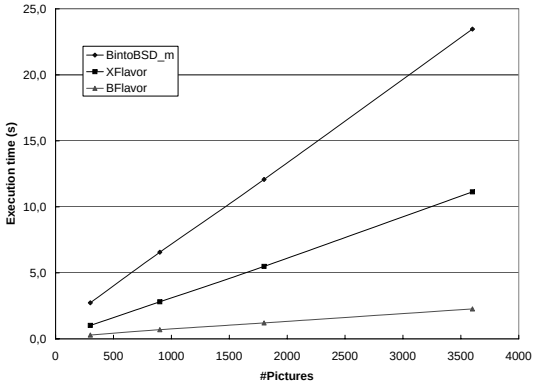

Fig. 7. Execution speed (SNR)

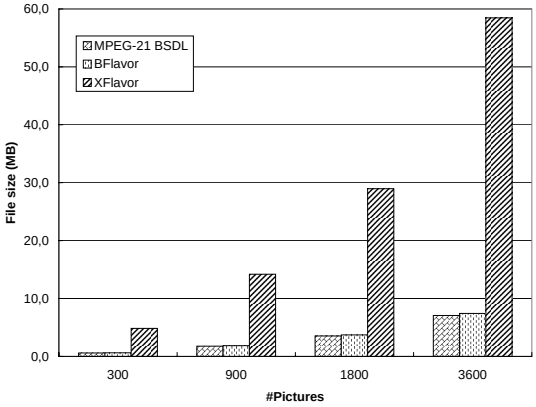

Fig. 8. BSD file sizes (Temporal)

and target namespace. Classes that represent a type will be preceded by the specific verbatim code associated with that type. In the example, the `Stuffing` class represents an alignment type, while the `GOB_Payload` class is a payload type. The value of the verbatim codes is used to pass some additional information about the datatype. For example, the payload datatypes contain some start codes as additional information as illustrated with the `GOB_Payload` class.

The modifications to create BFlavor on top of XFlavor have to be made inside the Flavorc translator. The working of this translator is as follows. The XFlavor code is parsed by making use of a parser generated by Lex/Yacc. Based on the internal representation, generated by the Lex/Yacc software, an XML Schema or a set of Java or C++ classes is generated. To implement the restrictions and extensions in the Flavorc translator, modifications have to be made in the Lex and Yacc description, as well as in the core of the Flavorc translator. In the Lex and Yacc description, the new verbatim codes have to be inserted in order to be recognized by the generated Lex/Yacc-parser. The core of the Flavorc translator has to be modified in two manners. Firstly, the generated XML Schema represents a BS Schema. Secondly, the generated Java or C++ classes are able to generate an XML document which can be further processed by MPEG-21's BSDtoBin Parser.

4 Experimental Results

To validate our harmonized approach, a comparison of MPEG-21 BSDL, XFlavor, and BFlavor is provided in terms of execution times, memory consumption, and file sizes. The languages are used

Table 1. Bitstream characteristics

	Base Layer			Enhancement Layer		
	#Pic.	Res.	QP	#Pic.	Res.	QP
Spat.	300	QCIF	13	300	CIF	13
Temp.	300	QCIF	13	600	QCIF	13
SNR	300	QCIF	25	300	QCIF	5

to exploit the scalability properties of H.263 Version 2 bitstreams. H.263 Version 2, also known as H.263+, supports both temporal, spatial, and Signal-to-Noise Ratio (SNR) scalability. The Foreman sequence with QCIF (176x144) resolution was used as test sequence. Table 1 shows the bitstream characteristics for the encoded test sequence with 300 pictures. The characteristics for the sequences with 900, 1800, and 3600 pictures are similar. The technologies generate a BSD of the bitstream which contains the base layer and the enhancement layer. During the transformation, the enhancement layer is dropped by using eXtensible Stylesheet Language Transformations (XSLTs). Finally, a bitstream is generated from the customized BSD which contains only the base layer. All tests were done five times, after which an average was taken.

The experiments were done on a PC having an Intel Pentium D 2.8 GHz CPU and 1 GB of system memory at its disposal. The operating system used was Windows XP Pro SP2, running Sun Microsystems's Java 2 Runtime Environment (Standard Edition version 1.5). SAXON 8 was used for applying XSLTs to BSDs. The memory consumption was registered by relying on the JProfiler 4.0 software package. The TMN-3.0 H.263 codec was used to encode/decode the test sequences.

The results of the BSD generation speed are shown in Table 2. Note that BintoBSD$_r$ stands for the original BintoBSD Parser (v1.2.1), while

Table 2. BSD generation speeds

	#Pic.	BintoBSD$_r$ (s)	BintoBSD$_m$ (s)	BFl. (s)	XFl. (s)
Spat.	300	278.6	2.7	0.3	1.0
	900	2332.2	6.6	0.7	3.1
	1800	9127.7	12.4	1.3	5.8
	3600	N/A	18.4	1.9	9.2
Temp.	300	232.6	2.4	0.2	0.6
	900	2013.1	6.4	0.6	1.6
	1800	8712.6	11.2	1.0	3.2
	3600	N/A	19.8	1.5	5.8
SNR	300	270.8	2.7	0.3	1.0
	900	2288.5	6.6	0.7	2.8
	1800	8902.2	12.1	1.2	5.5
	3600	N/A	23.5	2.3	11.1

Table 3. Adaptation and bitstream generation times

		XSLT (s)			Bitstream Generation (s)		
	#Pic.	BSDL	BFl.	XFl.	BSDL	BFl.	XFl.
Spat.	300	0.6	0.6	2.6	0.6	0.6	1.6
	900	1.1	1.1	5.1	0.7	0.7	4.7
	1800	1.6	1.6	9.7	1.0	0.9	9.3
	3600	2.3	2.3	14.7	1.2	1.2	14.2
Temp.	300	0.5	0.5	2.2	0.6	0.6	1.5
	900	0.9	0.8	3.1	0.7	0.7	6.1
	1800	1.4	1.3	8.1	0.9	0.9	12.1
	3600	2.5	2.7	15.5	1.3	1.3	24.3
SNR	300	0.6	0.6	2.1	0.6	0.6	1.6
	900	1.1	1.0	4.2	0.7	0.7	5.6
	1800	1.6	1.7	7.8	1.0	0.9	11.2
	3600	2.6	2.7	16.2	1.3	1.4	16.3

BintoBSD$_m$ stands for a modified version of the BintoBSD Parser which supports the BSDL extensions for achieving a usable adaptation framework [5]. It is clear that the BintoBSD$_r$ Parser cannot be used in practice because of the quadratic increasing execution time and the increasing memory consumption (35.0 MB and 62.0 MB for a bitstream having 900 and 1800 Pictures respectively). This is due to the fact that the entire BSD is kept in memory in order to allow the evaluation of XPath expressions. The BintoBSD$_m$ Parser, the BFlavor-based parser, and the XFlavor-based parser all maintain a low and constant memory consumption (less than 2 MB). Furthermore, Fig. 7 illustrates that these parsers show a linear behavior in terms of execution speed. One can see that the BFlavor-based parser outperforms the XFlavor-based parser and the BintoBSD$_m$ Parser. Table 3 shows the transformation and bitstream generation times for the adapted BSDs. The most notable is the poor transformation speed for the BSDs generated with XFlavor. This is because all the information to create the adapted bitstream has to be included in the customized BSD, resulting in large BSDs. Fig. 8 illustrates the difference in BSD file sizes between XFlavor, MPEG-21 BSDL, and BFlavor.

5 Conclusions and Future Work

We have presented BFlavor, a new solution for XML-driven content adaptation. BFlavor is designed to combine the strengths of XFlavor and MPEG-21 BSDL and to avoid their weaknesses. Experimental results have shown that BFlavor maintains the constant memory consumption of XFlavor and outperforms XFlavor and MPEG-21 BSDL in terms of execution times. Thanks to its possibility to refer to the original bitstream, BFlavor is also able to describe the high-level structure of a bitstream, resulting in compact BSDs that can be further processed by MPEG-21's BSDtoBin Parser. Future work will focus on the exploitation of scalability of the H.264/AVC SVC coding format by relying on BFlavor. This will require further development of BFlavor.

Acknowledgements

The research activities as 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. Devillers, S., Timmerer, C., Heuer, J., Hellwagner, H.: Bitstream Syntax Description-Based Adaptation in Streaming and Constrained Environments. IEEE Trans. Multimedia 7 (3) (2005) 463–470
2. De Schrijver, D., Poppe, C., Lerouge, S., De Neve, W., Van de Walle, R.: MPEG-21 Bitstream Syntax Descriptions for scalable video codecs. (accepted for publication in Multimedia Systems Journal)
3. Hong, D., Eleftheriadis, A.: XFlavor: Bridging Bits and Objects in Media Representation. In: Proceedings, IEEE Int'l Conference on Multimedia and Expo (ICME), Lausanne, Switzerland (2002)
4. De Neve, W., Van Deursen, D., De Schrijver, D., De Wolf, K., Van de Walle, R.: Using Bitstream Structure Descriptions for the Exploitation of Multi-layered Temporal Scalability in H.264/MPEG-4 AVC's Base Specification. In: Proc. of PCM 2005, Chejudo, Korea, Springer-Verlag (2005) 641–652
5. De Schrijver, D., De Neve, W., De Wolf, K., Van de Walle, R.: Generating MPEG-21 BSDL Descriptions Using Context-Related Attributes. In: Proceedings of the 7th IEEE International Symposium on Multimedia, Irvine, USA (2005) 79–86