2006

An efficient approach to generic multimedia adaptation

Joseph Thomas-Kerr
University of Wollongong, jak09@uow.edu.au

I. Burnett
Faculty of Informatics, University of Wollongong, ianb@uow.edu.au

Christian Ritz
University of Wollongong, critz@uow.edu.au

Publication Details
An efficient approach to generic multimedia adaptation

Abstract
This paper addresses efficiency issues identified in the Bitstream Syntax Description Language used by the MPEG-21 generic multimedia adaptation framework. In particular, when used to adapt modern content formats such as H.264/AVC, the time required for processing increases exponentially relative to the duration of the bitstream. In response, the paper proposes several additional features for the Bitstream Syntax Description Language which reduce the complexity of adaptation using BSDL to a linear function of bitstream duration. These features are implemented and validated using bitstreams of real-world length.

Disciplines
Physical Sciences and Mathematics

Publication Details

This conference paper is available at Research Online: http://ro.uow.edu.au/infopapers/2934
An Efficient Approach to Generic Multimedia Adaptation

Joseph Thomas-Kerr
Whisper Laboratories, School of Electrical Computer and Telecommunications Engineering, University of Wollongong, Wollongong NSW Australia 2522
+61 (2) 4221 3065
joetk@elec.uow.edu.au

Ian Burnett
i.burnett@elec.uow.edu.au

Christian Ritz
chritz@elec.uow.edu.au

ABSTRACT
This paper addresses efficiency issues identified in the Bitstream Syntax Description Language used by the MPEG-21 generic multimedia adaptation framework. In particular, when used to adapt modern content formats such as H.264/AVC, AAC, and SLS, the time required for processing increases exponentially relative to the duration of the bitstream. In response, the paper proposes several additional features for the Language which reduce the complexity of adaptation using BSDL to a linear function of bitstream duration. These features are implemented and validated using bitstreams of real-world length.

Categories and Subject Descriptors
H.5.1 [Multimedia Information Systems] – Audio input/output, Evaluation/methodology, Video (e.g., tape, disk, DVI)
H.3.4 [Systems and Software] – Performance evaluation (efficiency and effectiveness)
H.3.5 [Online Information Services] – Data sharing

General Terms
Algorithms, Performance, Standardization, Languages.

Keywords
Multimedia Content Adaptation, Bitstream Syntax Description, H.264/AVC, MPEG-21.

1. INTRODUCTION
Dynamic adaptation of scalable multimedia content has received considerable attention in recent literature [3, 4, 12]. This enables streaming multimedia to be consumed by a wide range of devices with varying processing power and memory capacities, across networks with divergent bandwidth and error characteristics. In particular, Devillers [4] proposes a generic adaptation architecture which allows processing nodes to perform the adaptation without specific knowledge about the bitstream being performed. Such an architecture facilitates interoperability for new content formats by enabling adaptation without modifying existing infrastructure.

This generic adaptation process is illustrated in Figure 1. It is based on performing the adaptation on a high-level representation of the bitstream rather than on the bitstream itself. This high-level representation is known as a Bitstream Syntax Description (BSD). This work was partially funded by the Smart Internet CRC.

Figure 1 – Generic Adaptation (adapted from [11])
and is extracted from the bitstream via a generic processor, aided by a description of the content format – the Bitstream Syntax (BS) Schema file. This BSD is then transformed according to a set of instructions, which a BSD to Binary processor applies to the bitstream in order to execute the adaptation.

In [4] Devillers proposes a language for the BS Schema and BSD, which is discussed in section 2.1, and has been adopted as part of the MPEG-21 Multimedia Framework [11]. However, recent content formats including the H.264/AVC video standard [8], AAC [1], and the Scalable to Lossless (SLS) audio coder [9] pose significant scalability problems for generic adaptation via BSDL. Specifically, BSDL cannot efficiently describe numerous structures which are prevalent in these formats, and the description tools which are available lead to a processing time which increases relative to the bitstream duration according to $O(n^2)$. This leads to the situation where a two minute H.264/AVC bitstream requires more than an hour to process using BinToBSD on a Pentium 4 (see section 5).

After discussing the available alternatives to BSDL in section 2, we consider the problems with generic adaptation of recent coding formats in section 3. Section 4 presents our proposed solutions, and the consequent performance improvements are demonstrated in section 5.

2. GENERIC BITSTREAM DESCRIPTION

2.1 Bitstream Syntax Description Language
The BS Schema language [4] extends of XML Schema [10] with mechanisms to specify the bit-stream equivalent for XML data types, as well as conditional structures to describe parsing of a bitstream. As a result, a BS Schema elegantly describes both the bitstream, its XML representation (the Bitstream Syntax Description, or BSD), and the translation between the two. It was designed specifically to support the generic adaptation architecture shown in Figure 1.
3.3 Dynamic length bit-fields

Another common feature in H.264/AVC is header fields which use a variable number of bits. Some – such as those with expGolomb encoding – may be parsed without external information. For others, the length of the field is indicated elsewhere within the bitstream. The frameNum field in Figure 2(a) is one such example. Another is the maximum bit-plane side-information in the SLS scalable audio coder [9].

BSDL provides a mechanism for resolving XML Schema union datatypes, which allows fields with variable length to be described (Figure 3). However, an XPath expression is required for each possibility within the union. When the location of the length data is specified in a SPS or PPS (as is in fact usually the case), the computation complexity of the BS Schema increases to O(p^n) (the XPath expressions hidden by "...and so on...") in Figure 3 are that of Figure 2(b), and p is the number of ifUnion elements evaluated before a true result is returned).

Additionally, each possible bit-length must be individually enumerated. While not a significant problem for the given example, a field with possible lengths from 1 to 64 or even 32 bits will quickly become untenable.
4. PROPOSED SOLUTIONS

Adaptation of H.264 bitstreams using BSDL in its present state, while theoretically possible, is implausible for any bitstreams of real-world length, due to the scalability issues discussed in section 3, as demonstrated (section 5). However, it is possible to address these issues with a number of simple modifications.

4.1 User-defined simple data types

Rather than specify encoded data types as part of the BSDL standard, we propose a simple addition to BSDL to allow custom parsing operations to be specified within the BS Schema using ECMAScript [5]. This is shown in Figure 4, where user-defined data types are a restriction of a new type – bs1: userType.

The content of the restriction is a bs1:script node, which is required to specify two functions – BinterBSD() for the binary to BSD process, and BSDtoBin(value) for the reverse. BSD provides read(nBits) and write(value, nBits) functions to allow the user functions to access the bitstream.

The body of the BinterBSD() function (Figure 4) implements the expGolomb parsing process specified by [13].

4.2 User-defined XPath variables

The complexity of the dereferencing operation presented in Figure 2(a) may be greatly simplified by recognizing that the set of valid PPS (and equally SPS) at any point in time may be represented as an object array, indexed by the respective ID. Any time a new parameter set is encountered in the bitstream, it is inserted into the array in the appropriate location, potentially replacing a parameter set that was previously there. XPath does not directly support an array in the appropriate location, potentially replacing a parameter set implemented with a more efficient XPath processor (Saxon 8.6.4) so as to minimize implementation deficiencies.

5. EXPERIMENTAL RESULTS

Each of the proposed solutions were tested via modifications to the BSDL reference software [7], on H.264/AVC bitstreams of varying length. Tests were carried out on a P4 1.7Ghz machine running Linux and the Sun JVM 1.5.0. Each trial was repeated varying length. Tests were carried out on a P4 1.7Ghz machine running Linux and the Sun JVM 1.5.0. Each trial was repeated

4.3 Bit-Length Facet

The use of xsd:union to specify variable length fields such as frameNum, while possible, is not intuitive. A far simpler solution

\[
\text{bs2:ifUnion value} = \ldots /\text{avc:fNumLength\_minus4} = 0
\]

\[
\text{bs2:ifUnion value} = \ldots /\text{avc:fNumLength\_minus4} = 1
\]

\[
\text{bs2:ifUnion value} = \ldots /\text{avc:fNumLength\_minus4} = 2
\]

Figure 3 – H.264/AVC frameNum simple type

51

Figure 4 – BS Schema definition of expGolomb data type

is to provide a facet which directly calculates the length (in bits) of the type being defined. Indeed, such a facet exists within BSDL to specify a dynamic length in bytes – bs2:length; it is easily extended with a unit attribute (Figure 5). This has the effect of replacing numerous XPath expressions (12 in the example – Figure 3) with a single expression.

5. EXPERIMENTAL RESULTS

Each of the proposed solutions were tested via modifications to the BSDL reference software [7], on H.264/AVC bitstreams of varying length. Tests were carried out on a P4 1.7Ghz machine running Linux and the Sun JVM 1.5.0. Each trial was repeated

\[
\text{bs2:variable name} = \text{sps}
\]

\[
\text{position} = \text{avc:spsID+1}
\]

Figure 6 – sps variable

Since this expression doesn’t involve any iteration over the set of slices, its evaluation time is constant, and the O(n^2) complexity observed in the base specification reduces to O(n).

In order to enable the use of arbitrarily defined XPath variables in BSDL, a mechanism must be provided to assign their value, and also their position within the node-set. We propose the addition of a bs2:variable element for this purpose, as shown in Figure 5.

<element name="sequenceParameterSet">
  <complexType>
    <extension base="annotated">
      <attribute name="value" type="bs2:integerXPathExpr"/>
      <attribute name="unit" use="optional" default="byte">
        <simpleType>
          <restriction base="string">
            <enumeration value="bit"/>
            <enumeration value="byte"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </complexType>
</element>

Figure 5 – Schema for Schema definition of bs2:length

<element name="sequenceParameterSet">
  <complexType>
    <extension base="annotated">
      <attribute name="value" type="bs2:integerXPathExpr"/>
      <attribute name="unit" use="optional" default="byte">
        <simpleType>
          <restriction base="string">
            <enumeration value="bit"/>
            <enumeration value="byte"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </complexType>
</element>

Figure 5 – Schema for Schema definition of bs2:length

\[
\text{bs2:ifUnion value} = \ldots /\text{avc:fNumLength\_minus4}
\]

Figure 3 – H.264/AVC frameNum simple type

51
three to ten times, and the results averaged (in each case the result of the first run was excluded because it includes additional time for JVM initialization and class loading).

As can be seen, the time required to process the bitstream using the unmodified BinToBSD software (the base series in Figure 7a) increases according to \( O(n^2) \) – For a two minute bitstream, BinToBSD processing time exceeds an hour.

Subsequent series’ in Figure 7a show the performance of the various proposed solutions. The introduction of user-defined types (4.1) is not intended to improve the scalability of the system. Rather, it enhances extensibility, by simplifying adoption of new encoded data types, and results demonstrate that the overhead of this enhancement is negligible.

The Bit-Length Pseudofacet (4.3) reduces parse time by approximately 75%, because of the removal of the \( p \) term discussed in section 3.3. However, complexity is still \( O(n^2) \); the system still cannot scale to movies of real-world duration. In contrast, the introduction of user-defined variables (4.2) reduces the complexity to \( O(n) \). This results in a Bitstream Syntax Description Language that may be used to perform generic adaptation on real-world H.264/AVC streams – the series’ labelled all in Figure 7a and b show the performance of BinToBSD with all modifications applied. The latter verifies the \( O(n) \) performance of the algorithm for bitstreams in excess of an hour. The resulting BSDL processing is around 10 times faster than playback.

A further benefit of the proposed modifications to BSDL is shown by the lowMemory series. All XPath expressions are now evaluated against the set of user variables – meaning that the BSDL itself is no longer required to be stored in memory. This has the effect of removing the correlation between bitstream duration and memory consumption, which is approximately 5 MB.

6. CONCLUSIONS
This paper has demonstrated that the Bitstream Syntax Description Language (BSDL) displays \( O(n^2) \) complexity when processing bitstreams of H.264/AVC and other recent coding formats including AAC and SLS, and is consequently unable to process such bitstreams of real-world duration. We have proposed a number of additions to BSDL which have the effect of reducing the complexity of BinToBSD processing to \( O(n) \). These results have been verified with bitstreams up to an hour in length. The

7. REFERENCES