

MULTIMEDIA CONTENT ADAPTATION WITH XML

MYRIAM AMIELH, SYLVAIN DEVILLERS

Philips Research France (PRF)

51, rue Carnot – B.P. 301

92 156 Suresnes, France

E-Mail: {Myriam.Amielh, Sylvain.Devillers}@philips.com

On the one hand, new devices gaining access to the Internet need to obtain multimedia contents adapted to their specific capabilities but lack a generic framework for content negotiation. On the other hand, scalable formats provide some means to retrieve different versions of a single file by simple manipulations of the bitstream. In this paper, XML is used as a suitable framework for content negotiation and presentation by describing the structure of a bitstream. A new schema language based on XML-Schema is introduced to specify a document model. The resulting document is transformed via XSLT Style Sheets into a new XML document from which an adapted bitstream is generated. This method is generic and applicable to any scalable coding format. An application to the emerging JPEG2000 image coding standard is given.

1 Introduction

Web technologies such as W3C standards are more and more widely employed with the aim of exchanging and publishing documents over the web. These new standards provide complementary tools for every level of transaction on the Internet: device capabilities description, document description, content identification, etc. Especially, the XML specification issued in 1998 allows to describe documents untying their content and their presentation in order to dynamically generate a suitable presentation for the end-user device [1]. Today, this set of technologies is mainly used to adapt a text document or the structure of a composite multimedia document, but the content of the media bitstream itself is not adapted, though several emerging encoding formats provide a high level of scalability. For that purpose, we describe in this paper an extension of the web publishing framework to multimedia content by defining an XML representation of a multimedia bitstream. This original solution allows to fully exploit multimedia content scalability.

After an introduction to content adaptation and web publishing in section 2, we define the notion of scalability for multimedia coding in section 3. The section 4 of this document is dedicated to our original extension of the web publishing framework to multimedia content using an XML representation of a multimedia bitstream. In order to validate this new technology we apply the method to the emerging still image coding standard JPEG2000 in section 5. Then, in section 6 we demonstrate the need of a *document model* to define constraints on the XML bitstream representation, and we show how XML-schema can be a suitable solution.

Finally, sections 7 and 8 focus on the position of our technology in a complete system and show the scope of industrial applications.

2 Web Publishing Framework

This section gives a short overview of content adaptation for Web publishing purposes and identifies a level of content at which no solution has been deployed so far.

2.1 *Content adaptation on the web*

The problem of content adaptation is well-known for text documents on the web and is related to web publishing. The issue is to publish the same text document in different versions adapted to the capabilities of the rendering device. For this purpose, the structure of the document must be separated from its presentation: the source document is structured with XML and then dynamically processed to generate a presentation tailored to the available resources, *e.g.* respectively in HTML or WML for respectively a web or WAP browser. This processing may be performed by XSLT style sheets, the W3C language specifying XML-to-XML transformations [2]. The overall framework principle is described hereafter: answering a client request for a resource, the server first exchanges its capabilities with the client in a content negotiation stage to determine the adapted version to be published. It chooses the relevant XSLT transformation, applies it to the source XML document, and returns the resulting XML presentation.

As an example of implementation we can mention *Cocoon* developed by the Apache project to provide a complete separation between document content, style and logic into distinct XML files and uses XSLT capabilities to merge them. Based on a 100% pure Java publishing framework that relies on W3C technologies, the Cocoon model divides the development of web content into two separate levels: the XML files are created by content owners, through a normal text editor or an XML-aware tool. The created document is then rendered by applying an XSL style sheet which formats the specified resource type (HTML, WML, PDF, XHTML, etc).

2.2 *Transformation of a composite document structure*

Generic processes for adapting the presentation of a composite multimedia document have already been designed and successfully experimented. In order to take the multiplicity of user devices into account, Villard *and al.* created a general framework for structured document production through the specification of a document presentation model [3]. The presentation generation process is divided into three steps. The first step consists in encoding the content structure independently from its presentation, referring to a multimedia document model, in

which the document information is organized around four dimensions: MediaContent and MediaUse dimensions (to describe the logical structure of the document), temporal dimension (for synchronization between document parts), and spatial dimension (for layout management). The second step is a transformation producing a new presentation which reflects the device capabilities.

Then, a last transformation is performed to adapt the document to the user context, taking his/her profile (language, skill level, physical deficiencies, etc) into account.

2.3 About bitstream adaptation

Although content adaptation is a topic for which numerous activities are currently being carried out, the provided solutions always relate to text documents or composite documents. Up to now, no solution has been provided for the adaptation of the media bitstream itself.

3 Scalable Multimedia Coding

In this section we define and illustrate the notion of *scalability* and we conclude on the role of scalability for content adaptation strategies.

The term *scalable* refers to methods that allow the partial decoding or transmission of a single compressed bitstream. Depending on the system features (bit rate, errors, available resources), the decoder can take some portions of a stream and still decode its content (audio, video or still images) at different quality levels. Similarly, the server can send selected portions of the initial content.

If a bitstream is scalable, decoders from low to high performances can coexist. While low performance decoders may decode only small portions of the bitstream producing a basic quality, high performance decoders may decode much more data and produce a significantly higher quality.

There are several types of scalability. The most commonly implemented ones are *SNR scalability*, *temporal scalability*, and *spatial scalability*. However, other progressive encoding techniques, more application-oriented, do exist. In order to show the wide range of possible applications, we introduce hereafter several different categories of scalabilities.

3.1 Progressive encoding methods

Signal to Noise Ratio (SNR) scalability consists in compressing a quality-degraded version of an image at the same time than one or several complementary images to enhance its quality. When added back to the base image, the complementary images allow to progressively regenerate a higher quality version of the original image.

Many applications require the delivery of image data over different types of communication channels. Typical wireless communication channels give rise to random and burst bit errors. Internet communications are prone to packet losses due to traffic congestion. For that purpose, SNR scalability allows to provide some error resilience control: the main information can be conveyed using a secure channel and the complementary information can be transmitted in a less reliable way.

The *temporal scalability* allows to decode a bitstream at various temporal resolutions. Practically, the decoder can extract a temporally sub-sampled version of the initial video from the bitstream.

The *spatial scalability* offers scalability of the content size. The terminal can first render a small image and then progressively display several larger images until full resolution is reached. For instance, a bitstream can be organized so that a 64^2 image is encoded prior to the complementary information for the 128^2 and 256^2 images.

Considering small devices such as mobile phones or Personal Digital Assistants (PDAs), scalability can also be a matter of computing capabilities. The bitstream syntax can be organized taking the end-user device capabilities into account, such as the available memory of the decoder or the CPU power.

In an object-oriented video coding context, *object scalability* consists in controlling a scene by assigning a weight to each of its components. Practically, more or less bits are allocated to each object in the scene. This type of scalability offers numerous applications. For instance, in a limited bandwidth video-conference system, it can be more efficient to assign a high number of bits to the most significant object such as the speaker and to leave the residual bits for less relevant objects.

The *visual scalability in 3D scenes* allows to progressively decode a 3D realistic environment, taking the viewer position into account, so that only the relevant information is processed. This information therefore depends on both the geometry of the scene and on the viewpoint trajectory. Thus, the decoder can avoid the useless computation of object hidden sides. To significantly reduce the amount of transmitted content, visual scalability can as well be used at the encoder side in order to convey only the relevant information.

3.2 *Examples of scalable coding methods*

This subsection gives some examples of coding methods featuring scalable properties.

3.2.1 Temporal scalability in MPEG video coding

The MPEG video coding methods support several kinds of progressivity, especially temporal scalability. An MPEG video bitstream contains a succession of frames. An I-frame is encoded independently of any other frame, a P-frame is predicted (using motion compensation) from another previously decoded frame and a B-frame is bi-

directionally predicted from past and future frames (**Figure 1**). Thus, the decoder can skip some B-frames to provide a lower time resolution video. Otherwise, all the decoded frames can be used to provide the full temporal rate.

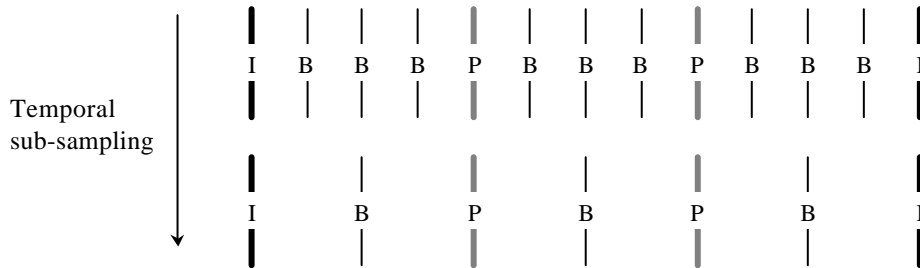


Figure 1. Temporal sub-sampling for MPEG Video.

3.2.2 MPEG-4 Fine Granular Scalability

The MPEG-4 specification for coding visual objects also provides the Fine Granularity Scalable profile (FGS) partly dedicated to SNR scalability encoding [4]. For that purpose, the bitstream is partitioned into several *layers* dedicated to a given level of SNR quality. The lower layer is referred to as the *base layer* and the higher layers are called *enhancement layers*. The FGS enhancement layer frames contain the difference between the base layer frames and the original image and are coded using *bit-plane variable length* encoding, which consists in encoding the most significant bits prior to the less significant bits and to operate a bit-plane shifting to eliminate the former.

3.2.3 A proprietary video codec

The 3D subband video codec proposed by Bottreau *and al.* [5] is based on a 3D wavelet transform, the third dimension being the time scale, and provides several progressive modes. Successive spatial, temporal and quality levels are indicated by *separators* in the bitstream. If the required spatial resolution, frame-rate or quality are inferior to the ones provided by the encoder, the decoder or server simply skips some parts of the bitstream.

3.3 Towards a generic bitstream editing tool

The preceding paragraphs showed how scalable coding formats allow to perform content adaptation without prior re-encoding. Only a bitstream edition tool is required to parse and cut the irrelevant bitstream parts off. However, a dedicated software component is required to operate a scalable delivery of the content. Up to know, it has been impossible to manage scalability without having any prior knowledge of standard or proprietary coding specifications. In the following

sections, we describe an original transparent way to perform scalable content adaptation using XML-based technologies.

4 Multimedia content adaptation with XML

In this section we detail a new solution for describing the content of a multimedia bitstream using XML technologies with the purpose of exploiting its scalability. Our approach is situated directly above the media element and is therefore complementary to other techniques dedicated to the adaptation of composite multimedia documents. In the terminology used below, a multimedia document is a single media unit (and not a composite media).

4.1 Overall mechanism

In this paper, we extend the publishing framework described in section 2 to multimedia contents by defining an XML representation of a multimedia bitstream. The overall mechanism, sketched in **Figure 2**, consists in the following three steps: In a first step, an XML representation is generated from a multimedia bitstream, describing its high-level structure.

Then, the XML representation is transformed by an XSLT style sheet which corresponds to the editing operations to be performed on the bitstream. Lastly, a new, adapted bitstream is generated back from the resulting XML document [6]. These three steps are detailed in next sections.

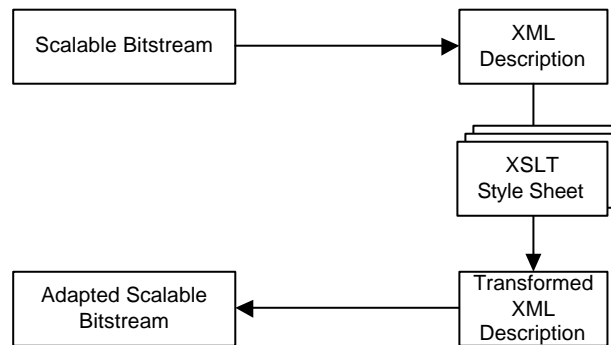


Figure 2. Multimedia Content Adaptation with XML

4.2 XML representation of a multimedia bitstream

In this section, we define an XML representation to describe the *high-level* structure of a multimedia bitstream. This representation is not meant to replace the original

binary format, but can be viewed as an additional layer describing its structure. In this respect, it is comparable to a metadata layer.

This representation may be itself scalable, *i.e.* it can describe the structure at a finer or coarser level of details depending on the application. However, it does not describe the structure on a bit-per-bit basis, which would be obviously too verbose, but on a segment basis.

A multimedia bitstream is generally organized as a sequence of *headers* and segments of data containing the encoded signal. In the following, we shall refer to these segments of data as *payloads*.

4.2.1 Headers

Headers usually contain the different parameters of the signal coding process. These parameters may be required to manipulate the bitstream, for instance to truncate it in a relevant way. This is the reason why they are written in the XML document in a readable format, *i.e.* a string of characters so that an XSLT style sheet may read and possibly modify their values. For example, if the header of an image coding format contains its width and its height, these parameters will typically appear in the XML representation as follows:

```
<size>
  <width>512</width>
  <height>512</height>
</size>
```

4.2.2 Payload

The payload contains the encoded signal and constitutes the major part of the bitstream. It is usually organized in packets of data. Its inner syntax is usually complex and compact: while headers mainly use symbols that are multiples of bytes, the payload should be decoded and interpreted on a bit per bit basis. Its compactness should therefore be preserved. Since XML can only contain “parsable”, *i.e.* text data, a specific solution is therefore required to include the payload which consists of pure binary data.

A first solution is to write the binary data with an hexadecimal notation. This solution is clearly very verbose and should be used only for small segments of binary data, typically a couple of bytes long.

A second solution is to use the base64 encoding algorithm, a method widely used in IETF protocols such as HTTP and SMTP, and defined in the MIME specification [7]. The principle is the following: three input bytes are coded on four bytes using six bits each ($3 \times 8 = 4 \times 6$). The six bits define a 64-character alphabet, containing the $26 \times 2 + 10$ alphanumeric characters (upper and lower case), “+”, “/”, and uses “=” for padding. This alphabet uses only printable character, excluding control characters (such as carriage return, line feed, tabulation or space) and is therefore very suitable for encoding binary data prior to including it into XML. The

cost of base64 encoding is a 33% increase in size compared to the initial binary format, which may be prohibitive in some applications. The following XML code gives an example of an element containing binary data encoded in base64.

```
<Packet>
  fgCf96yCKRpLU9p3BA08ZS65Vov3oLcomGwuKuWhSOVnyjRvqPCAtKgI
  DSaXh1g0ylh7gsZ==
</Packet>
```

A third solution is to use a Uniform Resource Identifier (URI) to point to an external entity containing pure binary data [8]. The fragment identifier of the URI can be used to indicate the relevant byte range in the original binary file, such as the example shown below.

```
<Packet>
  myFile.jp2#122-363
</Packet>
```

In this example, the local part of the URI (“myFile.jp2”) refers to the original binary file, and the fragment identifier (“122-363”) indicates the first and last byte offsets delimiting the relevant byte range in the file. However, the semantics of the URI fragment identifier is used here in a proprietary way. Other solutions may be possible by using XML related languages for linking and pointing to resources.

Note that the third solution differs from the first two in the fact that the binary data is not self-contained in the XML document, but is located in an external entity pointed to by the URI. This external entity will therefore be required along with the XML file to generate back a new bitstream from its XML representation.

4.3 *Transforming the bitstream XML representation*

We saw in section 3 that a scalable coding format allows to transform a content by simple editing operations. For example, if a JPEG2000 image is encoded in a progressive-by-resolution scheme, it is possible to render a smaller resolution image by cutting off relevant packets of data and modifying some header parameters related to the image size. Once we get an XML representation of the bitstream as described above, it is then possible to define the corresponding editing operations on the XML document, which will consist in removing the XML elements corresponding to the packets of data to be cut off, and modify some element or attribute values. It is then possible to define XSLT style sheets specifying the transformations to be applied on a particular coding format. In this way, we define an original method to manipulate a bitstream by editing its XML representation.

Note that the possible transformations are restricted to those that make sense for the coding format. This method does not allow for instance to transcode an MPEG-2 video into MPEG-4 since this requires some complex computations on the DCT coefficients.

4.4 *Generating a new bitstream from the transformed XML document*

The transformed XML representation describes the structure of a transformed bitstream that can be generated from this description along with the original binary file. To produce this bitstream, the XML description can be parsed and accessed via a standard interface such as a DOM (Document Object Model) [9].

The obtained bitstream is identical to the one that would have been obtained by performing the relevant editing operations directly on the input bitstream. The XML-to-binary transformation is therefore idempotent in the sense that we move to an XML representation to perform the editing operations with standard XML tools, and then move back to the original binary representation.

4.5 *Strengths of this approach*

To conclude, the mechanism we presented has the following properties:

- editing is done in the XML space, so through a human readable language,
- W3C standard languages such as XML and XSLT and related softwares are likely to be available on web servers,
- the specificity of a coding format is shifted from software to data which facilitates software maintenance (updates or new formats),
- new style sheets can be instantly applied on the bitstream description,
- this mechanism is generic enough to be applied to any encoding format and therefore a dedicated software is not required anymore for re-generating an adapted bitstream (see section 6).

5 Application to JPEG2000 Images

In this section we detail a practical application of the method described above to JPEG2000.

5.1 *Scalability in JPEG2000*

The emerging still image coding method JPEG2000 [10,11,12] is based on the wavelet transform, an inherently scalable method which allows progressive encoding driven by SNR quality, color component or resolution, *i.e.* image size. In the first case (progression by SNR quality), the bitstream is organized in layers, each layer carrying a quality increment. In the second case (progression by color component), data are organized by color components. For an input color image, the JPEG2000 encoder first transforms it into the luminance/chrominance space (YCrCb). By removing the last components, one therefore gets the luminance (*i.e.* gray-level) image. In the last case (progression by resolution), data are organized following the successive image size decompositions starting from the smallest one. For example, considering a 512^2 image, the bitstream first yields a 32^2 image, then a

64² image and so forth up to the full original resolution. All these image transformations can be achieved by editing the bitstream in a relevant way.

5.2 Codestream syntax

The JPEG2000 standard defines the following terminology:

- **tile**: a spatial subdivision of the image designed for independent coding,
- **bitstream**: the actual sequence of bits resulting from the encoding of a sequence of symbols,
- **codestream**: the main header followed by a collection of one or more tile-part headers and bitstreams, and concluded by the EOC (End of Codestream) marker. This is the image data in a compressed form with all of the signaling needed to decode it.

Main and tile-part headers are organized in *markers* and *marker segments*. A *marker* is a particular 2-byte value and a *marker segment* consists of a marker followed by two bytes indicating the marker segment length and the actual parameters. A detailed example of the SIZ marker characterizing image size will be shown in section 5.3.4.

The sequence of marker segments in the codestream is shown in **Figure 3**.

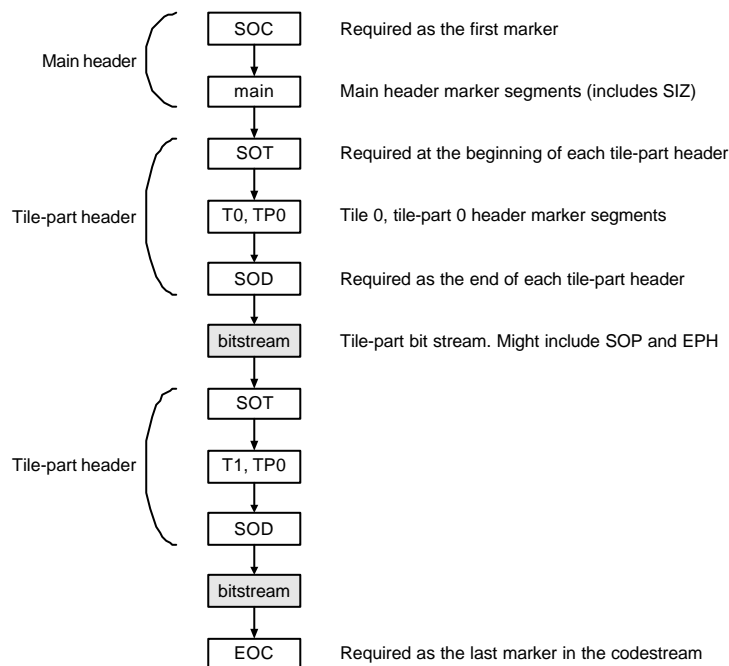


Figure 3. Construction of the JPEG2000 codestream

5.3 XML representation of a JPEG2000 codestream

5.3.1 Introduction

In this section, we describe how to structure the JPEG2000 codestream with XML. Establishing a correspondence between both representations is not straightforward nor unique since the codestream is structured as a *sequence* of segment markers, some of them ordered, while an XML document is a *tree*. We propose here one possible solution where we separate bitstreams and headers.

5.3.2 Bitstream

The JPEG2000 bitstream is segmented in *packets* of binary data. Transforming a scalable image will consist in removing a number of these packets. The packets are described in XML by using URIs pointing to the relevant byte ranges in the original file as seen in section 4.2.2.

5.3.3 Main and tile headers

The main and tile headers contain the different parameters of the image coding process, grouped in marker segments. Some of them include parameters required by the transformation which need to be written in a readable format such as the SIZ marker segment shown below. Other marker segments are not used by the transformation and may remain in a compact binary format without being further described. The binary segment may then be encoded in base64 or referred to with a URI as explained in section 4.2.2. We choose here to create an XML element for each marker segment with the following rules:

- the element is named after the three letter marker code,
- the marker length and other parameters contained in the marker segment are defined as sub-elements.

5.3.4 Example: SIZ marker segment

To illustrate this principle, we use the example of the SIZ (Image and Tile Size) marker segment as defined in the Appendix A of FDIS [12]. We define the corresponding fragment of Document Type Definition (DTD) listed in **Table 1**. An example of instantiation is given in **Table 2**.

Table 1. SIZ element DTD.

```
<!ELEMENT SIZ (Marker, Lsiz, Rsiz, Xsiz, Ysiz, XOsiz, YOsiz,
XTsiz, YTsiz,
          XTosiz, YTosiz, Csiz, Comp_siz+)>
<!ELEMENT Marker (#PCDATA)>
<!ELEMENT Lsiz (#PCDATA)>
<!ELEMENT Rsiz (#PCDATA)>
<!ELEMENT Xsiz (#PCDATA)>
```

```

<!ELEMENT Ysiz (#PCDATA)>
<!ELEMENT XOsiz (#PCDATA)>
<!ELEMENT YOsiz (#PCDATA)>
<!ELEMENT XTsiz (#PCDATA)>
<!ELEMENT YTtiz (#PCDATA)>
<!ELEMENT XTOsiz (#PCDATA)>
<!ELEMENT YTOsiz (#PCDATA)>
<!ELEMENT Csiz (#PCDATA)>
<!ELEMENT Comp_siz (Ssiz, XRsiz, YRsiz)>
<!ELEMENT Ssiz (sign, bitDepth)>
<!ELEMENT sign (#PCDATA)>
<!ELEMENT bitDepth (#PCDATA)>
<!ELEMENT XRsiz (#PCDATA)>
<!ELEMENT YRsiz (#PCDATA)>

```

Table 2. Instance of a SIZ element for a 3 component image.

```

<SIZ>
  <Marker>ff51</Marker> (1)
  <Lsiz>47</Lsiz>
  <Rsiz>0</Rsiz>
  <Xsiz>515</Xsiz>
  <Ysiz>512</Ysiz>
  <XOsiz>0</XOsiz>
  <YOsiz>0</YOsiz>
  <XTsiz>515</XTsiz>
  <YTtiz>512</YTtiz>
  <XTOsiz>0</XTOsiz>
  <YTOsiz>0</YTOsiz>
  <Csiz>3</Csiz>
  <Comp_siz> (2)
    <Ssiz>
      <sign>0</sign>
      <bitDepth>7</bitDepth>
    </Ssiz>
    <XRsiz>1</XRsiz>
    <YRsiz>1</YRsiz>
  </Comp_siz>
  <Comp_siz>
    <Ssiz>
      <sign>0</sign>
      <bitDepth>7</bitDepth>
    </Ssiz>
    <XRsiz>1</XRsiz>
    <YRsiz>1</YRsiz>
  </Comp_siz>
  <Comp_siz>
    <Ssiz>
      <sign>0</sign>
      <bitDepth>7</bitDepth>
    </Ssiz>
    <XRsiz>1</XRsiz>
    <YRsiz>1</YRsiz>
  </Comp_siz>
</SIZ>

```

In the previous example, we can highlight the following points:

- the `Marker` value (1) is written with an hexadecimal notation (0xFF51),
- `Comp_siz` element (2) is used to group the `Ssizi`, `XRsizi` and `YRsizi` parameters,
- types of parameters (integer, short, boolean...) cannot be specified with an XML DTD. They are defined as CDATA, *i.e.* string.

5.4 *JPEG2000 image transformation with XSLT*

Now that we have defined an XML description of a JPEG2000 image, we can use XSLT style sheets to define the transformations to adapt the image to the available resources. They consist in changing some attributes values and removing some elements. We have implemented three XSLT style sheets transforming the following images:

- image 1, progressively encoded with 3 color components,
- image 2, progressively encoded with 6 resolutions,
- image 3, progressively encoded with 4 SNR quality layers.

Each image is transformed in the following way:

- image 1 is transformed by a style sheet removing its last two color components to produce a gray-level image (*i.e.* with one luminance component),
- image 2 is transformed by removing its n last packets, where n is an input parameter, to produce 5 smaller resolutions from 256^2 down to the lowest resolution 16^2 ,
- image 3 is transformed by removing its n last quality layers to produce 3 progressively degraded images.

The following scenarios show an application of each progression type: a client with no color display may use progression by component to retrieve a gray-level version of a colored image. With progression by resolution, a client with a small display may download only the relevant size. Lastly, with progression by layer, when downloading an image, a client may first display it with a poor quality and then progressively improve the quality while further data is being downloaded.

6 Bitstream Syntax Definition Language

6.1 *Need for an XML document model*

In the method described in section 4, an XML representation is used for content adaptation. However, for the final application, the original binary format is still required, since it is the only one supported by the player. It is therefore necessary to generate back the original binary format, as explained in section 4.4. This transformation from XML to binary should be implicit so that no software dedicated

to this particular format is required. In other words, the XML representation should carry the information needed to generate a compliant bitstream.

However, we saw that some parameters need to be written in a readable format, *i.e.* a string of characters in order to be processed by XSLT. When included in XML, the information about its binary representation is then lost: for example, the element `<Ysiz>512</Ysiz>` does not specify whether the value should be binarized on four (*integer*) or two (*short integer*) bytes. Similarly, the element `<Marker>ff51</Marker>` does not specify that its value should be understood as an hexadecimal notation.

This information is given by the format specification and should be the same for any bitstream complying to this format. We therefore need a *document model* defining constraints on the XML representation, namely the datatypes of the element values. The following section explains existing methods to define a model of XML documents.

6.2 Introduction: XML document, schema, schema language

An *XML document* primarily consists of a strictly nested hierarchy of elements with a single root. Elements can contain character data, child elements, or a mixture of both. In addition, they can have attributes. **Table 3** gives an example of an XML document:

Table 3. Example of an XML document fragment.

```
<e1>
  <e2>Something</e2>
  <e3>Something else</e3>
</e1>
```

The author of an XML document may wish to define a set of constraints on this XML document, for example as follows: *The document should contain an element named “e1” containing two child elements “e2” and “e3”, themselves containing character data.*

This set of constraints is called a *document model* or a *schema*. The XML specification provides a grammar named Document Type Definition (DTD) to specify a document model. The constraints listed above are then specified as in **Table 4**:

Table 4. Example of a Document Type Definition.

```
<!ELEMENT e1 (e2, e3)>
<!ELEMENT e2 (#PCDATA)>
<!ELEMENT e3 (#PCDATA)>
```

The role of a *schema* is to *validate* an XML document, *i.e.* to check that it conforms to the given set of constraints. In other words, a document model or a schema delimits a set of conforming XML documents or *instances*.

The DTD provides a simple but limited grammar. In particular, it does not define types. For example, it cannot make a difference between an integer and a float: both will be represented as “CDATA”, *i.e.* a string, as in the DTD shown in **Table 1**.

The W3C has recently issued a language named XML-Schema [13] to specify schemas in a much more elaborate way. Such a language is called a *schema language*. It provides rich functionality with object-oriented-like concepts and a large range of datatypes.

In our case, the constraints would be specified by the schema shown in **Table 5**.

Table 5. Example of a schema expressed with XML-Schema.

```
<xsd:element name="e1">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="e2" type="xsd:string"/>
      <xsd:element name="e3" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

To summarize this short introduction, a *schema language* such as XML-Schema is used to express *schemas* which define a set of constraints on XML documents or *instances*.

6.3 Use of a schema language to generate the bitstream

Thus, we chose to use a schema language to specify schemas defining models for XML representations of a particular coding format. This language defines *datatypes*, *i.e.* how the element values should be binarized, and *structures*, *i.e.* how the elements are organized.

Though flexible and powerful, the W3C XML-Schema language does not fulfill the requirements of our application. For instance, XML-Schema defines datatypes such as *integer* in the mathematical sense, which have no implicit binary coding scheme and thus cannot be used. On the other hand, bitstream formats often use symbols such as arrays of bits which are not handled by XML-Schema. It is therefore necessary to build a new schema language based on XML-Schema with a set of restrictions and extensions. We call this new language the Bitstream Syntax Definition Language (BSDL).

In this way, the schema written in this language and assigned to the XML representation allows to generate a compliant bitstream as pictured in **Figure 4**.

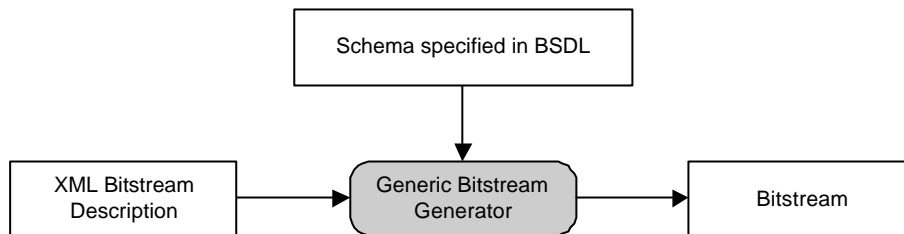


Figure 4. Generation of a bitstream from its XML description

6.4 Future work

In the previous sub-section, we introduced the need for a schema language for assigning datatypes to the XML representation and hence generating a bitstream. We are currently extending this language to allow a generic software agent to parse a bitstream from the schema describing the bitstream syntax as depicted in **Figure 5**.

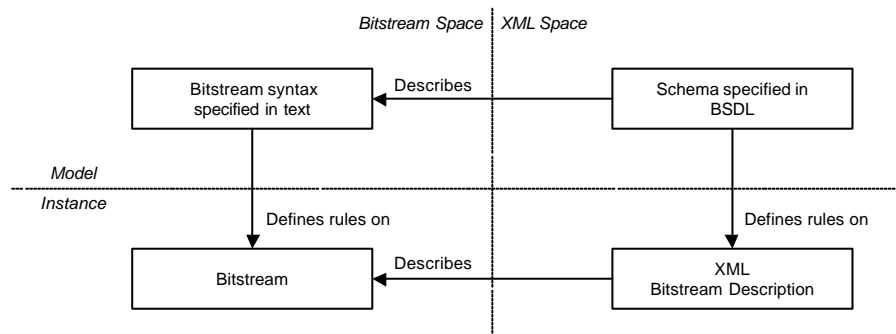


Figure 5. Principle of the Bitstream Syntax Definition Language.

7 System approach

The technology we describe in this paper takes place in the larger context of content adaptation for web publishing (Figure 6). When a client wants to download a document (1) the server sends a query (2) to get its capabilities and preferences in a profile description (3). Then, according to the received profile the requested document is processed (4) in order to dynamically generate a version adapted to the client profile and the resulting content is sent to the client (5).

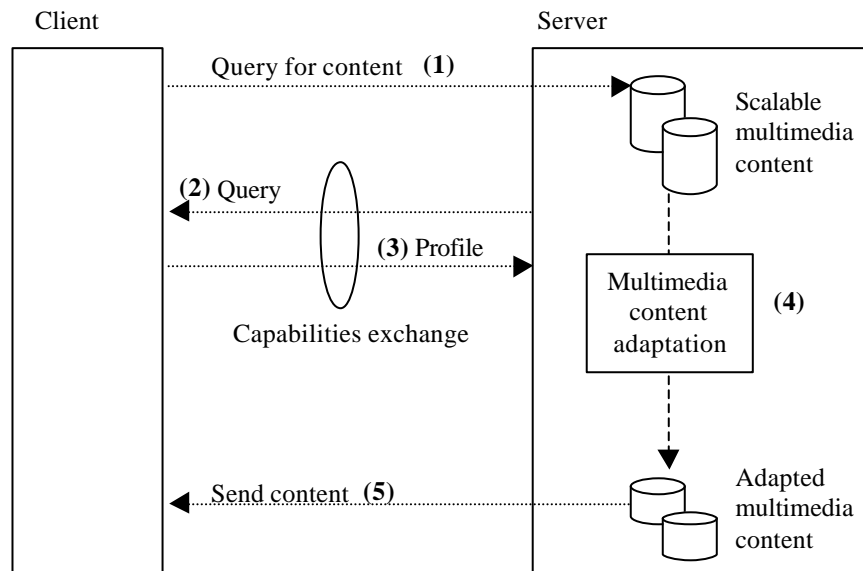


Figure 6. Integration of multimedia content adaptation in a complete system.

This section aims at identifying some of the existing solutions for other parts of the system which are complementary to our solution. Especially we briefly introduce the W3C solution CC/PP (Composite Capabilities/Preference Profiles) for describing devices capabilities, IETF Conneg and UAProf.

7.1 *CC/PP: a W3C solution for the description of device capabilities*

As the number and variety of devices connected to the Internet grows, there is a corresponding increase in the need to deliver content that is tailored to the capabilities of different devices. As part of a framework for content adaptation, a CC/PP profile is a description of device capabilities and user preferences that can be used to guide the edition of content presented to that device [14]. A CC/PP profile contains a number of attribute names and associated values that are used by a server to determine the most appropriate form of a resource to be delivered to a client.

7.2 *Conneg: content negotiation for HTTP*

The IETF formed a Content Negotiation Working Group (Conneg) to cover content negotiation inside and outside of HTTP [15]. HTTP allows web site authors to put multiple versions of the same information under a single resource URI. The Conneg

specification offers an extensible negotiation mechanism to automatically and efficiently retrieve the best version of an HTML document. For that purpose, when the resource is accessed, the user agent sends small *Accept-headers* which express user agent capabilities and user preferences. Then the origin server uses these *Accept-headers* to choose the best variant, which is returned in the response.

7.3 *UAProf: a WAP-oriented content negotiation solution*

The Wireless Application Protocols (WAP) standard is an adaptation of Web protocols and languages to the mobile phone applications. The User Agent Profile (UAProf) specification [16] extends WAP to enable the end-to-end flow of capability and preference information, between the WAP client, the intermediate network points and the origin server. It seeks to inter-operate seamlessly with CC/PP distribution over the Internet. The specification defines a set of components and attributes that WAP-devices may convey within the capability and preference information, such as hardware characteristics (screen size, color capabilities, manufacturer, etc), and network characteristics (latency, reliability, etc). As a request travels over the network from the client device to the origin server, each network element may optionally add additional profile information to the transmitted capabilities and preferences.

8 Industrial Applications

Beyond Internet applications, the multimedia content adaptation technology we present here can be straightforwardly considered for any industrial application dealing with devices producing, exchanging or receiving multimedia content.

In particular, this technology is relevant for advanced broadcasting chains involving Set-Top Boxes (STBs), high and low definition televisions and Personal Digital Assistants (PDAs). Indeed, the full resolution of a multimedia content such as a video could be stored on a STB and either directly displayed on a TV set or adapted to match the display capabilities of a PDA.

In the mobile phone domain, this technology can be exploited to download some content from a server (such as a Personal Computer, a Home Server or a Set-Top Box) while requesting an adapted version of the content to be delivered, taking the display performances and network bandwidth into account.

9 Conclusion

XML and related technologies have gained an overwhelming success for structuring, editing and exchanging electronic data. In this paper we showed how to extend the use of XML technologies to describe the syntax of a multimedia bitstream in order to dynamically exploit its scalability. As a validation, we gave an

example of a progressive JPEG2000 image for which we defined an XML representation and to which we applied an XSLT style sheet to fully exploit its scalable features without requiring a dedicated software agent. Moreover, the Bitstream Syntax Definition Language (BSDL) introduced in this paper provides a cross-standard and flexible approach to handle scalable content in order to adapt it to the available resources. The extension of XML technologies to multimedia bitstream modeling is an original and promising solution for advanced web publishing and allows numerous industrial applications, beyond the Internet framework.

References

1. (XML) Extensible Markup Language 1.0 (Second Edition), *W3C Recommendation*, October 6th, 2000.
2. XSL Transformations Version 1.0, *W3C Recommendation*, November 16th 1999.
3. L. Villard, C. Roisin and N. Layaida, An XML-based multimedia document processing model for content adaptation, *Digital Documents and Electronic Publishing DDEP00*, LNCS, Springer Verlag editions, September 2000.
4. ISO/IEC 14496-2/FDAM 4. Information technology - Coding of audio-visual objects - Part 2: Visual Amendment 4: Streaming video profile.
5. V. Bottreau, M. Bénetière, B. Felts and B. Pesquet, A fully scalable 3D subband video codec, ICIP'2001, *IEEE International Conference on Image Processing*, Thessaloniki, Greece, Oct. 7-10, 2001.
6. Sylvain Devillers, XML and XSLT Modeling for Multimedia Bitstream Manipulation, *10th International World Wide Web Conference*, Hong Kong, May 2-4, 2001.
7. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, IETF RFC 2045.
8. Uniform Resource Identifiers (URI): Generic Syntax, *W3C Draft Standard*, RFC 2396, August 1998.
9. Document Object Model (DOM) Level 1, Version 1.0, *W3C Recommendation*, Oct 1998.
10. Michael D. Adams et al., JPEG 2000: The Next Generation Still Image Compression Standard, Contribution to ISO/IEC SC29/WG01 wg1n1734.
11. Michael W. Marcellin and al, An Overview of JPEG-2000, *Proc. of IEEE Data Compression Conference*, pp. 523-541, 2000.
12. JPEG2000 Part I Final Draft International Standard. *ISO/IEC JTC1/SC29 WG1*.
13. XML-Schema Part 0: Primer, Part 1: Structures, Part 2: Datatypes, *W3C Recommendation*, May 2, 2001.
14. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies, *W3C Working Draft*, March 15, 2001.
15. Transparent Content Negotiation in HTTP, *IETF - RFC2295*, 1998.
16. Wireless Application Group, User Agent Profile Specification, *Wireless Application Protocol Forum*, November 10, 1999.