A Real-Time XML-based Adaptation System for Scalable Video Formats

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

Department of Electronics and Information Systems - Multimedia Lab
Ghent University - IBBT
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
davy.vandeursen@ugent.be,
http://multimedialab.elis.ugent.be

Abstract. Scalable bitstreams are used today to contribute to the Universal Multimedia Access (UMA) philosophy, i.e., accessing multimedia anywhere, at anytime, and on any device. Bitstream structure description languages provide means to adapt scalable bitstreams in order to extract a lower quality version. This paper introduces a real-time XML-based framework for content adaptation by relying on BFlavor, a combination of two existing bitstream structure description languages (i.e., the MPEG-21 Bitstream Syntax Description Language (BSDL) and the Formal Language for Audio-Visual Representation extended with XML features (XFlavor)). In order to use BFlavor with state-of-the-art media formats, we have added support for transparent retrieval of context information and support for emulation prevention bytes. These extensions are validated by building a BFlavor code for bitstreams compliant with the scalable extension of the H.264/AVC specification. Performance measurements show that such a bitstream (containing a bitrate of 17 MBit/s) can be adapted in real-time by a BFlavor-based adaptation framework (with a speed of 27 MBit/s).

Key words: BFlavor, content adaptation, H.264/AVC Scalable Video Coding

1 Introduction

People want to access their multimedia content anywhere, at anytime, and on any device. This phenomenon is generally known as Universal Multimedia Access (UMA) [1]. However, the huge heterogeneity in multimedia formats, network technologies, and end-user devices causes problems for content providers. They want to create their content once whereupon they can distribute it to every possible end-user device using every possible network technology. Scalable video coding is a solution for this multimedia diversity problem. It enables the extraction of multiple (lower quality) versions of the same multimedia resource without the need for a complete encode-decode process.

The use of scalable bitstreams implies the need of an adaptation system in order to extract lower quality versions. To support multiple scalable bitstream

formats, a generic approach is needed for the adaptation of these scalable bitstreams. As explained in [2], bitstream structure description languages can be used to build an adaptation system which supports multiple bitstream formats and which can easily be extended for new bitstream formats. A Bitstream Structure Description (BSD), which is typically an XML document containing information about the high-level structure (information about packets, headers, or layers of data) of a bitstream, is generated by a bitstream-to-BSD parser. 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. Based on an adapted version of the BSD, an adapted bitstream is created with a BSD-to-bitstream parser.

In this paper, special attention is paid to the BSD generation process, which needs the most computations. In Sect. 2, we introduce BFlavor, a new description tool that harmonizes two existing bitstream structure description languages. Section 3 elaborates on some extensions for BFlavor in order to fulfil the shortcomings of the previous version of BFlavor. This enables the creation of a BFlavor code for H.264/AVC's scalable extension which is discussed in Sect. 4. Finally, the conclusions are drawn in Sect. 5.

2 Using BFlavor to Describe Bitstreams

In this paper, we use BFlavor (BSDL + XFlavor) to create BSDs. BFlavor is a combination of two existing technologies for generating bitstream structure descriptions (i.e., the MPEG-21 Bitstream Syntax Description Language (MPEG-21 BSDL) and the Formal Language for Audio-Visual Representation extended with XML features (XFlavor)).

2.1 MPEG-21 BSDL and XFlavor

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 [3]. The Bitstream Syntax Schema (BS Schema), which contains the structure of a certain media format, is used by MPEG-21's 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 the original bitstream.

XFlavor is a declarative C++-like language to describe the syntax of a bitstream on a bit-per-bit basis. It 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 [4]. The XFlavor code, which contains a description

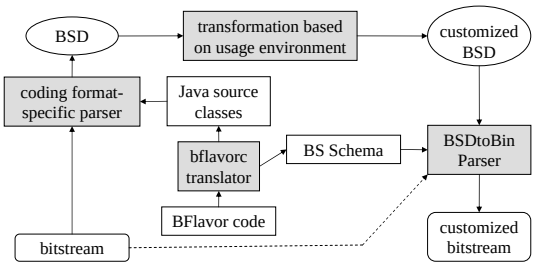

Fig. 1. BFlavor-based adaptation chain

of the syntax of a certain media format, is translated to Java or C++ source classes. This set of source 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.

MPEG-21 BSDL and XFlavor can be used as stand-alone tools [7], but both have pros and contras. XFlavor's 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 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 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.

2.2 BFlavor

As described in [5], BFlavor combines the strengths of MPEG-21 BSDL and XFlavor. As such, BFlavor allows generating a compact high-level BSD at a fast execution speed and with a low memory consumption. The BFlavor-based adaptation chain is illustrated in Fig. 1. The BFlavor code describes the structure of a specific bitstream format in an object-oriented manner. The bflavorc translator uses this code to automatically generate Java source classes. These Java source classes have to be compiled to a coding format-specific parser. This parser generates a BSD which is compliant with the BSDL-1 specification of MPEG-21 BSDL. After the transformation of the BSD, the customized BSD is used by MPEG-21 BSDL's BSDtoBin Parser to create an adapted bitstream, suited for the targeted usage environment. The BSDtoBin Parser needs a Bitstream Syntax Schema (BS Schema). This schema contains the structure of a specific

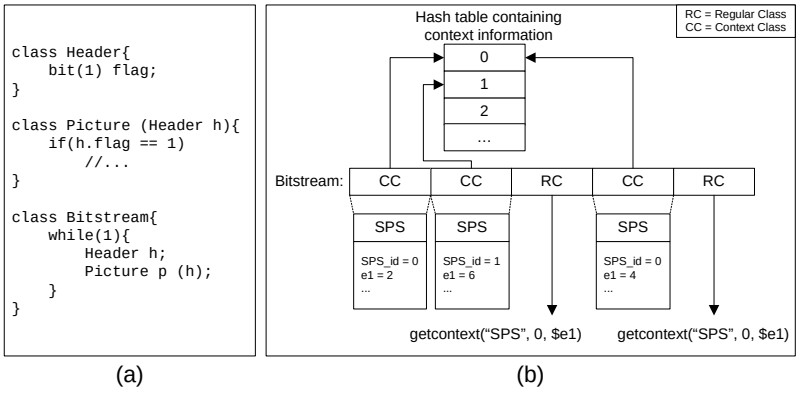

Fig. 2. (a) BFlavor code example for the first type of context information. (b) Steps for processing context classes

bitstream format, analogous to the BFlavor code. Therefore, the bflavorc translator is also able to generate such a BS Schema from the BFlavor code. Thus, the BSDtoBin Parser uses the customized BSD, the generated BS Schema, and the original bitstream to generate an adapted bitstream.

3 Extending BFlavor

Although BFlavor outperforms MPEG-21 BSDL and XFlavor in terms of execution time, memory consumption, and BSD size when parsing for example an H.263-compliant bitstream [6], it still contains some shortcomings in order to describe state-of-the-art media formats. In this section, attention is paid to two important problems: the collection of context information and the occurrence of emulation prevention bytes.

3.1 Collection of Context Information

When parsing a bitstream of a specific coding format, information about previously parsed bits is often needed to correctly steer the parsing process. This information is called *context information*. There are two types of context information. First, there is context information which is located within a fixed distance of the place in the bitstream where the context information is needed. An example of this type of context information is illustrated in Fig. 2(a). In this figure, a fictive BFlavor code fragment of a sequence of pictures with their headers is shown. The picture can access the context information located in the header by making use of the parameter mechanism in BFlavor. As one can see, access to this type of context information is already supported by BFlavor.

A second type of context information is information not necessarily appearing within a fixed distance of the current parsing position. For example, the header

in Fig. 2(a) could be repeated within the bitstream (containing other values). In this case, the parameter mechanism will not work anymore because we do not know which header will be active for the current picture at the time of writing the BFlavor code. A first example of coding formats, which make use of such a context information mechanism, are H.264/AVC and its scalable extension [8] (use of Sequence Parameter Sets (SPS) and Picture Parameter Sets (PPS)) which is further discussed in Sect. 4. A second example is SMPTE's Video Codec 1 (VC-1) (use of Sequence Headers and Entry-point Headers).

We have extended BFlavor to solve the problem of the second type of context information. It is now possible to mark a class in the BFlavor code as a context class (e.g., the SPS class in H.264/AVC). This is done by using the `context` verbatim code, as illustrated on line 5 in Fig. 4. The argument within the verbatim code is the name of the index element of the context class. This index element is used to access a specific instance of the context class. If the context class does not have an index element, no argument is given in the verbatim code. In this case, always the last occurrence of the context class will be active (examples are the Sequence headers and Entry-point headers of VC-1). When a context class is parsed, every syntax element together with its value is stored in a hash table. Based on the value of the index element of the context class, the context class is stored on the right place in the hash table, as illustrated in Fig. 2(b) (the context class `SPS` is stored in the hash table based on the `SPS_id` element).

To access a context class from within the BFlavor code, a new built-in function is defined: `getcontext()`. The working is illustrated in Fig. 2(b). This function can only be used in regular classes (i.e., classes that are not context classes). The value of element `e1` of context class `SPS` with index element equal to `0` is obtained by specifying the following arguments to the `getcontext()` function: the name of the context class (as a string), the value of the index element (as an integer), and the name of the actual element (as a variable prefixed with a $). Note that always the last occurence of the context class with the specific index element value will be accessed. When the `getcontext()` function with the same arguments (as shown in Fig. 2(b)) is called for the second time, the value of the element `e1` is different (it is first equal to 2; the second time, it is equal to 4). Other examples of the `getcontext()` function are shown in Fig. 4.

3.2 Emulation Prevention Bytes

A second problem we have to deal with, is the occurrence of emulation prevention bytes in a bitstream. Coding formats such as MPEG-1/2 Video and MPEG-4 Visual prevent start code emulation by adding restrictions to values of syntax elements (forbidden values). More recent coding formats such as H.264/AVC, VC-1, and the wavelet-based Dirac specification use the mechanism of emulation prevention bytes. When a start code occurs coincidentally in the bitstream, an emulation prevention byte is inserted in order to prevent a start code emulation. For example, when an H.264/AVC coded bitstream contains the start code 0x000001 by accident, the emulation prevention byte 0x03 is inserted resulting in the following bitstream fragment: 0x00000301.

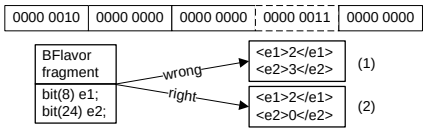

Fig. 3. Example of an emulation prevention byte

These emulation prevention bytes cause problems when using BFlavor, as illustrated in Fig. 3. The fourth byte (0x03) is an emulation prevention byte. The given BFlavor code fragment results in a wrong description (1) shown in Fig. 3. This is because the current version of BFlavor does not take into account the occurrence of emulation prevention bytes.

To solve this problem, we have extended BFlavor with two new verbatim codes (i.e., `emulationBytes` and `emulateBytes`) as illustrated on the first two lines of Fig. 4. The `emulationBytes` verbatim code is used by BFlavor to ignore the emulation prevention bytes. A list of couples *(code with emulation prevention byte, code without emulation prevention byte)* is given as an argument to this verbatim code. With this information, BFlavor can now correctly describe the bitstream given in Fig. 3, resulting in description (2). Obviously, the process to construct a bitstream from a given bitstream structure description (i.e., the BSDtoBin Parser of MPEG-21 BSDL) must also support the use of emulation prevention bytes. Therefore, the `emulateBytes` verbatim code tells an enhanced version of the BSDtoBin Parser which codes have to be emulated. The argument of this verbatim code is a list of couples *(code which cannot appear in the bitstream, emulated version of the code)*.

4 A BFlavor Code for H.264/AVC Scalable Video Coding

4.1 Joint Scalable Video Model

The Joint Video Team (JVT) has started the development of the Joint Scalable Video Model (JSVM), which is an extension of the single-layered H.264/AVC coding format. It is now possible to encode an video sequence once at the highest resolution, frame rate, and visual quality, after which it is possible to extract partial streams containing a lower quality along one or more scalability axes (i.e., the temporal, spatial, and Signal-to-Noise Ratio (SNR) axis).

As discussed above, the scalable extension of H.264/AVC makes use of emulation prevention bytes as well as context classes (i.e., SPSs and PPSs). Therefore, we have created a BFlavor code for JSVM version 5 in order to evaluate the extensions we added to BFlavor. An excerpt of this code is shown in Fig. 4.

4.2 BFlavor Code for JSVM-5

A JSVM-5-compliant bitstream is a succession of Network Abstraction Layer Units (NALUs). Next to the slice layer NALU, there are three main categories

```
 1:%emulationBytes{ (000003, 0000); %emulationBytes}
 2:%emulateBytes{ (000000, 00000300); (000001, 00000301);
 3:               (000002, 00000302); (000003, 00000303); %emulateBytes}
 4:
 5:%context{seq_parameter_set_id%context}
 6:class Seq_parameter_set_rbsp{
 7:    bit(8) profile_idc;
 8:    //...
 9:    bit(8) level_idc;
10:    UnsignedExpGolomb seq_parameter_set_id;
11:    if(profile_idc == 83)
12:        //...
13:}
14:
15:%context{pic_parameter_set_id%context}
16:class Pic_parameter_set_rbsp{
17:    UnsignedExpGolomb pic_parameter_set_id;
18:    UnsignedExpGolomb seq_parameter_set_id;
19:    bit(1) entropy_coding_mode_flag;
20:    //...
21:}
22:
23:class Slice_header(int nal_unit_type, int nal_ref_idc){
24:    UnsignedExpGolomb first_mb_in_slice;
25:    UnsignedExpGolomb slice_type;
26:    UnsignedExpGolomb pic_parameter_set_id;
27:    bit(getcontext("Seq_parameter_set_rbsp",
28:  getcontext("Pic_parameter_set_rbsp", pic_parameter_set_id.value, $seq_parameter_set_id),
29:  $log2_max_frame_num_minus4 ) + 4 ) frame_num;
30:    //...
31:}
32:
33:class Nal_unit_header_svc_extension{
34:    bit(6) simple_priority_id;
35:    bit(1) discardable_flag;
36:    bit(1) extension_flag;
37:    if(extension_flag == 1){
38:        bit(3) temporal_level;
39:        bit(3) dependency_id;
40:        bit(2) quality_level;
41:    }
42:}
```

Fig. 4. An excerpt of the fine-granulated BFlavor code for JSVM-5

of NALUs: SPSs containing information related to the whole sequence; PPSs containing information related to a set of pictures; Supplemental Enhancement Information (SEI) NALUs containing additional information that is not needed to correctly decode the bitstream. The first NALU in the bitstream is a SEI NALU containing scalability information (i.e., information about the different scalability axes). This NALU is followed by a set of SPSs and PPSs. Every slice refers to a PPS and every PPS refers to an SPS. As a consequence, every SPS and PPS has to be kept in memory during the BSD generation process. In our BFlavor code, both the SPS and the PPS classes are context classes. In the slice header class, values of the SPS and PPS can be obtained by making use of the `getcontext()` built-in function. One can also see the signalling of emulation prevention bytes on the first two lines of the code.

4.3 Performance Results

In order to evaluate the performance of our BFlavor-based adaptation chain, we have generated five encoded scalable bitstreams compliant with the JSVM-5 specification. A part of the trailer *the new world* [1] was used with a resolution

[1] This trailer can be downloaded from `http://www.apple.com/trailers`.

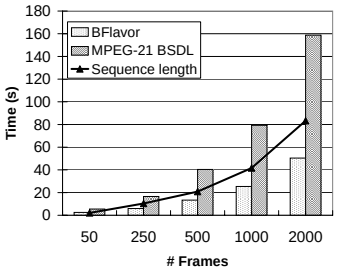

Fig. 5. Total execution times for the whole adaptation chain. In this chain, one spatial layer was removed from the JSVM-5-compliant bitstream

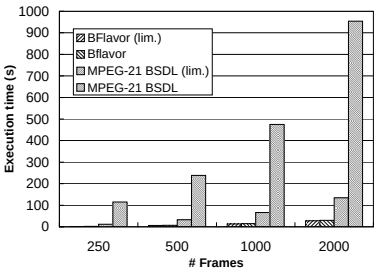

Fig. 6. Execution times for the BSD generation of a JSVM-5-compliant bitstream

of 1280×512 pixels at a frame rate of 24Hz. The encoded bitstreams contain 5 temporal, 4 spatial, and 3 quality levels.

For each bitstream, the corresponding BSD is generated once by making use of BFlavor and once by making use of an optimized version of the BintoBSD Parser of MPEG-21 BSDL, as explained in [9]. The transformation of the BSD is done by making use of Streaming Transformations for XML (STX) [10]. *Joost* (v. 2005-05-21) is used as STX engine. Finally, the adapted bitstream is generated by using an enhanced version of MPEG-21 BSDL's BSDtoBin Parser, using as input the transformed BSD, the BS Schema, and the original bitstream. We also used two BFlavor codes with a different granularity. The first BFlavor code describes the syntax of the JSVM-5 bitstream up to and including the NALU header for the slices. The second BFlavor code describes the syntax of the slices up to and including the slice header. Performance measurements were done on a PC having an Intel Pentium D 2,8GHz CPU and 1GB of RAM at its disposal. Every step was executed five times, whereupon the average was calculated.

As one can see in Fig. 5, the BFlavor-based adaptation chain is executed in real-time (in contrast to the MPEG-21 BSDL-based adaptation chain). Note that in this figure the resolution of the video sequence was rescaled to 640×256 pixels. Both technologies have a linear behavior in terms of execution time, a constant memory consumption (circa 3 MB for BSD generation), and a relatively compact BSD (26 MB uncompressed or 317 KB compressed with WinRAR 3.0's default text compression algorithm when the size of the bitstream is 176 MB).

Although the optimized version of MPEG-21 BSDL's BintoBSD parser shows the same characteristics in the performance measurements, there is a remarkable difference when we look at the BSD generation time (see Fig. 6). When parsing the JSVM-5-compliant bitstream up to and including the slice header, a lot of information has to be retrieved from the active SPS and PPS. This is not the case when parsing up to and including the NALU header. The BFlavor-based adaptation chain can adapt a JSVM-5-compliant bitstream in real-time

Table 1. Execution time results of the adaptation chain. *(limited)* points to a description up to and including the NALU header

Techniques	#Pic.	BSD generation (s)	STX (s)	Bitstream generation (s)	Total time (s)	Speed (Mbit/s)
BFlavor (limited)	50	0.4	1.4	0.8	2.6	2.2
	250	1.7	3.0	1.2	5.9	11.2
	500	6.4	5.0	1.9	13.4	23.5
	1000	13.3	9.0	3.0	25.4	26.2
	2000	27.9	17.0	5.6	50.4	27.9
BFlavor	50	0.4	1.7	1.5	3.6	1.6
	250	2.1	4.0	3.7	9.8	6.8
	500	7.2	6.8	6.7	20.6	15.2
	1000	14.7	13.4	12.3	40.4	16.5
	2000	30.1	26.4	23.7	80.3	17.5
BSDL (limited)	50	3.0	1.5	0.8	5.5	1.0
	250	11.8	3.3	1.4	16.5	4.0
	500	32.8	5.5	2.1	40.4	7.8
	1000	66.2	9.8	3.2	79.3	8.4
	2000	134.5	18.6	5.7	158.8	8.9
BSDL	50	23.9	1.9	1.7	27.4	0.2
	250	114.8	5.3	4.0	124.1	0.5
	500	238.6	9.5	7.2	255.3	1.2
	1000	474.8	17.7	13.3	505.9	1.3
	2000	954.1	34.1	25.5	1013.7	1.4

when parsing up to and including the slice header, as illustrated in Table 1 (80 s needed for a sequence of 83 s). Looking at the MPEG-21 BSDL-based adaptation chain, we see a significant loss of performance when parsing up to and including the slice header. It is clear that the use of hash tables by BFlavor performs much better than the XPath evaluation mechanism used in MPEG-21 BSDL for the retrieval of context information.

5 Conclusions

In this paper, we have extended BFlavor, which is a combination of the fast BSD generation speed and constant memory consumption of XFlavor and the possibility to create the compact BSDs of MPEG-21 BSDL. Support for the transparent retrieval of context information and support for emulation prevention bytes were added to BFlavor. The transparent retrieval of context information is realized by making use of hash tables. Emulation prevention bytes are signalled on top of the BFlavor code. Both features were needed to create a BFlavor code for the JSVM version 5. Performance measurements have shown that a BFlavor-based adaptation chain for JSVM-5-compliant bitstreams operates in real-time.

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. Vetro A., Christopoulos C., Ebrahimi T.: Universal Multimedia Access. IEEE Signal Processing Magazine vol. 20, no. 2 (2003), pp. 16
2. Panis G., Hutter A., Heuer J., Hellwagner H., Kosch H., Timmerer C., Devillers S., Amielh M.: Bitstream Syntax Description: A Tool for Multimedia Resource Adaptation within MPEG-21. EURASIP Signal Processing: Image Communication Journal, vol. 18, no. 8 (2003), pp. 721-747
3. Devillers S., Timmerer C., Heuer J., Hellwagner H.: Bitstream Syntax Description-Based Adaptation in Streaming and Constrained Environments. IEEE Trans. Multimedia, vol. 7 no. 3 (2005), pp. 463-470
4. Hong D., Eleftheriadis A.: XFlavor: Bridging Bits and Objects in Media Representation. Proc. of IEEE Int'l Conference on Multimedia and Expo (ICME), Lauzanne, 2002
5. De Neve W., Van Deursen D., De Schrijver D., Lerouge S., De Wolf K., Van de Walle R.: BFlavor: a Harmonized Approach to Multimedia Resource Adaptation, inspired by XFlavor and MPEG-21 BSDL. Accepted for publication in EURASIP Signal Processing: Image Communication Journal
6. Van Deursen D., De Neve W., De Schrijver D., Van de Walle R.: BFlavor: an Optimized XML-based Framework for Multimedia Content Customization. Proc. PCS 2006, China, 2006
7. 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. Lecture Notes in Computer Science vol. 3767 (2005), pp. 641652
8. Reichel J., Schwarz H., Wien M.: Joint Scalable Video Model JSVM-5. Doc. JVT-R202, 2006
9. De Schrijver D., De Neve W., De Wolf K., Van de Walle R.: Generating MPEG-21 BSDL Descriptions Using Context-Related Attributes. Proc. of the 7th IEEE ISM conference, pp. 79-86, USA, 2005
10. De Schrijver D., De Neve W., Van Deursen D., De Cock J., Van de Walle R.: On an Evaluation of Transformation Languages in a Fully XML-driven Framework for Video Content Adaptation. Accepted for publication in proc. of IEEE ICICIC 2006, China, 2006