FORMAT-INDEPENDENT MULTIMEDIA STREAMING

Joseph Thomas-Kerr, Ian Burnett, Christian Ritz

University of Wollongong, CRC SIT (Smart Internet Technology)
joetk@elec.uow.edu.au, i.burnett@elec.uow.edu.au, chritz@elec.uow.edu.au

ABSTRACT

The Bitstream Binding Language (BBL) is a new technology developed by the authors and being standardized by MPEG, which describes how multimedia content and metadata can be mapped onto streaming formats. This paper describes a particular application of BBL – format-independent multimedia streaming. This means that streaming servers no longer require additional software modules in order to support new content formats as they are introduced. Instead, the server requires only a BBL description of the mapping between the content format and the stream, and any content in the new format may then be delivered by the streaming server. This approach is validated using the H.264/AVC format as an example, and performance data are provided.

1. INTRODUCTION

Multimedia technology continues to develop at an ever increasing rate. New audio, video, and hybrid encoding formats are regularly developed, and the number of devices accessing or processing multimedia content has grown exponentially, as has their variability in terms of available processing power. This diversity hampers interoperability because tools that handle multimedia data are generally required to have custom software written to handle each format. As new content formats are defined, they do not become useful until software has been written and deployed for the set of platforms which process or consume them, including streaming servers, multimedia gateways, and consuming devices from PCs to mobile devices.

It is clear that the complexity of many operations on multimedia content mandates the use of custom software. However, other approaches have been developed which address certain tasks with multimedia data in a generic – format-independent – way. Where format-specific information is required, it is provided by a data file which is simple, portable, and needs to be written only once. This considerably simplifies the adoption of new media formats. Two examples of this generic approach are Flavor [1] – an automatic parser generator, and the Bitstream Syntax Description Language (BSDL) [2], which describes the high-level syntax of a scalable bitstream for the purpose of content adaptation.

This paper addresses format-independent multimedia streaming. Using this approach, support for new content formats is provided via a simple data file, aiding their adoption. There are a number of existing tools which provide partial solutions to this problem (discussed in section 2), but these merely shift the format-specific software modules from the streaming server to another application. Instead, this paper demonstrates how the Bitstream Binding Language (BBL) may be used to enable format-independent streaming. BBL was previously proposed by the authors [3], and is being standardized as part of MPEG-21 [4] – a format-agnostic framework for multimedia transaction and delivery.

This work was partially funded by the Smart Internet CRC

BBL is a generic language which describes how to map collections of multimedia content and metadata into output bitstreams. It specifies how to packetize and schedule both binary and XML content, so that – for example – an MPEG-21 collection can be mapped onto an RTP or MPEG-2 Transport Stream, regardless of the format of the individual media or metadata content.

Section 3 discusses how BBL is applied as a format-independent streaming server, and section 4 presents an example application – streamed delivery of H.264/AVC over RTP. Results of this example scenario are presented in Section 5, and Section 6 concludes the work.

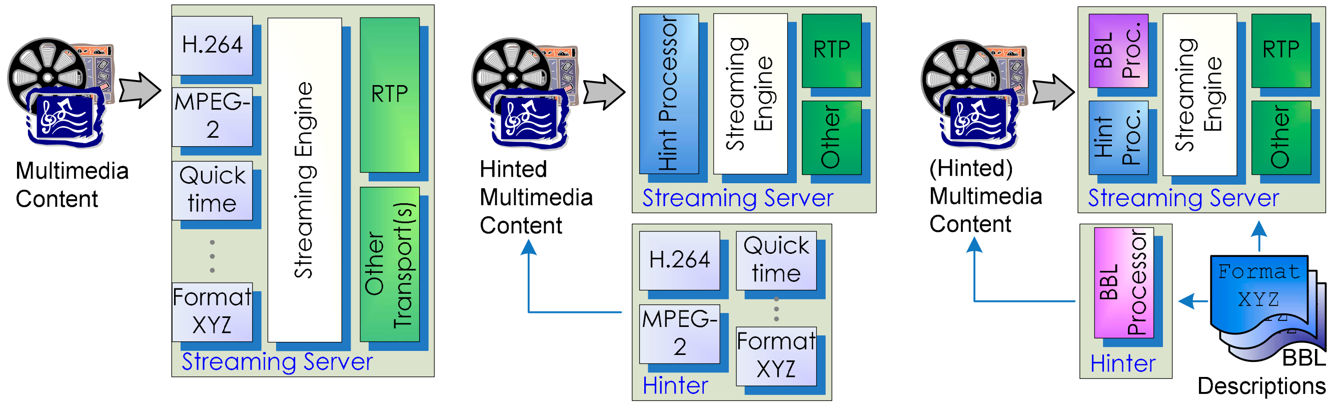
2. A GENERIC STREAMING SERVER

Figure 1 shows a number of possible architectures for a multi-format streaming server. The simplest case (Figure 1a) has software modules for each supported format to process content of that form and ready it for streaming. When a new content format is developed, additional software modules must be developed and integrated into the streaming server in order to support the new format.

2.1. Hint Tracks

Quicktime files [5] and the ISO file format [6] provide a mechanism known as "hint tracks" which suggest how a server could stream the content in the file. This means that the streaming server itself (Figure 1b) no longer needs to explicitly provide software to support each individual content format (at least for content which may be contained in a Quicktime or ISO file). Instead, the server may stream the content by processing the hint track(s). This architecture significantly increases scalability, since hint track processing is essentially a sequence of byte-copy operations – requiring much less computation than parsing the bitstream to determine how it is to be streamed.

This computation is still required, but it may now be conducted offline – and often on a different machine – in a



(a) Monolithic Streaming Server **(b) Hinted Streaming Server** **(c) BBL-based Streaming Server**

Figure 1 – Streaming Server Architectures

separate hinter application. Consequently, the hinter still requires specific software to process each individual format, and must be updated in order to support new encodings as they are developed. In practice, there are significantly more hinter applications than there are streaming servers. As a result, interoperability for new content formats is made even more difficult, since the number of applications for which new software must be developed is substantially larger.

2.2. gBSD-based 'generic streaming'

Ransburg et al have considered this issue, and devised a 'gBSD-based generic streaming server' [7]. gBSD – generic Bitstream Syntax Description – is a tool related to BSDL (see section 1) which uses a single XML Schema to describe all bitstreams. Ransburg et al propose "to use an extended version of the gBSD as a hint file." Specifically, the gBSD is extended with a marker to identify Access Units (AUs – defined as the smallest unit of data to which timing may be attached) and specify a timestamp for each AU.

While the gBSD schema is generic (format-independent), the generation process is not. Generating a gBSD for a piece of content requires specific software which is able to parse the format in question. Consequently, the 'gBSD-based generic streaming server' has essentially the architecture of Figure 1b. That is, streaming itself is generic, but the hinting application (this time based on gBSD) is not – it requires additional software to support new content formats.

Additionally, the identification of access units does not generally provide sufficient information to stream content. Many content formats place additional restrictions on packetization below the level of an access unit. For example, the specification for H.264/AVC over RTP [8] places constraints on the fragmenting of NAL units (part of an AU).

Content formats also often require custom header information to be transmitted as part of the stream – for example, H.264/AVC or MPEG-4 over RTP [8, 9]. The fields in the custom header are generally based on the payload, but not included within it. For these reasons, the extended gBSD hint file provided by Ransburg et al does not provide enough information to stream the content.

2.3. BBL-based streaming server

In contrast, a streaming server based on BBL (Figure 1c) does not require any format-specific software. All information required to stream content of a particular format is stored in a BBL description file. Whereas a hint track or extended gBSD describe one piece of content, a BBL description relates to all content of that format.

This means that support for new encodings as they are developed may be provided by merely disseminating a BBL description. No additional software modules need to be written, which considerably simplifies the process of providing streaming support for new formats.

The streaming server may use the BBL description to process content on-the-fly. This is useful in a live streaming situation – where the content is not available for offline hinting, or where dynamic network conditions can guide the streaming process. Alternatively, a BBL description may be used to control a hinter, processing the content offline and providing the scalability benefits of hinted streaming.

The BBL language addresses the shortcomings highlighted in section 2.2. It allows the identification of syntactical content structures at any level – not just Access Units – and it provides the ability to add custom headers or other data to packets as required.

3. BITSTREAM BINDING LANGUAGE

Figure 2 depicts the model used by BBL to enable format-independent multimedia streaming. Given an input bitstream, BBL describes how to identify the content to be included in each packet. It provides instructions to determine the timing of the packet, and the value of header fields. The latter may involve both standard headers (such as the RTP header), and format-specific headers, where it is necessary to define both the syntax and the values of each field.

Identification of packet content: In general, multimedia bitstream formats are made up of numerous layers of syntactical structures. In a streamed delivery scenario, the packetization of the bitstream must proceed on the basis of these structures, in order to ensure timely delivery and facilitate error resilience [8, 9]. A format-independent mechanism is therefore required that is able to identify the

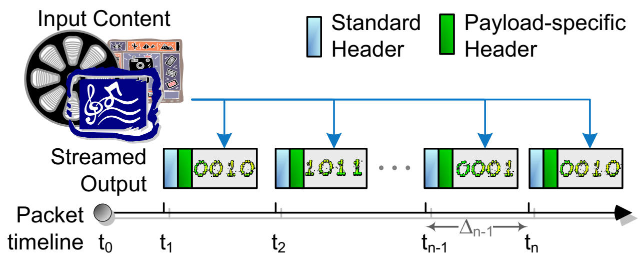


Figure 2 – Abstract model for format-independent streaming

syntactical elements of a bitstream, such as Flavor [1] or BSDL [2]. BBL uses BSDL for this purpose, because it allows bitstreams to be described in varying levels of detail (for example, header fields may be explicitly described, while payload data remains hidden).

BSDL exposes the structure of a bitstream as XML, which allows standard XML tools to operate on the binary data. BBL makes extensive use of XPath [10] – a language which provides addressing and querying for XML along with significant processing functionality – to identify packet content and declare timing information.

To specify packet content, an XPath expression selects the set of content to be packetized, and a number of rules are applied to determine how to divide the set into individual packets. The available rules are based on the requirements of numerous use cases, including [8] and [9]. They may include a maximum packet size or duration, a limit on the count of a particular structure within a single packet, or that particular sub-structures must remain whole.

Timing information: Some content formats have a constant or variable packet duration which may be read or inferred from the bitstream (for example, Theora, MP3, MPEG-4 Visual). Others use explicit timestamps (such as MPEG-2 Program Streams). H.264/AVC, on the other hand, contains no internal temporal information. It must be provided externally.

BBL supports all of these cases. Packets are placed on a timeline beginning at t_0 where the delivery time t of packet n is given by

$$t_n = t_{n-1} + \Delta_{n-1} \qquad \ldots (1)$$

where Δ represents the duration of a packet (Figure 2). Both t_n and Δ_n may be specified in the BBL description. Typically, only one is used for a particular session, however there are some situations where resynchronization points in the bitstream may have an explicit timestamp, while other packets are given a duration offset.

Temporal information is declared in BBL using two XPath expressions. The first identifies the bitstream segment(s) to which the temporal parameter is to be applied. The second describes how the timestamp or duration is calculated from the fields within the bitstream segment (which have been identified by BSDL), and/or values which have been stored from other sections of the bitstream.

Standard Header data: On the Internet, RTP is used almost exclusively as the streaming protocol. However, BBL was designed for use in multiple domains (such as Digital TV, where MPEG-2 Transport Streams are typically used),

```
<packetStream>
  <contentTemplate>
 <include ref="/avc:h264/avc:slice |
                              /avc:h264/avc:parameterSet" depth="-1">
     <!-- ... -->
   </include>
  </contentTemplate>
  <variables>
   <!-- ... -->
   <assign name="delTime" value="if ($newAU)
              then $delTime + $framePeriod else $delTime"/>
   <assign name="expectedPicOrder"
     value="if ($nalType = 5) then 0
                  else if ($newAU) then $expectedPicOrder + 2
                  else $expectedPicOrder"/>
   <assign name="timestampOffset"
          value="if (./avc:h264/avc:slice) then
        $frameTime * ($picOrder - $expectedPicOrder) div 2
          else $timestampOffset"/>
  </variables>
</packetStream>
```

Figure 3 – Extract of BBL description for H.264 over RTP

and provides a mechanism to specify alternative output stream handlers. This handler mechanism is also extensible, so that new streaming protocols may be easily integrated into the BBL framework.

A handler receives the data for each packet, along with its delivery timestamp, and other parameters defined specifically for the handler. For the RTP handler, this includes the timebase, payload type, SDP data, and marker bit. These parameters provide the values for some of the RTP header fields. Others fields, such as the sequence number and SSRC, are not media specific – they are set by the streaming server without information about the content.

Payload-Specific headers: The mechanism used to specify packet content may contain multiple separate elements. This allows payload-specific headers to be added to packet data. BSDL is used to specify the structure of the header, and XPath expressions to calculate the field values.

4. BBL FOR H.264/AVC OVER RTP

H.264/AVC [11] is a recent video encoding format used as an example application for BBL, since it has a number of characteristics distinct from previous coding formats which make generic streaming more challenging. These include parameter sets and a lack of internal timing information.

A H.264 stream is made up of sequences of Network Abstraction Layer (NAL) Units. These contain slices of picture data, parameter sets or other supplementary data. In general, each NAL unit in the input bitstream is carried in a separate RTP packet [8]. In the BBL description (Figure 3), this is accomplished by selecting the NAL units to be packetized using the `include` element, then applying fragmentation rules to separate the NAL units into packets (not shown).

In order to derive timing information for each NAL unit, their association to Access Units (AU) must be identified. The bitstream may contain a specific AU delimiter which simplifies this process but its presence is not guaranteed and so cannot be assumed. Consequently, the general process specified in clause 7.4.1.2.4 of [11] is used to calculate the Boolean variable $newAU, by detecting changes in certain field values between one slice NAL unit and the next.

The NAL Units in an AU have the same delivery time ($delTime) – based on an external frame rate. The RTP header timestamp, however, must be offset from the delivery time according to the display order of the pictures. This is implemented in BBL by comparing the `pic_order_cnt` field ($picOrder) to its expected value ($expectedPicOrder).

5. EXPERIMENTAL RESULTS

A prototype implementation of the BBL processor has been developed, and this section presents initial test results. The results demonstrate that the algorithm performs correctly, and also that its complexity is low enough to enable multiple on-the-fly sessions. A full evaluation of the scalability of the algorithm is pending an optimized implementation, however in general, BBL processing may be conducted offline to produce hint tracks, such that scalability is not critical.

The tests were conducted using a QCIF test sequence of 382 frames (15.3 seconds at 25fps). The sequence was encoded using the H.264 reference software in three configurations, to validate the BBL description across a range of significantly different H.264 bitstreams. Each test was repeated ten times, and the results averaged. The configurations used were:

(a) Baseline Profile with NAL size limited to 100 bytes (a profile targeted towards mobile applications [12]);

(b) Main profile, (introducing bi-predicted frames), NAL size 1500 bytes; and

(c) Extended profile using data partitioning – an error resilience feature provided by H.264 where each slice is split into 3 portions with varying loss importance.

The correctness of the algorithm is validated by comparing the output of the BBL processor to the RTPdump output of the H.264/AVC reference software. In all cases, both are identical.

Scalability is assessed by measuring memory usage, and CPU time as a proportion of the duration of the sequence (% CPU utilization), for each test[1].

Results are shown in Table 1. CPU and memory usage both indicate that the prototype system will scale to several tens of simultaneous sessions – with the exception of the baseline profile (test (a)). In this case, the CPU utilization is significantly larger due to the greater number of NAL units (and packets) to be processed. To improve scalability in such an application, AU delimiters could be employed to reduce processing complexity, or offline processing (hinting) used with greater priority.

6. CONCLUSION

This paper has demonstrated how the Bitstream Binding Language may be used to implement a format-independent streaming server. This facilitates multimedia interoperability in the face of newly developed content formats, by enabling streaming support via a data file (the BBL instructions),

Test configuration	NAL unit count	% CPU utilization	Memory usage (Mb) Avg.	Max.
(a) Baseline	4555	23.0%	1.62	6.63
(b) Main	459	3.1%	0.94	4.49
(c) Extended	1146	5.5%	1.16	4.95

Table 1 – On-the-fly BBL Processing, performance results

rather than requiring new software to be developed to support the new format. BBL can be used to process content on-the-fly, or offline to produce highly scalable hint tracks whilst still providing format-independent streaming.

This approach has been tested using the H.264 video format. It produces RTP streams with the correct timing and data, and the prototype implementation may scale to several tens of simultaneous sessions, depending on the number of NAL units which much be processed per second.

Future work for BBL will focus on mechanisms to improve the scalability of on-the-fly processing, including the provision of a method to utilize XPath extension functions. This will significantly reduce the number of XPath expressions to be processed for each NAL unit.

7. REFERENCES

[1] A. Eleftheriadis, "Flavor: a language for media representation," presented at Fifth ACM Intl. Conf. on Multimedia, 1997.

[2] M. Amielh and S. Devillers, "Bitstream Syntax Description Language: Application of XML-schema to Multimedia Content Adaptation," presented at WWW, 11th Intl. Conf. on, 2002.

[3] J. Thomas-Kerr, et al., "Bitstream Binding Language - Mapping XML multimedia containers into streams," presented at Multimedia & Expo, 2005. IEEE Intl. Conf. on, 2005.

[4] ISO/IEC, "CD 21000-18, IT - Multimedia framework (MPEG-21) -Part 18: Digital Item Streaming," 2005.

[5] Apple, "QuickTime File Format." http://developer.apple.com /documentation/QuickTime/QTFF/qtff.pdf, 2001.

[6] ISO/IEC, "14496-12, IT - Coding of audio-visual objects - Part 12: ISO base media file format," 2005.

[7] M. Ransburg and H. Hellwagner, "Generic Streaming of Multimedia Content," in *Internet & Multimedia Sys. & Apps.*, 2005.

[8] S. Wenger, "H.264/AVC over IP," *Circuits & Systems for Video Technology, IEEE Trans. on*, vol. 13, pp. 645, 2003.

[9] J. v. d. Meer, et al., "RFC3640: RTP Payload Format for Transport of MPEG-4 Elementary Streams." http://www.ietf.org/rfc/rfc3640.txt, 2003.

[10] A. Berglund, et al., "XML Path Language (XPath) 2.0," in *Working Draft*. http://www.w3.org/TR/xpath20, 2005.

[11] ITU-T, "Recommendation H.264: Advanced video coding for generic audiovisual services," 2005.

[12] T. Wiegand, et al., "Overview of the H.264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, pp. 560, 2003.

[1] The BBL processor was implemented in Java and tested on a P4 3.0GHz PC, 1Gb of RAM, Windows XP & Sun 1.5.0_04 JVM. The reported memory usage excludes that used by the JVM itself.