2008

Format-independent rich media delivery using the bitstream binding language

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

Follow this and additional works at: <https://ro.uow.edu.au/infopapers>

Part of the [Physical Sciences and Mathematics Commons](#)

Recommended Citation

Thomas-Kerr, Joseph; Burnett, I.; and Ritz, Christian: Format-independent rich media delivery using the bitstream binding language 2008.
<https://ro.uow.edu.au/infopapers/3152>

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Format-independent rich media delivery using the bitstream binding language

Abstract

Several recent standards address virtual containers for rich multimedia content: collections of media with metadata describing the relationships between them and providing an immersive user experience. While these standards - which include MPEG-21 and TVAnytime - provide numerous tools for interacting with rich media objects, they do not provide a framework for streaming or delivery of such content. This paper presents the bitstream binding language (BBL), a format-independent tool that describes how multimedia content and metadata may be bound into delivery formats. Using a BBL description, a generic processor can map rich content (an MPEG-21 digital item, for example) into a streaming or static delivery format. BBL provides a universal syntax for fragmentation and packetization of both XML and binary data, and allows new content and metadata formats to be delivered without requiring the addition of new software to the delivery infrastructure. Following its development by the authors, BBL was adopted by MPEG as Part 18 of the MPEG-21 Multimedia Framework.

Disciplines

Physical Sciences and Mathematics

Publication Details

Thomas-Kerr, J., Burnett, I. S. & Ritz, C. H. (2008). Format-independent rich media delivery using the bitstream binding language. *IEEE Transactions on Multimedia*, 10 (3), 514-522.

Format-Independent Rich Media Delivery Using the Bitstream Binding Language

Joseph Thomas-Kerr, *Student Member, IEEE*, Ian Burnett, *Senior Member, IEEE*, and Christian Ritz, *Member, IEEE*

Abstract—Several recent standards address virtual containers for rich multimedia content: collections of media with metadata describing the relationships between them and providing an immersive user experience. While these standards—which include MPEG-21 and TVAnytime—provide numerous tools for interacting with rich media objects, they do not provide a framework for streaming or delivery of such content. This paper presents the Bitstream Binding Language (BBL), a format-independent tool that describes how multimedia content and metadata may be bound into delivery formats. Using a BBL description, a generic processor can map rich content (an MPEG-21 digital item, for example) into a streaming or static delivery format. BBL provides a universal syntax for fragmentation and packetization of both XML and binary data, and allows new content and metadata formats to be delivered without requiring the addition of new software to the delivery infrastructure. Following its development by the authors, BBL was adopted by MPEG as Part 18 of the MPEG-21 Multimedia Framework.

Index Terms—Bitstream binding language (BBL), MPEG-21 multimedia framework, multimedia communication, streaming.

I. INTRODUCTION

THE amount of multimedia content available via the Internet, and the number of formats in which it is encoded, stored, and delivered continues to grow rapidly. So too the number and diversity of the devices and software applications which produce, process, and consume such content. This constantly changing landscape presents an increasing challenge to interoperability, since more and more software and hardware must be upgraded as new formats are developed. However, many of the operations performed on multimedia content are similar across coding formats. Consequently, numerous approaches have been developed that process content using generic software, with format-specific details provided in a data file. This considerably simplifies interoperability, since support for a new content format may be provided by disseminating a simple file, rather than requiring application providers to extend and modify their software. Examples of this approach are Flavor [2], which is a C++-like syntax that describes the structure of multimedia content to automate bitstream parsing, and the recent MPEG Reconfigurable Media Coding Project

[1], which targets the provision of a declarative language for building entire codecs by combining primitive function blocks.

There have been a number of approaches developed in recent years that are designed to facilitate transaction and processing of rich media,¹ including MPEG-21 [3] and TVAnytime [4]. These technologies are also—to greater and lesser degrees—format-independent in terms of the media and metadata components which make up objects within the respective approach. However, while they provide numerous tools for interacting with rich media objects, a fully format-independent framework for streaming and delivery of such objects is not available. Such technologies provide various virtual containers for multimedia content and metadata, but do not provide delivery mechanisms for such containers over the numerous channels on which users will want to access the content.

Furthermore, the format-independent nature of such containers means that it is not possible to provide a single mapping into a particular delivery format (such as the Real-time Transport Protocol (RTP) [6]). Mappings may exist for some content types that could be placed in the container, MPEG-1/-2 [7], or digital item adaptation metadata [8] for example. However, in general, the existence of the required mapping for every potential component cannot be relied upon. A solution is therefore required which can specify any and all required mappings between the many formats in which the content is stored and the many channels on which it is to be delivered. These mappings must be specified in a manner that can be machine-processed to actually perform the delivery. As with Reconfigurable Video Coding and Flavor, this approach would allow new formats to be supported without requiring any additional software to be written.

As will be seen in Section II-D, technologies exist for some components of this task, but a solution for the problem as a whole is lacking. This paper presents the BBL, which is the authors' approach to the issue. BBL is a high-level declarative language that specifies how multimedia content and metadata is to be bound into a delivery format. Following its development by the authors, BBL was adopted by MPEG as Part 18 of the MPEG-21 Multimedia Framework (Digital Item Streaming) [5].

Application scenarios for this technology are covered in more detail in Section II-A, but it is important to point out that BBL is not a new delivery format. BBL does not replace RTP, or any other protocol or file format. Instead, it provides meta-information that describes to a streaming server (or other application; see Section II-A) how to deliver content.

Given the application scenarios, Section II-B goes on to identify requirements for format-independent rich media de-

Manuscript received January 18, 2007; revised September 20, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Marco Rocchetti.

The authors are with the Whisper Laboratories, North Wollongong, NSW 2522 Australia (e-mail: joetk@elec.uow.edu.au; i.burnett@elec.uow.edu.au; chriz@elec.uow.edu.au).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TMM.2008.917337

¹That is, multimedia content augmented with metadata to enhance the experience of users interacting with the content.

livery. A model for the delivery framework is then developed in Section II-C, and related work discussed in Section II-D. Section III presents BBL in action, showing how it implements the model, and Section IV gives some prototypical computational complexity data and further discussion.

II. DEVELOPING A FRAMEWORK FOR FORMAT-INDEPENDENT MULTIMEDIA DELIVERY

This section presents the development of a format-independent framework for rich media delivery. Section II-A introduces some relevant application scenarios, including multichannel delivery, distributed content adaptation, generic syntax translation, and virtual container assembly. Section II-B identifies several requirements for the framework, from which an abstract model for the description of rich media delivery is developed (Section II-C). Finally, related work is discussed in Section II-D.

A. Application Scenarios

- 1) *Multichannel, rich content delivery* [3]. Rich multimedia content may often be transmitted over several channels. For example, a media provider may wish to distribute their content to some users who connect via a PC and the wired Internet, others on cellular smart-phones, and further users with a digital television receiver. Currently, the content provider would often store a separate version of the content for each usage scenario, and use specialized software to map the content onto each channel. Instead, using a format-independent approach, multiple bindings of the content can be provided as inputs to a generic processor. Each binding specifies how to adapt the content and metadata to the particular characteristics of the channel and/or receiving device, and describes how to map (and if necessary, transcode) the data into each output.
- 2) *Format-independent, distributed content adaptation* is a use case targeted specifically targeted by Digital Item Streaming [3]. It involves transmitting XML metadata that describes the high-level structure of scalable multimedia data alongside the content itself. This metadata is used by format-independent processing nodes within the network to adapt the bitstream temporally, spatially or otherwise to meet network/terminal constraints. In order to minimize latency within the system, the metadata must be synchronized with the content—a process which varies depending on the content format. Consequently, if the adaptation framework is to remain format-independent as a whole, the process of mapping both the content and metadata onto the transport stream must also be format-independent.
- 3) *Generic multimedia syntax translation* [9]. As new content formats are developed, they do not become useful until infrastructure is deployed to allow the format to be processed and consumed. Additionally, the increasing diversity of content and metadata specifications makes it difficult for (particularly portable) devices to interoperate with the breadth of content available. A format-independent binding language can be used as a lightweight mechanism to allow content to be converted from one format into another so that it may be consumed on legacy devices. For example, the Music Player Multimedia Appli-

Fig. 1. Assembling content into Virtual containers can be thought of as another method of content delivery.

cation Format (Music Player MAF) [10] is a new content format standardized by MPEG which combines MPEG-21 and MPEG-7 metadata, JPEG images, and MP3 audio content in an ISO file format to provide an augmented digital music library. While existing devices—portable music players, mobile phones, PC software, and so on—may not support this format, a generic approach could be used to translate Music Player MAF content into representations that these devices understand. This makes it feasible for operators to begin to deploy content using new formats while still enabling it to be consumed by existing devices.

- 4) *Virtual container assembly* is a format-independent binding language may equally be used to describe the inverse process—the assembly of legacy content into new formats or containers such as MAFs or digital items. Fig. 1 depicts this process: legacy content (for example MP3 audio with ID3 metadata [11]) is processed by a generic processor according to a description file, in order to generate a rich content container.

B. Requirements

These application scenarios highlight several requirements for a binding language:

- 1) *Content format independence.* A binding language must allow description of how multimedia content—image, audio, video, and so on—in any format is to be fragmented and packetized for delivery.
- 2) *Metadata format independence.* Most recent multimedia metadata specifications are specified with an XML syntax—for example MPEG-7 [12], Dublin Core [13], and TVAnytime [4]. It is likely that future metadata standards will increasingly be described using RDF, however it too is most commonly serialized in XML [14]. Given that the rich content containers discussed previously also use an XML syntax [3], [4], native support for XML is clearly an important requirement. However, several widely used multimedia metadata technologies—(notably ID3 [11] and EXIF [15])—use a binary syntax. This means that a format-independent binding language will need to be able to efficiently deal with metadata in these and potentially other arbitrary binary structures.
- 3) *Delivery format independence.* On the internet, RTP predominates multimedia transport. However, the application domains of the rich media standards discussed are consid-

Fig. 2. Model for generic delivery of Rich Media Content.

erably wider than this. Digital Television,—for example MPEG-2 Transport Streams over DVB-S/T/C [16]—Mobile Multimedia (3GPP [17] and DVB-H [16]), as well as digital radio (DAB/DMB [18])—are all potential channels for rich content delivery. A binding language should provide means to map content onto these and other delivery channels, as well as provide means to extend the framework to new delivery channels as they are developed.

C. Model for Generic Delivery of Rich Media Content

Fig. 2 presents the model for multimedia content delivery which was developed for this work. It identifies the generic tasks associated with the streaming and delivery of multimedia content, independent of its format. As will be seen below (Section II-D), solutions for some parts of the model exist in the literature, and where appropriate they have been adopted into this work. However, there is a need for a solution which addresses the problem as a whole.

The task of multimedia delivery is considered as a series of smaller components: the extraction of fragments of one or more inputs, the combination these fragments into packets, and mapping of the packets onto output formats. Postprocessing may also be required to compress or otherwise modify the data.

1) *Binary Abstraction*: Many recent multimedia metadata standards are expressed as XML. In fact, the hierarchical structure which XML provides is conceptually similar to the syntactical structures used in binary formats (as demonstrated by the BSDL mapping between binary and XML; see Section II-D1). Consequently, via a binary abstraction layer (Fig. 2), XML tools can be used to manipulate binary data. This enables a single syntax for a format-independent binding language regardless of the format of the input data.

2) *Fragmentation*: The fragmentation process is required to identify each portion of the input data to be mapped into a separate output packet. Importantly, many fragmentation processes will involve very large numbers of similar pieces—for example the packets of a video or audio stream—making it highly inefficient to describe each fragment individually. In such cases, it is more desirable to describe the selection of the entire set of fragments, along with rules that determine how to divide the set into individual pieces. These rules may include a maximum fragment size or duration, a limit on the count of a particular element within a single fragment, or that particular substructures must remain unfragmented.

3) *Packetization*: Once fragments of input content have been identified, they are inserted into packets. This process is required

to identify the temporal and other parameters associated with each fragment, and also to provide any syntactical structures not present in the input data but required by the output format. Examples of such structures are XML or binary fields to structure metadata, or supplementary packet headers (e.g., aggregation mode for H.264 over RTP [19]).

4) *Postprocessing*: A postprocessing stage allows for custom manipulation of the output data, such as compression of XML, transcoding of media content, or encryption.

5) *Output Mapping*: Finally, a mechanism is required to identify the mapping of the packets of data into the output stream, in such a way that the temporal and other characteristics of each packet are satisfied.

D. Related Work

While there are no existing tools for generic content delivery as a whole, there are a number of approaches within the literature that address various parts of the model, including tools for binary abstraction, approaches to XML fragmentation, and various output mappings. These are presented below.

1) *Binary Abstraction*: Flavor [2], the Bitstream Syntax Description Language (BSDL) [3], and a recent combination of the two —XFlavor+ [20] all provide format-independent languages to describe the structure of multimedia bitstreams. In particular, BSDL is designed as an extension of XML Schema [21], allowing a single schema to elegantly provide a description of a binary stream, its XML representation, and the mapping between the two. While not originally designed for this purpose, a bitstream syntax schema directly provides an abstraction layer to allow binary content to be treated as if it were XML (Fig. 2).

2) *XML Fragmentation*: The authors of BSDL propose that one of the XML stylesheet languages be used to transform the output of the BSDL process to effect bitstream adaptation.[3]. These tools are potential candidates for the description of fragmentation within the model. However, in practice, fragmentation simply involves selection of parts of an XML document, rather than arbitrary transformation. Stylesheets use the XML Path Language (XPath) [22] for this purpose, and consequently this was identified as the most suitable tool for use within the model.

3) *Output Mapping—Metadata*: Recent literature contains a number of approaches for streaming metadata: Ransburg *et al.* propose an RTP payload format for carrying adaptation metadata [8]. Annodex [23] provides a mechanism to insert simple metadata within a particular content format for annotation, indexing and hyperlinking. Giradot [24], Niedermeier [25] and

Wong [26] also propose various formats for streaming XML metadata.

These approaches all provide specific streaming formats. They may potentially be the output of a format-independent delivery process; that is, a binding language operates at a more abstract level: it is designed to allow metadata to be mapped into any desired format, as required by the specific application.

III. IMPLEMENTING THE MODEL: BBL

This section provides a detailed overview of the BBL and its use. Instead of presenting a specification that is precise but difficult to understand, a detailed example will be used to highlight the most relevant aspects of the language.² Additionally, while it is the language and its processing model that are the most pertinent aspects of this section, part A will also discuss the architecture of the prototypical BBL server that has been developed, since this significantly aids understanding. Sections III-B to III-G discuss specific aspects of the language.

The example corresponds to the application scenario described in Section II-A, where a rich content container is to be delivered via multiple channels. More specifically, a broadcaster wishes to deliver a Digital TV Program defined by an MPEG-21 Digital Item (DI) that references the audiovisual content, and contains TVAnytime program metadata, usage license information, and MPEG-7 metadata. The DI is described using the Digital Item Declaration Language (DIDL), and the various metadata are to be streamed with different periods of repetition to facilitate random access [3]. Some of the audience for the TV program have a DVB-based MPEG-2 Transport Stream (TS) channel to the broadcaster, while others have an IP-based channel, and so will use RTP.

Instructions for fragmenting the DIDL and its metadata are shared between the two channels. These instructions are declared in a BBL file (shown in Fig. 3) that is included by the individual files for both channels. Fig. 3 will be discussed in more detail in Sections III-B to III-G.

Delivery of the video is handled by an external system for the MPEG-2 TS part of the scenario, utilizing the large base of existing infrastructure in the broadcast TV domain. In such a scenario, BBL provides only the metadata, and the *output mapping* component essentially acts as a multiplexer, adding the metadata sections to the existing video stream.

Conversely, the BBL processor is responsible for both media- and meta-data in the RTP scenario. Here, BBL can be thought of as a module that is added to a streaming server, to enable it to stream data in any format. As a result, additional BBL instructions are necessary to describe how the media resource is to be streamed. These instructions are shown in Fig. 4, and make use of a BSD schema (Fig. 5) to identify the structure of the binary media data.

While in this case (as well as numerous others) delivery of MPEG-2 via RTP is already specified by a standard [7] and supported by most available streaming servers, this is not universally so. BBL provides an alternate method for content delivery that does not require a lengthy standardization process,

```
<BBL>
<Binding xmlSource="TVProgram.xml" timeScheme="frac(1000)">
  <Register>
    <Encoder id="BiM"
      encoderURI="http://whisper.elec.uow.edu.au/mpeg/xml/BiM">
      <bim:schema>TVAProgramInformation.xsd</bim:schema>
    </Encoder>
  </Register>
  <PacketStream deliveryTime="1" repeat="400" handler="meta">
    <ContentTemplate>
      <Include ref="/did:DIDL" depth="-1">
        <Fragmentation attributes="id">
          <FragmentAt deliveryTime="100" repeat="400" id="TVAnytime"
            match="/did:Container/did:Item/did:Descriptor[2]/did:Statement/
            dmprc:DMPInformation/dmprc:StructuredData/dmprm:TVAMain"/>
          <FragmentAt deliveryTime="500" repeat="200" id="licenseInfo"
            match="/did:Container/did:Item/did:Descriptor[1]/did:Statement/
            ipmp:IPMPGeneralInfoDescriptor/ipmp:LicenseCollection"/>
          <FragmentAt timeScheme="mp7t" repeat="5000" id="MP7Seg"
            match="/did:Container/did:Item/did:Item[2]/did:Descriptor[2]/
            did:Statement/dmprc:DMPInformation/dmprc:StructuredData/
            mp7:Mpeg7/mp7:Description/mp7:MultimediaCollection/
            mp7:TemporalDecomposition/mp7:AudioVisualSegment"
            deliveryTime="/mp7:MediaTime/mp7:MediaTimePoint"/>
        </Fragmentation>
      </Include>
    </ContentTemplate>
  </PacketStream>
  <Packet xmlSource="?/did:DIDL/did:Container/did:Item/
    did:Descriptor[2]/did:Statement/@ref?">
    <Content>
      <did:Descriptor>
        <did:Statement>
          <Include ref="/" depth="-1"/>
        </did:Statement>
      </did:Descriptor>
    </Content>
    <Encode encoding="BiM"/>
  </Packet>
</Binding>
</BBL>
```

Fig. 3. BBL Instructions for TV Program fragmentation.

or server retooling, in order to support new content types. Additionally, BBL provides a very flexible way to customize content and metadata delivery for the requirements of the channel(s), user(s), and the content itself.

For other examples of delivering multimedia content with BBL, see [9] and [27].

A. BBL Server Architecture

On receiving a request to deliver BBL-described content, the server (Fig. 6) parses the delivery instructions into a BBL object tree that coordinates the subsequent delivery process. BBL Objects make calls to other modules within the server in order to effect the delivery. These modules include the following.

- Binary to BSD and BSD to binary parsers, which provide the binary abstraction function (see section C).
- XPath processor (in this case Saxon 8B). Apart from the ref attribute of `<Include>` (see section D), XPath expressions are executed within the context of the current packet. This limited context allows a standard object-based XPath processor to be used efficiently.
- Streaming XPath processor for `<Include>`, to avoid the memory overhead associated with standard XPath processing on large documents. This is an extension to Saxon

²For the specification, see [5].

```

<BBL xmlns:mpg2="urn:mpeg2:systems:bitstream">
  <Binding timeScheme="frac(90000)">                                <!-- (a) -->
    <Variables>
      <Define name="time" value="0"/>
    </Variables>
    <Register>                                                        <!-- (b) -->
      <Handler handlerURI="http://whisper.elec.uow.edu.au/
        mpg2/mpeg21/dis/handler/RTP">
        <rtp.timeBase>90000</rtp.timeBase>
        <rtp.payloadType>33</rtp.payloadType>
        <rtp.port>1235</rtp.port>
      </Handler>
      <BSD namespace="urn:mpeg2:systems:bitstream"
        bsSchemaLocation="MPEG2Sys.xsd" fromBinarySource="false"/>
    </Register>                                                        <!-- (c) -->
    <PacketStream deliveryTime="0">
      <HandlerParams>
        <rtp.marker>?if (time = 0) then 1 else 0?</rtp.marker>
      </HandlerParams>
      <ContentTemplate>
        <Include ref="/mpg2:Bitstream/mpg2:Pack" depth="-1">        <!-- (d) -->
          <Timing>
            <DeliveryTimes match="." value="$time"/>                <!-- (e) -->
          </Timing>
          <Fragmentation>
            <Count value="1" match="."/>                            <!-- (f) -->
          </Fragmentation>
        </Include>
      </ContentTemplate>
      <Variables>                                                    <!-- (g) -->
        <Assign name="time" value="./mpg2:SCR0 * 1073741824 +
          ./mpg2:SCR1 * 32768 + ./mpg2:SCR2 + ./mpg2:SCR3 div 300"/>
      </Variables>
    </PacketStream>
  </Binding>
</BBL>

```

Fig. 4. BBL instructions for MPEG-2 video fragmentation.

```

<xsd:schema bs2:rootElement="bitstream">
  <!-- ... -->
  <xsd:element name="Pack" bs2:ifNext="000001BA">                  <!-- (a) -->
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PackSC" type="xsd:int"/>
        <xsd:element name="null" type="bs1:b2" fixed="1"/>
        <xsd:element name="SCR0" type="bs1:b3"/>
        <xsd:element ref="mkr"/>
        <xsd:element name="SCR1" type="bs1:b15"/>
        <xsd:element ref="mkr"/>
        <xsd:element name="SCR2" type="bs1:b15"/>
        <xsd:element ref="mkr"/>
        <xsd:element name="SCR3" type="bs1:b9"/>
        <xsd:element ref="mkr"/>
        <xsd:element name="ProgramMuxRate" type="bs1:b22"/>
        <xsd:element name="null" type="bs1:b7"/>
        <xsd:element name="StuffingLength" type="bs1:b3"/>
      <xsd:element name="PackStuffing" type="xsd:byte" minOccurs="0"
        maxOccurs="unbounded" bs2:nOccurs="mp2:StuffingLength"/>
      <xsd:element ref="SystemHeader" minOccurs="0"/>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="AudioPES" type="PESType"
          bs2:ifNext="000001C0 000001DF"/>
        <xsd:element name="VideoPES" type="PESType"
          bs2:ifNext="000001E0 000001EF"/>
        <xsd:element name="OtherPacket" type="miscPacket"
          bs2:ifNext="000001BC 000001FF"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

Fig. 5. Part of BSD schema for Mpeg-2 system streams.

Fig. 6. System architecture of prototype BBL-based server.

which evaluates individual node-tests against elements as they are parsed, caching matching elements for later retrieval.

- ECMAScript compiler and runtime, which provides an extension mechanism for more complex delivery requirements. A detailed discussion of the use of ECMAScript within BBL is beyond the scope of this paper.
- Zero or more encoder modules that provide postprocessing for packets (see F).
- Handler(s) that are responsible for placing packets on the designated delivery channel(s) (see G). Often, only a single channel (hence handler) is used. However, sometimes it is desirable to deliver the content on multiple channels. For example, parts of a DI could be sent on a reliable channel, and others on a channel that minimizes delay.

A BBL session operates in two separate phases: setup and execution. In the setup phase, the main BBL instruction file is parsed, along with subordinate BBL files referenced by the main file, and any BSD schemas referenced by the BBL files. The parser output is the object tree, which then calls the remaining setup operations. ECMAScript functions and XPath expressions are compiled, the various processor modules are initialized, and encoders and handlers instantiated.

When this is complete, the server enters the execution phase. Here, `<Packet>` and `<PacketStream>` tasks reserve a thread from a pool, then go to sleep until the appointed delivery time. At the correct time, the delivery instructions are executed (see C to G) and the thread either returns to the pool (for a `<Packet>`) or goes back to sleep until the next delivery time.

While execution occupies almost all of the time and resources of the BBL server, setup, and configuration are nonetheless important, and are discussed in Section III-B.

B. Configuration

The configuration of a BBL session is specified by the `<Register>` element and its descendants: `<Encoder>`, `<Handler>`, `<Function>`, and `<BSD>` (see Figs. 3(a) and 4(b)). These elements register the encoders, handlers, ECMAScript functions and BSD Schemata, respectively, used in the file.

Encoders and handlers may have additional descendant elements that are used to parameterize the encoder/handler instance. For example, the handler at Fig. 4(b) has several child

elements with the prefix `rtp` that set up the time-base and session description (SDP). The namespace and schema for these parameters are defined as part of the encoder/handler specification, as is the `uri` used to identify the module.

The `<BSD>` element (4c) identifies that data in a particular namespace is binary rather than XML. The attribute `bsSchemaLocation` locates the schema for data in this namespace, which is used to parse input binary data, and/or, generate output binary data. See (C) below for further details.

`xmlSource` (3a) and `binarySource` attributes in the BBL instructions declare XML or binary source documents (respectively), which provide data to a BBL session. Source documents are accessed via `<Include>` elements (and several other means, see [5]). Multiple sources may be declared within the document, to allow for the distributed nature of some multimedia collections, and a hierarchical mechanism is used to dereference these source documents, similar to that used by XML Namespaces [28]. For any element, the in-scope XML or binary source document is that referenced by the first `xmlSource` or `binarySource` attribute (respectively) encountered by searching the ancestor-or-self axis of the element [22], in order of decreasing proximity to the element (that is, first the element itself, then its parent, then its grandparent, and so on to the root node).

Parameter content and source attributes may contain literal values, or alternatively may contain an XPath expression that resolves to the desired literal value. In the latter case, the expression is delimited by “?”. For example, in Fig. 3, the main XML source document `TVProgram.xml` (a) contains a reference to an external file that has the `TVAnytime` data. The URL for this file is contained in the specified `ref` attribute, and is used to locate the data for the `<Packet>` at (e).

Finally, a `timeScheme` attribute (3d and 4a) specifies the syntax that is used to specify temporal data within the BBL instructions. Multiple time schemes may be used in a document, and the scheme in force at any given element is determined in the same manner as the in-scope source documents. Various time schemes are provided to simplify interoperability between BBL and commonly used metadata time formats—such as the MPEG-7 time scheme, NPT, and the time codes defined by SMPTE [5]. Time values in binary formats typically signal time using an integer field that increments with a defined frequency. The fractional time scheme is provided to cater for this case, where the frequency is specified by the integer in parentheses.

C. Binary Abstraction

The Bitstream Syntax Description Language (BSDL) [3] acts as an abstraction layer to allow binary resources to be handled in BBL in the same way as XML. A BSDL Schema (which is an XML Schema with additional information) describes all resources of a particular format. For example, the BSDL Schema from which Fig. 5 is taken describes any MPEG-2 Systems bitstream [29], such as the Program Streams typically stored in files with the extension “`mpg`”. For further discussion of BSDL, the reader is directed to [3].

The `Pack` structure (Fig. 5(a)) is the atomic temporal unit of an MPEG-2 Program Stream, and a `Pack`’s time stamp is given by the System Clock Reference (SCR) field. To avoid start code

TABLE I
FRAGMENTATION RULES

<code><Size></code>	– Declares that content should be fragmented so that its size in bytes (when serialized and/or encoded) is no greater than the given <code>value</code> .
<code><Duration></code>	– Specifies that content should be fragmented so that the total duration of elements in any one packet is no greater than the given <code>value</code> .
<code><Count></code>	– Specifies that content should be fragmented so that the number of nodes matched by the <code>match</code> expression is no greater than the given <code>value</code> .
<code><Constraint></code>	– Applies a constraint to the fragmentation of content elements which are <code>matched</code> by an XPath expression. Possible constraints are <ul style="list-style-type: none"> a) Matched nodes must be the first content in any packet; b) Matched nodes must be the last content in any packet; c) The subtree of matched nodes must be <code>unbroken</code> (not fragmented into separate packets).
<code><FragmentAt></code>	– Declares that a new fragment is to be created at any node <code>matched</code> by an XPath expression. The specific node(s) matched are present in both parent and child fragments (optionally with <code>fragmentRef</code> and <code>fragmentID</code> attributes to assist reconstruction). Any included descendants of the matched node(s) are present in the child fragment.

emulation this field is split into parts, so it must be reassembled to read the field value. This is done in the variable `$time` in the BBL instructions [Fig. 4(g)].

While BSDL is an extension of XML Schema, this does not mean that binary resources must be transformed into XML in order to be delivered. A BSDL Schema is simultaneously a binary schema *and* an XML schema, and as a result it can be used to refer to binary data *as if it were XML*.

D. Fragmentation

Fragmentation is implemented within BBL by the `<Include>` element. The `include` element selects one or more nodes from a source document via an XPath expression (on the `ref` attribute). These node(s), along with a number of levels of descendants (specified by the value of `depth`), are included in the output in place of the `<Include>` element.

If the `include` element is part of a `PacketStream` declaration (see E), then the included content can be further fragmented with the rules in Table I. In Fig. 3(c), the `include` element retrieves the root `DIDL` node, and its descendants to any depth (−1). This content is then fragmented by `FragmentAt`, so that the usage license details are repeated within the stream more frequently than other metadata, and so that metadata relevant to specific points in the program (MPEG-7 Segments) are not delivered until that time. In Fig. 4(d), `<Include>` is used to select all of

the Packs in the program stream, which are subsequently split into a single Pack per output packet.

E. Packetization

There are two mechanisms to define packetization within BBL. The first—the `<Packet>` element—specifies how single packets are assembled. Its use is described in Section III-E1. Secondly, a sequence of similar or related packets may be defined more efficiently using the `<PacketStream>` element (Section III-E2).

1) *Single Packet*: Content is declared using the following.

- a) Elements from namespaces other than BBL's may be directly instantiated as descendants of the Content element. This includes elements from BSD namespaces. This is the case for example with `<did:Descriptor>` and `<did:Statement>` at (3f).
- b) Elements which are instantiated in other XML documents may be retrieved by an `<Include>` element and included anywhere in the content (3c).
- c) Binary data may be retrieved with an `<Include>` with an XPath reference (4d) according to the XML abstraction presented by the BS Schema (5a).
- d) The result of variable computation may be included in packet content via `<Value-of>` (not shown).

2) *Packet Definition as a Stream*: Involves specifying the entire set of content to be delivered, along with one or more rules to govern the fragmentation of the content. Content is specified using the same mechanisms as single packets (above), while fragmentation rules are declared using the elements in Table I. Fig. 3(d) shows an example of the `FragmentAt` rule, and Fig. 4(f) of `Count`.

The delivery time of each packet within the stream must also be identified. This information is typically embedded within packet data or elsewhere in the source content, and is variously specified as an absolute timestamp, a temporal offset between adjacent packets, or in some cases indirectly via the hierarchical structure of the bitstream.³ Each of these scenarios is accommodated by `<PacketStream>`, using either `<DeliveryTimes>` or `<Durations>`. Fig. 4(e) for example uses the former to specify the time extracted from the SCR.

F. Postprocessing

Postprocessing is specified with the `<Encode>` element. This references an encoder registered in the same BBL file or one of the files importing it, and can be a child of a `Packet(Stream)` or any element in the packet content. In Fig. 3(g), the TVAnytime metadata is compressed using BiM [25].

G. Output Mapping

Each `Packet(Stream)` is output to the first registered handler by default (this is the case at Fig. 4(b)). Alternatively, a `Packet(Stream)` may reference a different handler via its `id`, using a `handler` attribute, as shown in Fig. 3(b). In the example, the handler with `id = "meta"` is not declared in the file shown;

³This is for example the case with H.264/AVC bitstreams, which contain no internal temporal information [19]. The bitstream is structured into NAL units, one or more of which make up individual video frames, and the frame rate must be specified by information external to the H.264 stream.

TABLE II
ON-THE-FLY BBL PROCESSING, PERFORMANCE RESULTS

Channel	Length (m:ss)	CPU Time		Mem. (Mb)	
		(s)	(%)	Avg.	Max.
(a) MPEG-2 TS	7:44	5.34	1.15%	1.42	1.74
(b) RTP	7:44	32.48	7.00%	3.89	5.45

it is declared in the importing file, so that the handler type can change between the scenarios (MPEG-2TS versus RTP).

IV. EVALUATING THE BITSTREAM BINDING LANGUAGE

Using the examples discussed in the preceding section, we now present a performance evaluation of the prototype BBL Server (Table II). The server was implemented in Java and tested on a Pentium M 1.85 GHz PC, with 1 GB of RAM, running Windows XP and the Sun 1.6.0 JVM. Each test was performed ten times, and the results averaged. Note that we are not evaluating network performance, since bytes on-the-wire are, in this case,⁴ identical to that which would be output from an existing streaming server. Instead, this section compares the performance of the *generic* BBL approach with the *format-specific* traditional streaming server. All of this happens on the server, not on the wire.

In the case of the MPEG-2 Transport Stream, the BBL server manages only the metadata; the multimedia content is processed by a format-specific multiplexer. In contrast, BBL is responsible for processing the entire RTP session, requiring significantly more CPU time to do so. This highlights the tradeoff inherent in providing a generic solution that can adapt to a wide range of content and metadata formats, versus the approach typical of current streaming servers, where modules are purpose-built and optimized for each format. Being a generic tool, the BBL server will not scale to the hundreds or thousands of simultaneous sessions typically observed with a conventional multimedia streaming server. While the implementation may benefit significantly from optimization, software written specifically for individual formats will, in general, always outperform generic software.

In a broadcast scenario, such as is typically the case with an MPEG-2 Transport Stream, this is not a critical issue since a single stream is delivered to a large audience. However, for RTP and other delivery channels, streaming infrastructure is required to handle a large number of simultaneous sessions, and hence for BBL to be viable, a mechanism is needed to allow it to scale. We have identified two solutions to this issue.

- 1) *Pre-processing of content* may be performed, where BBL processing is conducted offline, and the process output placed in a file format which is easily streamed by existing servers. In this way, the format-independent BBL processing may be performed by an offline application, on a separate device. Because this offline process has no need to handle multiple simultaneous sessions, scalability is no longer a critical issue. Online processing is performed by existing streaming servers using very scalable *hint-tracks*.

⁴It is certainly possible to use BBL to define new on-the-wire delivery formats, and this is one of the central uses for the language. However, the network performance of such a format is determined not by BBL, but by the design of the format itself.

This approach is demonstrated in [27], where the BBL processor uses a handler that is able to create hinted ISO or QuickTime files. These are essentially generic multimedia containers with hint-tracks, and are supported by most multimedia streaming servers to facilitate highly scalable multi-user streaming.

Hints are essentially byte-copy instructions, and so are processed very efficiently by the streaming server, removing the requirement that the server itself possess detailed knowledge about individual formats. The application of hint tracks to the BBL-based streaming architecture provides a complete format-independent, scalable streaming architecture. This is achieved by replacing individual format-specific modules within the hinter with a BBL processor, which is then driven by BBL instructions for hinting the multimedia content.

- 2) *Pre-compilation of BBL Instructions* is another alternative for improving the scalability of BBL processing. This option acknowledges that a single BBL description may apply to many pieces of content, and so it may be desirable to compile the BBL instructions into an efficient intermediate format (analogous to Java byte-code) ahead of time. For example, one of the computationally expensive operations within the BBL processor is the use of BSDL as a binary abstraction. Rather than explicitly generating the XML representation of binary data, against which XPath expressions are evaluated, it is possible to compile such expressions into an algorithm that operates directly on the binary data, bypassing the XML representation entirely. This task is future work.

V. CONCLUSION

This paper has proposed a Bitstream Binding Language (BBL), which provides a flexible and format-independent mechanism for delivery of rich multimedia Content. The central contribution of BBL is a framework for the description of mappings between a virtual container and a delivery channel, which may be a streaming format such as RTP, or a static format (an ISO/QuickTime file, for example). BBL has been adopted by MPEG as part of the MPEG-21 Multimedia Framework, and can be used to describe such mappings for any multimedia content format, enabling the delivery of format-agnostic virtual containers, and facilitating interoperability with new media types as they are developed.

A BBL Processor has been developed, and its use demonstrated for MPEG-21 Digital Item delivery over MPEG-2 Transport Streams and RTP.

In addition to providing a mapping for content into one or more output formats, BBL may be used to customize the presentation of content according to the requirements of the user, terminal and/or delivery channel. For example, BBL could be used to insert Supplemental Enhancement Information [19] into an H.264/AVC stream, for those channels/terminals which can make use of the extra information. Further, with a scalable bitstream, BBL could be configured so that it delivers different layers to different clients, depending on the parameters of each.

In the light of Reconfigurable Video Coding [1], the flexibility provided by BBL becomes even more significant. In ef-

fect, this represents a paradigm shift away from highly rigid bitstream formats, to an approach that allows the bitstream to be completely restructured according to the requirements of the content, channel or terminal. Reconfigurable Video Coding provides a decoder that allows new video formats to be developed without extensive standardization and application development efforts. BBL brings similar ease of extensibility to a multimedia streaming and delivery architecture.

ACKNOWLEDGMENT

The authors would like to thank G. Drury for extensive help with editing of documentation and for programming assistance. They would also like to thank MPEG participants and the anonymous reviewers for evaluation and constructive criticism that led to many improvements in the paper.

REFERENCES

- [1] *Working Draft 3 of 23001-4: Codec Configuration Representation* ISO/IEC, 2007 [Online]. Available: <http://www.iso.org>
- [2] D. Hong and A. Eleftheriadis, "XFlavor: Bridging bits and objects in media representation," in *Proc. IEEE Multimedia Expo*, 2002, 2002.
- [3] I. Burnett, Ed., *The MPEG-21 Book*. New York: Wiley, 2006.
- [4] R. M. Tol and E. A. Montie, "TV anytime: STORit on myTV," in *Proc. Int. Conf. Consumer Electronics*, 2000.
- [5] *21000-18, IT—Multimedia framework (MPEG-21)—Part 18: Digital Item Streaming* ISO/IEC, 2007 [Online]. Available: <http://www.iso.org/>
- [6] H. Schulzrinne *et al.*, *RFC3550: RTP: A Transport Protocol for Real-Time Applications* 2003 [Online]. Available: <http://www.ietf.org/rfc/rfc3550.txt>
- [7] D. Hoffman *et al.*, *RFC2250: RTP Payload Format for MPEG1/2 Video* 1998 [Online]. Available: <http://www.ietf.org/rfc/rfc2250.txt?number=2250>
- [8] M. Ransburg *et al.*, "Transport mechanisms for metadata-driven distributed multimedia adaptation," in *Proc. 1st Int. Conf. Multimedia Services Access Networks (MSAN)*, 2005.
- [9] J. Thomas-Kerr *et al.*, "Enhancing interoperability via generic multimedia syntax translation," in *Proc. 2nd Int. Conf. Automated Production of Cross Media Content for Multi-Channel Distribution*, 2006.
- [10] K. Diepold *et al.*, "MPEG-A: Multimedia application formats," *IEEE Multimedia*, vol. 12, no. 4, pp. 34–34, 2005.
- [11] M. Nilsson, *ID3 Tag Version 2.4.0—Main Structure* 2000 [Online]. Available: <http://www.id3.org/id3v2.4.0-structure.txt>
- [12] B. S. Manjunath *et al.*, *Introduction to MPEG-7: Multimedia Content Description Interface*. Chichester, U.K.: Wiley, 2002.
- [13] P. Johnston and A. Powell, *Expressing Dublin Core Metadata Using XML* 2006 [Online]. Available: <http://dublincore.org/documents/2006/05/29/dc-xml/>
- [14] D. Beckett, *RDF/XML Syntax Specification (Revised)* 2004 [Online]. Available: <http://www.w3.org/TR/rdf-syntax-grammar/>
- [15] *Exchangeable Image File Format for Digital Still Cameras: Exif Version 2.2* JEITA, 2002 [Online]. Available: <http://it.jeita.or.jp/document/publica/standard/exif/english/Exif.zip>
- [16] U. H. Reimers, "DVB-the family of international standards for digital video broadcasting," *Proc. IEEE*, vol. 94, no. 1, pp. 173–182, Jan. 2006.
- [17] T. Schierl *et al.*, *3GPP Compliant Adaptive Wireless Video Streaming Using H.264/AVC* 2005.
- [18] M. Grube *et al.*, "Applications of MPEG-4: Digital multimedia broadcasting," *IEEE Trans. Consumer Electron.*, vol. 47, no. 3, pp. 474–484, 2001.
- [19] S. Wenger *et al.*, *RFC3984: RTP Payload Format for H.264 Video* 2005 [Online]. Available: <http://www.ietf.org/rfc/rfc3984.txt>
- [20] W. De Neve *et al.*, "Using bitstream structure descriptions for the exploitation of multi-layered temporal scalability in H.264/AVC's base specification," *Lecture Notes Comput. Sci.*, pp. 641–652, 2005.
- [21] H. S. Thompson *et al.*, *XML Schema Part 1: Structures* 2004, <http://www.w3.org/TR/xmlschema-1>.
- [22] A. Berglund *et al.*, *XML Path Language (XPath) 2.0* 2007 [Online]. Available: <http://www.w3.org/TR/xpath20>
- [23] S. Pfeiffer *et al.*, "AnnodeX: A simple architecture to enable hyper-linking, search & retrieval of time-continuous data on the web," in *Proc. 5th ACM SIGMM Int. Workshop on Multimedia Information Retrieval*, 2003.

- [24] M. Girardot and N. Sundaresan, "Efficient representation and streaming of XML content over the internet medium," in *Proc. IEEE Conf. Multimedia and Expo (ICME)*, 2000.
- [25] U. Niedermeier *et al.*, "An MPEG-7 tool for compression and streaming of XML data," in *Proc. IEEE Int. Conf. Multimedia and Expo*, 2002.
- [26] E. Y. C. Wong *et al.*, "Semantic-based approach to streaming XML contents using Xstream," in *Proc. 27th Annu. Int. Conf. Computer Software and Applications (COMPSAC)*, 2003.
- [27] J. Thomas-Kerr *et al.*, "Generic, scalable multimedia streaming and delivery with example application for H.264/AVC," in *Proc. Advances in Multimedia Information Processing*, 2006, pp. 349–356.
- [28] T. Bray *et al.*, *Namespaces in XML* 1999 [Online]. Available: <http://www.w3.org/TR/REC-XML-names>
- [29] *ISO/IEC, 13818-1:2000 Amendment 1: Carriage of Metadata Over ITU-T Rec H.222* 2002, ISO/IEC 13818-1 streams, ISO/IEC JTC1/SC29/WG11 N5059.

Joseph Thomas-Kerr (S'04) received the B.E. degree in electrical engineering in 2003 from the University of Wollongong, NSW, Australia, where he is currently pursuing the Ph.D. degree, supported in part by the Smart Internet Technology CRC.

His current research interests include format-independent systems for multimedia streaming and delivery, reconfigurable multimedia processing, and multimedia semantics.

Ian Burnett (M'87–SM'02) received the B.Sc., M.Eng., and Ph.D. degrees in electrical and electronic engineering from the University of Bath, Bath, U.K.

He is an Associate Professor at the University of Wollongong, NSW, Australia. His current research interests are in multimedia processing and delivery, speech and audio coding, 3-D spatial audio, and audio separation.

Dr. Burnett has been an active participant in MPEG and MPEG-21 in recent years, notably as Chair of the Multimedia Description Schemes subgroup and Australian Head of Delegation.

Christian Ritz (M'00) received the B.S. degree in electrical engineering, the B.S. degree in mathematics, and the Ph.D. degree in electrical engineering from the University of Wollongong, NSW, Australia. His Ph.D. research focused on very-low-bit-rate coding of wideband speech.

He is currently a Lecturer at the University of Wollongong. His current research interests include wideband speech coding using waveform interpolation, blind source separation of mixed speech signals, transcoding of speech, and audio coding.