Format-independent Media Delivery, Applied to RTP, MP4, and Ogg

Davy Van Deursen, Wim Van Lancker, Pedro Debevere, and Rik Van de Walle

In Proceedings of the 4th International Conference on Multimedia and Ubiquitous Engineering, 6 pages on CD-rom, August, 2010, Cebu, Philippines.

FORMAT-INDEPENDENT MEDIA DELIVERY, APPLIED TO RTP, MP4, AND OGG

Davy Van Deursen, Wim Van Lancker, Pedro Debevere, and Rik Van de Walle

Ghent University – IBBT
Department of Electronics and Information Systems – Multimedia Lab
Gaston Crommenlaan 8, bus 201, 9050 Ledeberg-Ghent, Belgium
Email: {davy.vandeursen, wim.vanlancker, pedro.debevere, rik.vandewalle}@ugent.be

ABSTRACT

The current multimedia landscape is characterized by a significant heterogeneity in terms of coding and delivery formats, usage environments, and user preferences. This paper introduces a transparent multimedia content adaptation and delivery approach, i.e., model-driven content adaptation and delivery. It is based on a model that takes into account the structural metadata, semantic metadata, and scalability information of media bitstreams. Further, a format-independent multimedia packaging method is proposed based on this model for media bitstreams and MPEG-B BSDL. Thus, multimedia packaging is obtained by encapsulating the selected and adapted structural metadata within a specific delivery format. This packaging process is implemented using XML transformation filters and MPEG-B BSDL. To illustrate this format-independent packaging technique, we apply it to three packaging formats: RTP, MP4, and Ogg.

I. INTRODUCTION

The multimedia landscape is characterized by a growing amount of multimedia content and an increasing diversity in end-user devices that are able to consume multimedia. In order to provide (personalized) multimedia content anywhere, at anytime, and on any device, a transparent multimedia content adaptation and delivery approach is needed. In this context, metadata, which are generally defined as 'data about data', play a crucial role. Multimedia metadata enable the effective organization, access, and interpretation of multimedia content. Therefore, metadata have an increasingly important role in bringing order to the growing amount of available multimedia content. In this paper, we tackle the aforementioned problems by using format-independent content adaptation and delivery techniques. Furthermore, these techniques provide a seamless integration with today's manifold available multimedia metadata schemes.

To perform high-level adaptation operations (e.g., exploitation of scalability or scene selection), we have developed *model-driven content adaptation*. Its basic design is inspired by the principles of XML-driven content adaptation techniques, while its final design and the implementation thereof are based on Semantic Web technologies such as the Resource Description Framework (RDF), Web Ontology Language (OWL), and SPARQL Protocol And RDF Query Language (SPARQL). Semantic Web technologies are used to enhance the interoperability among the different metadata standards for multimedia content, thanks to their natural representation of objects and relationships. Furthermore, the adaptation algorithm is steered by a model for describing structural, semantic, and scalability information of media bitstreams. This model, implemented by making use of OWL, provides support for a seamless integration of the adaptation operations and semantic metadata. Therefore, it enables the definition of adaptation operations on a higher level (i.e., based on the model). Furthermore, when existing coding formats are mapped to this model, they can be adapted in a format-independent way.

A logical step after the adaptation of multimedia content is multimedia delivery. Multimedia content is usually not delivered as elementary bitstreams but packed in a particular delivery format. Today, a significant number of delivery or packaging formats exists; examples are MPEG-4 Part 14 (MP4 file format), Ogg, and Real-time Transport Protocol (RTP). Thus, we have to deal with different coding formats on the one hand, and different delivery formats on the other hand. The major contribution of this paper is the development of a format-independent multimedia packager, i.e., a generic software module that is independent of the incoming coding format and the outgoing delivery format. We investigate three packaging formats in more detail: RTP, Ogg, and MP4. For these three delivery formats, we illustrate how they are implemented within our format-independent multimedia packager.

II. MODEL-DRIVEN CONTENT ADAPTATION AND PACKAGING

In this section, we present a new media resource adaptation and packaging technique, which is inspired by the principles of XML-driven content adaptation techniques. Its design is based on Semantic Web technologies and a model for media bitstreams covering the structural, semantic, and scalability properties of these media bitstreams.

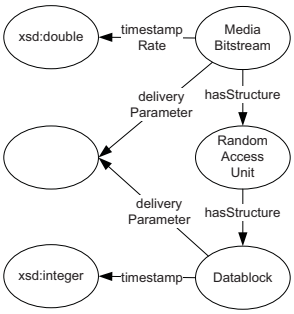

Fig. 1. Excerpt of the structural part of the model for media bitstreams. Ellipses and arrows represent OWL classes and properties respectively.

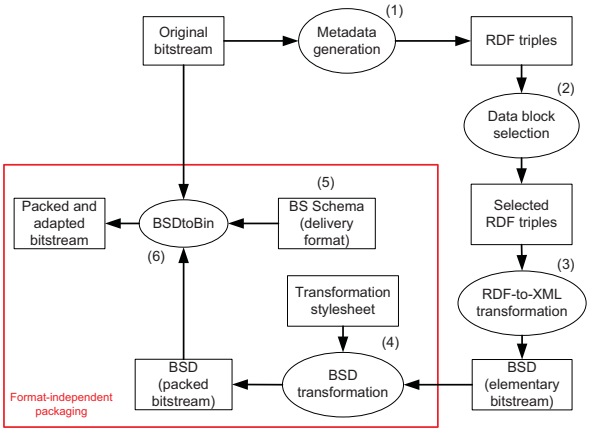

Fig. 2. The general workflow of model-driven content adaptation and packaging.

II-A. Model for Media Bitstreams

The model for media bitstreams provides support for a seamless integration of adaptation operations and semantic metadata. As such, it enables the definition of adaptation operations on a higher level (i.e., based on the model), on the condition that current and future coding formats can be mapped to this model. The model for media bitstreams is implemented as an OWL ontology. The instances of the model (i.e., the structural metadata) are expressed in RDF. The transformation of the structural metadata is implemented by using SPARQL queries, which are independent of the coding format. A visualization of an excerpt of the structural part of our model is given in Fig. 1.

The structural metadata part of the model describes information regarding the high-level structure of a compressed *MediaBitstream*. Such a *MediaBitstream* points to the physical location of the media bitstream by means of the *bitstreamSource* property. Also, delivery parameters (to assist in the packetization process) can be present by means of the *deliveryParameter* property. An example of a delivery parameter is the sampling frequency when the underlying coding format is AAC.

A *MediaBitstream* points to a list of *RandomAccessUnit*s by means of the *hasStructure* property. Random access refers to the ability of the decoder to start decoding at a point in a compressed media bitstream other than at the beginning and to recover an exact representation of the decoded bitstream.

Each random access unit points to a list of *DataBlock*s by means of the *hasStructure* property. A *DataBlock* points to a particular byte range of the compressed media bitstream. Further, each *DataBlock* has a *timestamp*, which represents a number related to the display time of the data block. In order to actually calculate the display time, the *timestampRate* property of the *MediaBitstream* class is used. The latter contains a number indicating the amount of timestamps that are contained in one second.

```
PREFIX mmo: <multimedia_model.owl#>
CONSTRUCT {
  # triples to describe a datablock:
  ?db rdf:type mmo:DataBlock.
  # ...
}
WHERE {
  ?bitstream rdf:type mmo:MediaBitstream.
  ?bitstream mmo:format 'video/H264'.
  ?bitstream mmo:hasStructure ?rau.
  ?rau mmo:hasStructure ?db.
  ?db rdf:type mmo:DataBlock.
  ?db mmo:timestamp ?ts.
  FILTER(?ts < 400)
}
```

Listing 1. SPARQL query selecting data blocks based on timestamp restrictions.

II-B. Workflow

The general workflow of model-driven content adaptation and packaging is depicted in Fig. 2. If media bitstreams need to be adapted and packaged with our proposed method, metadata instances compliant to our model need to be generated during the metadata generation step (1). The requested parts of the media streams are obtained during the data block selection step (2), where RDF graphs describing data blocks are queried using SPARQL. An example of such a query is shown in Listing 1. Based on the selected data blocks, a simple RDF-to-XML transformation is performed (3). The result of this transformation is an XML description of the selected datablocks, called a Bitstream Syntax Description (BSD). The latter can be used to create a packaged version of the adapted media bitstream. The classes and properties defined in our model, needed for the packaging process, are mapped to XML elements and attributes respectively.

The actual packaging process starts with the transformation of the BSD representing (part of) the elementary media bitstream (4). The resulting BSD represents an adapted and

```
<xsd:schema>
 <xsd:complexType name="RTP_header">
  <xsd:sequence>
   <xsd:element name="V" type="rtp:b2" fixed="2"/>
   <xsd:element name="P" type="rtp:b1"/>
   <xsd:element name="X" type="rtp:b1"/>
   <xsd:element name="CC" type="rtp:b4"/>
   <xsd:element name="M" type="rtp:b1"/>
   <xsd:element name="PT" type="rtp:b7"/>
   <xsd:element name="SN" type="xsd:unsignedShort"/>
   <xsd:element name="TS" type="xsd:unsignedInt"/>
   <xsd:element name="SSRC" type="xsd:unsignedInt"/>
   <!-- ... -->
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="RTP_packet">
  <xsd:sequence>
   <xsd:element name="rtp_header" type="rtp:RTP_header"/>
   <xsd:element name="rtp_payload" type="bs1:byteRange"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="RTP_stream">
  <xsd:complexType>
   <xsd:sequence maxOccurs="unbounded">
    <xsd:element name="rtp_packet" type="rtp:RTP_packet"/>
   </xsd:sequence>
   <xsd:attribute ref="bs1:bitstreamURI"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Fig. 3. BS Schema for RTP.

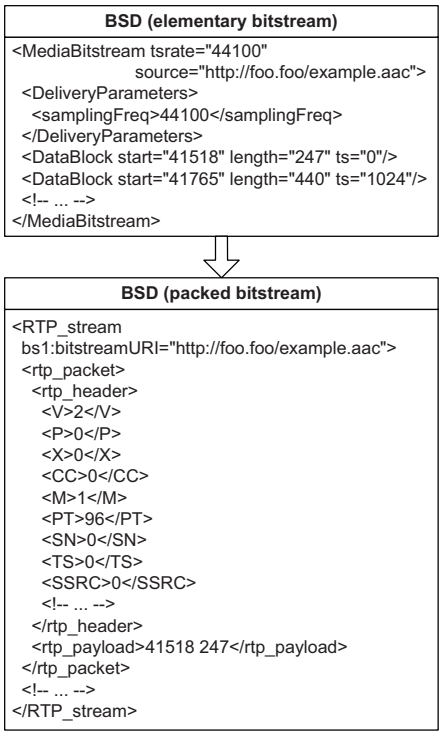

Fig. 4. XML-driven RTP packaging.

packaged media bitstream. The obtained BSD is compliant with MPEG-B BSDL [1], which implies that the BSDL framework can be used for further processing. The BSD transformation can be implemented using XSLT or STX, which enables the use of a format-independent transformation engine. Additionally, a Bitstream Syntax Schema (BS Schema, [1]) needs to be created, describing the high-level structures and syntax elements of the packaging format (5). Finally, the adapted and packaged media bitstream is created using BSDL's format-independent BSDtoBin parser [1], based on the BSD representing the adapted and packaged media bitstream, the BS Schema describing the delivery format, and the original media bitstream (6).

III. IMPLEMENTATION OF EXISTING DELIVERY FORMATS

In order to illustrate our proposed format-independent packaging technique, we apply it to three popular packaging formats: RTP, MP4, and Ogg. For each packaging format, we discuss the development of the BS Schema. Further, examples are provided regarding the BSD transformation from a BSD describing an elementary bitstream to a BSD describing a packed bitstream. Note that, for the sake of simplicity, our examples use AAC as underlying coding format for the RTP and MP4 packaging formats, and Ogg Vorbis for the Ogg packaging format.

III-A. Real-time Transport Protocol (RTP)

The Real-time Transport Protocol (RTP, [2]), formally known as RFC 3550, defines a standardized packet format for delivering audio and video over the Internet. More specifically, RTP provides end-to-end delivery services such as payload type identification, sequence numbering, timestamping, and delivery monitoring for data with real-time characteristics.

An excerpt of the BS Schema for the RTP format is depicted in Fig. 3. An RTP stream consists of a sequence of RTP packets, each containing an RTP header and payload data. RTP header components include: a sequence number (`SN`), which is used to detect lost packets; payload identification `PT`, which describes the specific media encoding so that it can be changed if it has to adapt to a variation in bandwidth; frame indication (`M`), which marks the beginning and end of each frame; source identification (`SSRC`), which identifies the originator of the frame; and intramedia synchronization (`TS`), which uses timestamps to detect different delay jitter within a single stream and compensate for it.

In Fig. 4, XML-driven RTP packaging is shown with an example XML instance describing a media resource compliant to our model introduced in Sect. II-A. Data blocks are packed in RTP packets. Note that, depending on the underlying coding format and the size of the data block, multiple data blocks can be part of one RTP packet or one

```
<xsd:schema>
 <xsd:complexType name="TTSBox">
  <xsd:complexContent>
   <xsd:extension base="mp4:FullBox">
    <xsd:sequence>
     <xsd:element name="entry_count" type="xsd:unsignedInt"/>
     <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="sampleCount" type="xsd:unsignedInt"/>
      <xsd:element name="sampleDelta" type="xsd:unsignedInt"/>
     </xsd:sequence>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:complexType name="SSBox">
  <xsd:complexContent>
   <xsd:extension base="mp4:FullBox">
    <xsd:sequence>
     <xsd:element name="sampleSize" type="xsd:unsignedInt"/>
     <xsd:element name="sampleCount" type="xsd:unsignedInt"/>
     <xsd:element name="entrySize" type="xsd:unsignedInt"
                  minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:complexType name="MediaDataBox">
  <xsd:complexContent>
   <xsd:extension base="mp4:Box">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
     <xsd:element name="data" type="mp4:bsdl_payload"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
 <xsd:element name="MP4_stream">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:choice maxOccurs="unbounded">
     <xsd:element name="ttsBox" type="mp4:TTSBox"/>
     <xsd:element name="ssBox" type="mp4:SSBox"/>
     <xsd:element name="mdatBox" type="mp4:MediaDataBox"/>
     <!-- ... -->
    </xsd:choice>
   </xsd:sequence>
   <xsd:attribute ref="bs1:bitstreamURI"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Fig. 5. BS Schema for MP4.

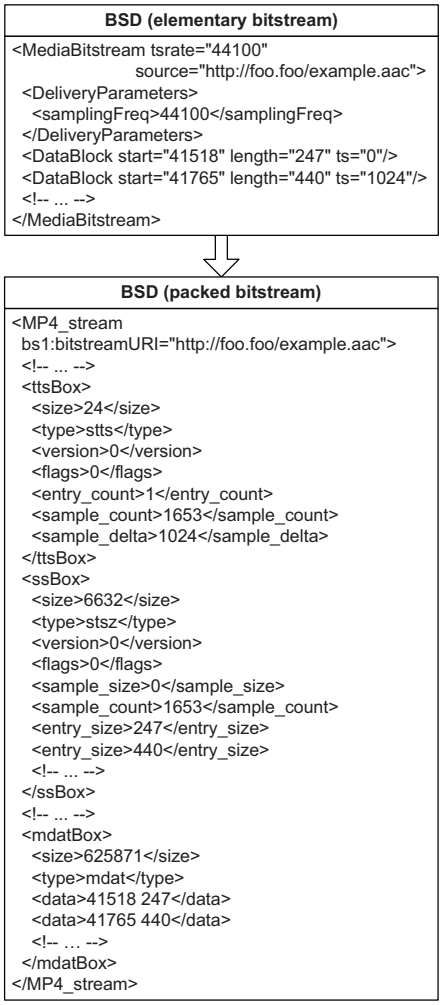

Fig. 6. XML-driven MP4 packaging.

data block can be spread over multiple RTP packets. In case one data block is encapsulated in one RTP packet, the byte range of the `rtp_payload` element corresponds to the byte range of the data block. Regarding the timing information, RTP timestamps (`TS` element) for the AAC coding format are calculated based on the timestamp information of data blocks and the underlying sampling frequency (i.e., 44100 Hz in this example).

III-B. MP4 File Format

MPEG-4 Part 14 or MP4 file format, formally known as ISO/IEC 14496-14:2003 [3], is a multimedia container format standard specified as a part of MPEG-4. It is able to store digital video and digital audio streams as well as other data such as subtitles and still images. The MP4 file format is built on top of the ISO Base Media File Format (known as MPEG-4 Part 12), which is in turn based on Apple's QuickTime file format.

An excerpt of the BS Schema for the MP4 file format is shown in Fig. 5. Files conforming to the MP4 file format are formed as a series of objects, called 'boxes'. All data is contained in boxes and there is no other data within the file. The 'box' is a building block defined by a unique type identifier and length. In Fig. 5, only the definition of three boxes are shown, due to space constraints. More specifically, the Time To Sample Box (`TTSBox`) is depicted, which describes a compact version of a table that allows indexing from decoding time to sample number. Each entry in the table gives the number of consecutive samples with the same time delta, and the delta of those samples. By adding the deltas a complete time-to-sample map may be built. Further, the Sample Size Box (`SSBox`) is shown, describing a table giving the size in bytes for each sample. Finally,

```
<xsd:schema>
 <xsd:complexType name="oggPage">
  <xsd:sequence>
   <xsd:element name="tag" type="ogv:tag"/>
   <xsd:element name="version" type="ogv:b8"/>
   <xsd:element name="header_type" type="ogv:b8"/>
   <xsd:element name="gran_pos" type="bs1:longLE"/>
   <xsd:element name="bit_serial_nr" type="bs1:intLE"/>
   <xsd:element name="page_seq_nr" type="bs1:intLE"/>
   <xsd:element name="crc_checksum" type="bs1:intLE"/>
   <xsd:element name="n_page_segments" type="ogv:b8"/>
   <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="segment_table_i" type="ogv:b8"/>
   </xsd:sequence>
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="packet" type="bs1:byteRange"/>
   </xsd:choice>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="Ogg_stream">
  <xsd:complexType>
   <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="OggPage" type="ogv:oggPage"/>
   </xsd:sequence>
   <xsd:attribute ref="bs1:bitstreamURI"/>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Fig. 7. BS Schema for Ogg.

the Media Data Box (`MediaDataBox`) describes the media data. For example, for video tracks, this box would contain the encoded video frames.

In Fig. 6, XML-driven MP4 packaging is illustrated. The Time To Sample Box makes use of the timestamps provided by the incoming data blocks to build up its table. Further, the Sample Size Box describes the size of each incoming data block. Note that in this example, each data block corresponds to one AAC frame (which consists of 1024 audio samples). Finally, the byte ranges described by the `data` elements located in the Media Data Box correspond to the byte ranges described by the incoming data blocks.

III-C. Ogg Container Format

The Ogg container format is a multimedia container format and the native file and stream format for the Xiph.org multimedia codecs [4]. It is an open format free for anyone to use. It is able to encapsulate compressed audio and video streams. Ogg is a stream-oriented container, meaning it can be written and read in one pass, making it a natural fit for Internet streaming and use in processing pipelines. Note that this stream orientation is the major design difference to other file-based container formats.

An excerpt of the BS Schema for the Ogg container format is shown in Fig. 7. Ogg provides packet framing (through `OggPages`), error detection (`crc_checksum`), and periodic timestamps for seeking (`gran_pos`). An Ogg stream is structured by dividing incoming packets into segments of up to 255 bytes and then wrapping a group of contiguous

BSD (elementary bitstream)

```
<MediaBitstream tsrate="44100"
                source="http://foo.foo/example.ogg">
 <!-- … -->
 <DeliveryParameters>
  <samplingFreq>44100</samplingFreq>
 </DeliveryParameters>
 <DataBlock start="23712" length="186" ts="2880"/>
 <DataBlock start="28034" length="200" ts="3904"/>
 <!-- ... -->
</MediaBitstream>
```

BSD (packed bitstream)

```
<Ogg_stream
 bs1:bitstreamURI="http://foo.foo/example.ogg">
 <!-- ... -->
 <oggPage>
  <tag>4F676753</tag>
  <version>0</version>
  <header_type>0</header_type>
  <gran_pos>3904</gran_pos>
  <bit_serial_nr>31415</bit_serial_nr>
  <page_seq_nr>2</page_seq_nr>
  <crc_checksum>933697086</crc_checksum>
  <n_page_segments>37</n_page_segments>
  <segment_table_i>186</segment_table_i>
  <segment_table_i>200</segment_table_i>
  <!-- … -->
  <packet>23712 186</packet>
  <packet>28034 200</packet>
  <!-- … -->
 </oggPage>
 <!-- ... -->
</Ogg_stream>
```

Fig. 8. XML-driven Ogg packaging.

packet segments into a variable length page preceded by a page header. Both the header size and page size are variable; the page header contains sizing information and checksum data to determine header/page size and data integrity. Thus, an Ogg container consists of a sequence of pages, in order, belonging to a single coding format instance.

In Fig. 8, XML-driven Ogg packaging is depicted. The incoming data blocks correspond to the packets contained in an Ogg page. Hence, the byte ranges described by the `packet` elements correspond to the byte ranges described by the incoming data blocks. Note that, when a data block has a larger size than 255 bytes, the data block is divided over multiple packets. With respect to the timing information, the syntax element $granulePosition$ (i.e., the XML element `gran_pos`) gives an indication of the timing of the current page. More specifically, in case of an Ogg page encapsulating an Ogg Vorbis stream, the granule position is a count of the number of raw audio samples from the beginning of the stream. Hence, the absolute time of a granule position is $granulePosition/samplingFrequency$. Thus, the value of the granule position is a combination of the timestamps of the incoming data blocks and the sampling frequency.

IV. RELATED WORK

A number of approaches regarding format-independent streaming have been proposed in the past. For instance, Digital Item Streaming (DIS, [5]), which is Part 18 of MPEG-21, enables the incremental delivery of a Digital Item (covering both metadata and media resources) in a piece-wise fashion. DIS relies on the Bitstream Binding Language (BBL) for this purpose. BBL defines syntax and semantics to describe instructions on how a Digital Item can be fragmented and mapped into one or more delivery channels. It uses the same principles for serializing the packed media bitstream as our proposed method, i.e., MPEG-B BSDL is used to abstract the media bitstream and to enable the use of format-agnostic software modules. However, the BBL approach requires a new language to be used to specify the fragmentation and packetization process. Our proposed method to perform format-independent packaging only requires knowledge of commonly used XML transformation languages such as XSLT or STX. Furthermore, our model for media bitstreams provides support for the multimedia packaging process (i.e., timestamp support and coding-format specific parameters). Hence, this information can already be calculated during the (offline) metadata generation step, which is in contrast to the BBL approach where this information needs to be calculated during the packaging process.

Ransburg *et al.* propose to use *Media Streaming Instructions* within BSDs to implement a generic streaming server [6]. More specifically, access units (i.e., the smallest unit of data to which timing may be attached) are identified and timestamps are assigned to them. Using Media Streaming Instructions, the fragmentation process and timestamp calculation is performed during the BSD generation step (i.e., during structural metadata generation). However, the fragmentation process is dependent on the delivery format (e.g., fragmentation of H.264/AVC streams is different for RTP and MP4 packetization). Also, BSDs including Media Streaming Instructions are processed by delivery-format specific software modules (e.g., an RTP packetizer).

Finally, the Darwin Streaming Server[1] (DSS) is an open source, cross-platform RTP/RTSP streaming server. It provides a coding-format agnostic design, i.e., no codecs are present in the server. The streaming of media resources is guided by hint tracks, which contain all the information necessary to packetize and stream the media resource. The creation of these hint tracks is coding-format specific. Note that the functionality of hint tracks is also present in our strucutural metadata (i.e., the mapping of timestamps to byte ranges of the media resource). However, support for adaptation operations is not available in DSS. Also, packing multimedia content with other delivery formats (other than RTP) is not possible.

[1] http://developer.apple.com/opensource/server/streaming/

V. CONCLUSIONS

The current multimedia landscape is characterized by a significant heterogeneity in terms of coding and delivery formats, usage environments, and user preferences. In order to provide (personalized) multimedia content anywhere, at anytime, and on any device, we proposed a transparent multimedia content adaptation and delivery approach. More specifically, we introduced model-driven content adaptation and delivery. The proposed approach relies on a model, implemented using Semantic Web technologies, that takes into account the structural metadata, semantic metadata, and scalability information of media bitstreams. Further, a format-independent multimedia packaging method (i.e., a generic software module that is independent of the incoming coding format and the outgoing delivery format) was proposed based on the model for media bitstreams and MPEG-B BSDL. To illustrate this format-independent packaging technique, we applied it to three packaging formats: RTP, Ogg, and MP4. More specifically, we provided excerpts of their BS Schemas and example instances within our presented workflow.

ACKNOWLEDGMENTS

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), and the European Union.

VI. REFERENCES

[1] ISO/IEC, "Information technology – MPEG systems technologies – Part 5: Bitstream Syntax Description Language," ISO/IEC 23001-5:2008, February 2008.

[2] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RFC 3550: "RTP: A Transport Protocol for Real-Time Applications," Available on http://www.ietf.org/rfc/rfc3550.txt.

[3] ISO/IEC, "Information technology – Coding of Audio, Picture, Multimedia and Hypermedia Information – Part 14: MP4 file format," ISO/IEC 14496-14:2003, December 2003.

[4] S. Pfeiffer, RFC 3533: "The Ogg Encapsulation Format Version 0," Available on http://www.ietf.org/rfc/rfc3533.txt.

[5] ISO/IEC, "Information technology – Multimedia framework (MPEG-21) – Part 18: Digital Item Streaming," ISO/IEC 21000-18:2007, June 2007.

[6] M. Ransburg, S. Devillers, C. Timmerer, and H. Hellwagner, "Processing and Delivery of Multimedia Metadata for Multimedia Content Streaming," in *Proceedings of 6th Workshop on Multimedia Semantics - The Role of Metadata*, Aachen, Germany, March 2007.