XFlavor: Providing XML Features in Media Representation

Danny Hong* and Alexandros Eleftheriadis†
Department of Electrical Engineering
Columbia University
New York, NY 10027, USA

April 22, 2004

Abstract

We present XFlavor, a framework for providing XML features in media representation. Using XFlavor, any media data can be converted back and forth between binary and XML representations. By converting a media bitstream into an XML document, we make the data easier to access and more flexible to use. Consequently, development of software tools for processing media data is greatly facilitated, as a generic XML parser can be used to read/write the data in XML form.

Keywords: Media Representation; XFlavor; Flavor; Software Tools

1 Introduction

With more and more media data becoming available, it is necessary to automatically process the data using software tools. For example, many tools are already being developed and utilized for filtering, indexing, searching, and so on. Additionally, with personal computers getting more powerful, the demand for software that processes coded media information will only increase. This includes not only encoders and decoders, but also applications that manipulate such information. Examples are editing tools, synthetic content creation tools, video annotation tools, and etc. Most of the research activities have been focused on coming up with efficient and effective algorithms/methods needed for the applications (e.g., how to get a better match when searching for an image), but little work has been done on how to efficiently build tools that actually process the data. There is the need to simplify and speed up development of software tools that process coded media information, and Flavor [1, 2, 3] has been introduced as a solution.

Flavor, which stands for Formal Language for Audio-Visual Object Representation, is a language created for describing any coded media bitstream in a formal way so that the code for reading and writing the bitstream can be automatically generated. For this, Flavor comes with a translator that automatically generates standard C++ or Java code from the Flavor description so that direct access to coded media information by application developers can be achieved with essentially zero programming. Flavor is also introduced in MPEG-4 as Syntactic Description Language [4].

Similar to Flavor, facilities that simplify and speed up development of software have been proven invaluable in almost every domain. For example, lex and yacc [5] help to write programs that transform structured textual input, such as translators and compilers. In such programs, two tasks that occur over and over are dividing the input into meaningful units (or tokens), and then

*danny@ee.columbia.edu
†eleft@ee.columbia.edu

discovering the relationship among the units. With lex and yacc, a formal way to describe the tokens and their relationship is provided. Then, from the description, the code that parses the input according to the specified grammar is automatically generated. After their first conception in 1970s, many variants and versions of lex and yacc (e.g., flex, bison, Abraxas PCYACC, etc.) have been introduced, proving the importance of the tools. Additionally, most of the modern translators and compilers are built using lex and yacc or similar tools.

In the field of telecommunications, we also have a wide variety of facilities that help to write Remote Procedure Call (RPC) [6, 7] applications. The rpcgen [7] code generator helps to write RPC applications by automatically generating network interface code. It also generates External Data Representation (XDR) [8, 9] routines so that local data structures are converted into network format and vice-versa for consistent exchange of information between communicating applications. Using the code generator, software developers need to focus only on the main features of their application. Similarly, Interface Definition Language (IDL) is used to specify server interfaces for open distributed processing, and an IDL compiler generates the necessary code (server and client stubs) that takes care of the marshalling and unmarshalling of the information to be transmitted and received via network, respectively. Other widely used facility in this area is the Abstract Syntax Notation 1 (ASN.1) [10, 11] standard.

In summary, the above-mentioned facilities come with a formal way to describe the structure of the data to be processed, and with their translator/compiler, they automatically generate the code that can be used for reading and writing the data (via file or network I/O).

XFlavor takes an alternate approach to simplifying and speeding up development of media-processing software tools. Rather than generating the necessary code for obtaining required values from bitstreams, XFlavor provides an XML [12] representation of media data so that the required values are easily obtainable. XFlavor consists of the Flavor translator (that has been enhanced to support XML features) and a software tool called `bitgen`. Using XFlavor, media data can be converted back and forth between the binary and XML representations. Once the data is in XML form, due to the self-describing nature of the XML document, the semantic values of the data (e.g., the width and height of an image) are directly available. As a result, applications can easily be developed using a generic XML parser to obtain the values. Additionally, XML related tools such as XSLT [13] can be used to easily manipulate the data.

The XML representation is not to replace the original binary representation, as both representations have advantages and disadvantages; the XML representation is introduced to complement the binary representation. In addition to simplifying development of software tools, the representation allows more flexible and extensible manipulations of the media data, where an XML-aware application can access any media data: one parser can access data in different formats. However, there are cases where processing the data in its native binary form is required. Memory-efficient, real-time applications will need to process the data in its bitstream level. We describe how both the XML and binary representations can be used interchangeably, taking advantage of different benefits offered by the two representations.

We first give a brief introduction of Flavor in Section 2, and in Section 3, XFlavor is described along with the benefits of using the XML representation. Closely related work is also introduced. Then, Section 4 describes a prototypic system showing a practical application of XFlavor. Section 5 discusses our new architecture of XFlavor that we are currently working on, and finally, we conclude with Section 6. More detailed information about Flavor and XFlavor can be found at `http://flavor.sourceforge.net`. The source code for the software is available as an open source project and the complete Flavor package, including XFlavor, can be downloaded from the Flavor web site.

2 Flavor

Building media-processing tools requires a universal task from software developers, which is to write code for mapping media information from bitstreams to program data structures. The problem with this is that the task represents a significant amount of the overall development effort. Due to the nature of the data, accessing media information is a nontrivial task. Such information is invariably coded in a highly efficient form to minimize the cost of storage and transmission. This source coding [14] operation is almost always performed in a bitstream-oriented fashion: the data to be represented is converted to a sequence of binary values of arbitrary and typically variable lengths, according to a specified syntax. The syntax itself can have various degrees of sophistication. One of the simplest forms is the GIF87a [15] format, consisting of essentially two headers and blocks of coded image data using the Lempel-Ziv-Welch compression. Much more complex formats include JPEG [16], MPEG-1 [17], 2 [18, 19] and 4 [20, 21], among others.

General-purpose programming languages such as C++ [22] and Java [23] do not provide native facilities for coping with such data. Software codec or application developers need to build their own facilities, involving two components. First, they need to develop software that deals with the bitstream-oriented nature of the data, as general-purpose microprocessors are strictly byte-oriented. Second, they need to implement parsing and generation code that complies with the syntax of the format at hand (be it proprietary or standard). Another problem is that these two tasks have to be duplicated by software developers who need to have access to the compressed representation within their application. Building the facilities represents a significant portion of the overall development effort, and additionally, hand-written code is susceptible to errors and it may not be fully optimized.

Flavor addresses these problems in an integrated way. First, it allows a "formal" description of the bitstream syntax. Formal here means that the description is based on a well-defined grammar, and as a result is amenable to software tool manipulation. A second and key aspect of Flavor's architecture is that the description has been designed as an extension of C++ and Java, in which the typing system incorporates bitstream representation semantics. This allows describing in a single place both the in-memory representation of data as well as its bitstream-level (compressed) representation. Both C++ and Java are heavily used object-oriented languages in media applications development, and as a result, seamless integration of Flavor description with both C++ and Java code and the overall architecture of an application is ensured.

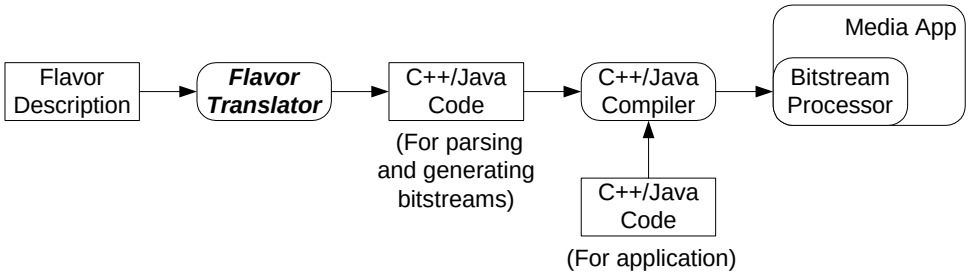

Figure 1: The Flavor description and translator can be used to create the bitstream processing part of a media application.

Flavor comes with a wide range of facilities to define sophisticated bitstreams, including `if-else`, `switch`, `for`, and `while` constructs. In contrast with regular C++ or Java, these are all included in the data declaration part of the class, so they are completely disassociated from code that belongs to class methods. This is in line with the declarative nature of Flavor, where the focus is on defining the structure of the data, not operations on them. Additionally, Flavor comes with a

translator that generates C++ or Java code from a Flavor description, and the code includes methods that can read or write data that complies to the described syntax. Figure 1 shows the steps taken to create a media-processing application using Flavor.

3 XFlavor

The major component of XFlavor is the Flavor translator that has been enhanced to support two additional features. First, in addition to the bitstream reading and writing code, the new translator can also produce the code for generating XML documents corresponding to the bitstreams described by Flavor. Second, from a Flavor description of a bitstream syntax, a corresponding XML schema can be created. Another component of XFlavor is a software tool called `bitgen`, which can be used for converting the XML representation of media data back into its native binary format.

The main goal of XFlavor is to offer an alternate form of the media data, enabling developers to write media-processing applications without having to know the bitstream syntax. None of the existing standards provide a tool that helps one to write such applications, with the exception of Flavor (see Section 2). XFlavor, however, takes a different approach from that of Flavor. Rather than generating the code for processing bitstreams, bitstreams are converted into XML documents for easier and more flexible processing. The XML representation complements the binary representation, and XFlavor enables to take advantage of both representations of media data.

In the following subsections, the three features of XFlavor are described in detail. Benefits as well as disadvantages of using such features are also discussed. Finally, we conclude this section by providing some information about work closely related to XFlavor.

3.1 Converting Bitstreams into XML Documents

One of the features of XFlavor is that it can be used to convert media data in binary format into a corresponding XML document. For example, given a Flavor description of the MPEG-2 Program Stream (PS) syntax, XFlavor can generate the necessary code for converting PS bitstreams into XML documents. Figure 2 shows the System header part of the syntax description. Using the generated code, an MPEG-2 PS stream can be converted into an XML document, and in Figure 3, the part of the document corresponding to the System header is depicted.

One advantage of using XML representation of media data is that the data is easier to access and manipulate. As shown in Figure 3, the XML document abstracts the bitstream layer and the semantic values of the data are directly available. As a result, media-processing software tools can be developed faster and easier, as any XML parser can be used to directly obtain the values needed for the application-specific processing. In the bitstream format, such values must be extracted via bit string manipulations, according to the specified syntax.

Another advantage is that software tools are provided with generic access to media data. For example, consider a search operation that uses DCT values to search for images and videos [24, 25, 26]. Then, dedicated software modules are needed to parse the bitstreams and obtain the DCT values. If the images and videos are coded in different standards, then we need as many bitstream parsers as there are supported formats. However, if XML were to be used to represent the data, then one search engine with an XML parser would be sufficient.

In addition, as the name indicates, XML documents are extensible. Extra information can be added to (or deleted from) a given document and a software tool can use both the new and old documents without having to change the parser. Simply, it is easier to extend (a given standard) by new information or fields, while still being compatible by software that does not understand these extensions. Given an XML document containing all the structures and information, applications can

```
class SystemHeader {
  unsigned int(32)   start_code;
  unsigned int(16)   header_length;
  bit(1)             marker = 0b1;
  unsigned int(22)   rate_bound;
  bit(1)             marker = 0b1;
  unsigned int(6)    audio_bound;
  bit(1)             fixed_flag;
  bit(1)             csps_flag;
  bit(1)             sys_aud_lock_flag;
  bit(1)             sys_vid_lock_flag;
  bit(1)             marker = 0b1;
  unsigned int(5)    vid_bound;
  bit(1)             pkt_rate_restr_flag;
  bit(7)             reserved = 0x7F;
  while (nextbits(1) == 0b1) {
    unsigned int(8)  stream_id;
    bit(2)           bit_pattern = 0b11;
    bit(1)           buff_bound_scale;
    unsigned int(13) buff_size_bound;
  }
}
```

Figure 2: The Flavor description of the MPEG-2 Program Stream System header.

```
<system_header>
  <start_code bitLen="32">443</start_code>
  <header_length bitLen="16">12</header_length>
  <marker bitLen="1">1</marker>
  <rate_bound bitLen="22">7653</rate_bound>
  <marker bitLen="1">1</marker>
  <audio_bound bitLen="6">1</audio_bound>
  <fixed_flag bitLen="1">1</fixed_flag>
  <csps_flag bitLen="1">0</csps_flag>
  <sys_aud_lock_flag bitLen="1">1</sys_aud_lock_flag>
  <sys_vid_lock_flag bitLen="1">1</sys_vid_lock_flag>
  <marker bitLen="1">1</marker>
  <vid_bound bitLen="5">1</vid_bound>
  <pkt_rate_restr_flag bitLen="1">1</pkt_rate_restr_flag>
  <reserved bitLen="7">127</reserved>
  <stream_id bitLen="8">224</stream_id>
  <bit_pattern bitLen="2">3</bit_pattern>
  <buff_bound_scale bitLen="1">1</buff_bound_scale>
  <buff_size_bound bitLen="13">80</buff_size_bound>
  <stream_id bitLen="8">192</stream_id>
  <bit_pattern bitLen="2">3</bit_pattern>
  <buff_bound_scale bitLen="1">0</buff_bound_scale>
  <buff_size_bound bitLen="13">80</buff_size_bound>
</system_header>
```

Figure 3: The XML representation of an instance of the MPEG-2 Program Stream System header.

simply access required elements while ignoring others. These extensions are very hard in a binary format, since they destroy the bit/byte offsets of the information, thus rendering any new content useless in non-flexible but nevertheless compliant applications.

Finally, given an XML document, all the XML related technologies and software tools can be directly used. One very important XML technology is XSLT [13]. XSLT is a language defined

using XML for transforming XML documents. With XML alone, interoperability is only obtained among the applications that understand a certain set of predefined tags. However, with XSLT, any application can be enabled to use any XML document as long as the actual data in the document is useful to the application. This "universal interoperability" is achieved by transforming XML documents into different structures usable to different applications.

Using such technology, a transcoder can be easily created by simply declaring a set of rules to transform the source elements. The stylesheet (containing XSLT rules) along with the source document can be fed into an XSLT processor to generate a new document with desired structure and information. A set of stylesheets can also be used to manipulate the elements of a given document. Similarly, simple text manipulating tools such as Perl can be used to easily manipulate the data in the XML form. Most binary media formats have internal structure, and exposing this in an XML form makes this structure human-readable and processable by generic XML tools or simple text manipulating tools.

Such an idea has recently been proposed in MPEG-21 [27] for easy and general manipulation of scalable content with the introduction of a bitstream syntax description language, BSDL [28, 29]. More information about BSDL is given in Subsection 3.4.

3.2 Generating Schemas from Flavor Descriptions

XFlavor can also be used to automatically generate a corresponding XML schema [30] from a Flavor description. An XML schema is used to define the syntax of XML documents by specifying how the elements are laid out in the documents as well as the attributes that are associated with the elements. It also sets the data types for the content of the elements. The schema can be thought of as a set of rules that applications must agree on in creating and manipulating data so that their output can be interchanged among them.

One benefit of using a schema is that data checking is automated. Most of the currently available XML parsers support schemas and they automatically check the "validity" of XML documents. The data checking process is a very important part of a media application, however, it is very tedious to implement. Using the schema saves much effort needed in developing an application, as software developers only need to work on the actual decoding part of the application.

```
<xsd:complexType name="SystemHeader">
  <xsd:sequence>
    <xsd:element name="start_code" type="flUInt"/>
    <xsd:element name="header_length" type="flUInt"/>
     ...
    ...
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="stream_id" type="flUInt"/>
      <xsd:element name="bit_pattern" type="flBit"/>
      <xsd:element name="buff_bound_scale" type="flBit"/>
      <xsd:element name="buff_size_bound" type="flUInt"/>
    </xsd:sequence>
  </xsd:sequence>
  <xsd:attribute name="aligned" type="xsd:unsignedInt" use="optional"/>
</xsd:complexType>
```

Figure 4: The XML schema corresponding to the Flavor description given in Figure 2.

Another benefit of using a schema is that it can be used as a guide in creating content. Given the structure of the data, a corresponding XML document can be easily created. Since each element that can appear in the XML document is specified in the schema, an appropriate user interface can

easily be derived with each field in the interface corresponding to each element. Creating or modifying an XML document according to the given schema ensures that the document is strictly conforming to the original bitstream specification, and the conforming XML document can be exactly converted into the original bitstream format. For this, XFlavor also comes with a software tool (`bitgen`) that uses XML schemas for converting XML documents into corresponding bitstreams. To realize such conversion, schemas must contain the bitstream syntax information. Figure 4 shows a sample schema generated from a Flavor description shown in Figure 2.

In Flavor, the actual media bitstream is represented using *parsable variables*. These variables are defined as a certain type (e.g., unsigned integer in big-endian byte-ordering) along with the number of bits used in the bitstream. For example, in Figure 2, the first variable `start_code` is a parsable variable of type unsigned integer (in big-endian byte-ordering by default), and in the bitstream it is represented with 32 bits. This type of variable, which represents a specific field in the bitstream syntax, can be directly represented in an XML document using an element as defined in the corresponding schema of Figure 4. The `start_code` element in the schema is declared to be of the type `flUInt`, which is one of the data types defined to correspond to all types of parsables variables. The data types used by the Flavor-generated XML schema are defined in a the `fltypes.xsd` file, and it is included in the Flavor package. For example, the definition of `flUInt` is given in Figure 5.

```
<xsd:complexType name="flUInt">
    <xsd:simpleContent>
        <xsd:extension base="xsd:unsignedInt">
            <xsd:attribute name="aligned" type="xsd:unsignedInt" use="optional"/>
            <xsd:attribute name="big" type="xsd:boolean" use="optional" default="true"/>
            <xsd:attribute name="bitLen" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:unsignedInt">
                        <xsd:maxInclusive value="32"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:extension>
    </xsd:simpleContent>
</xsd:complexType>
```

Figure 5: The definion of the `flUInt` type in XML.

Note that in addition to the type information, a set of attributes is defined to identify the type of the content of the element. Defining the element as `<xsd:element name="start_code" type="xsd:unsignedInt"/>` is not sufficient because through the XML parser, the content is available only as a string and a detailed type information is needed to convert the string into corresponding bit string.

The `bitLen` attribute is required for all data types as it determines the number of bits to use to represent the corresponding content. The `aligned` attribute is optionally used if the bit string of the corresponding content needs to be aligned. Additionally, the `flUInt` type may be accompanied with the `big` attribute to specify the byte-ordering (big or little-endian) to use to represent the content in bit string.

3.3 Compressing XML Documents

XML representation makes processing of data easier by adding extra information and having the data text based. However, this defeats one of the main purposes of media representation, which is to compress the data as small as possible. Hence, the XML representation, though it provides very

nice features for processing data, should not be used to replace the original binary representation. For storage and transmission, the binary format should be used to minimize the associated cost. Additionally, for fast and memory-efficient processing of media data, it is better to use the binary data than the expanded textual data.

To fix the above-mentioned problem associated with XML representation, many solutions have been proposed to compress XML data into a binary form. MPEG-7 [31] has already standardized binary encoding of its XML documents using BiM [32]. For wireless applications where resources are scarce, a binary version of XML is being deployed (WBXML [33]). There are also a number of general-purpose XML compression tools such as XMill [34], XMLZip [35] and Millau [36]. All these tools compress XML documents in their own binary format. However, in our case, we are dealing with data that already has its own binary form that is highly compressed. Taking this into account, we have developed a software tool, `bitgen`, as part of XFlavor, for converting the XML represented media data back into its native bitstream format.

Given a media data in XML form, `bitgen` simply discards all the tags and just encodes the content of the elements according to the given bitstream syntax, yielding a compressed data in its original binary format. This yields a better compression result than using any one of the above-mentioned XML compression tools because they incorporate the information contained in the tags when compressing XML documents, in order to maintain the structure. Also, we are dealing with an already compressed bitstream to begin with, and any additional compression on the original data would be very difficult.

To actually compare the effectiveness of the compression tools with XFlavor (`bitgen`), we define compression effectiveness (CE) as the size of the original binary data divided by the size of the compressed XML data. The larger the value of the CE, the more effective the compression tool is. Note that XFlavor will compress XML data so that the CE always equals to 1. Table 1 lists the CE of XMill, XMLZip, BiM, and XFlavor. Two GIF87a images (File 1 and File 2) are first converted into corresponding XML documents, and they are compressed using the XML compression tools. Millau was also examined; however, the encoder and decoder were not available for testing at the time of this writing. As shown in the table, all three compressors (XMill, XMLZip, and BiM) have CE less than 1. XMill produces the best result among the three because its sole purpose is to compress the data, whereas XMLZip and BiM produce much bigger compressed data because they add some useful features in the compressed domain. With XMill, the compressed XML data has to be decompressed in order to process the data. But, with XFlavor, the XML data can be converted into the original bitstream syntax and the resulting data can be processed using existing decoders that conform to the given bitstream specification. Additionally, the size of the original bitstream is always smaller than the same XML data compressed using XMill.

	File 1	**CE**	**File 2**	**CE**
Original File Size	*10,048*		*278,583*	
XML File Size	*305,634*		*8,200,278*	
XMill (gzip, CF=6)	14,348	0.70	362,173	0.77
XMill (gzip, CF=9)	14,311	0.70	357,335	0.79
XMill (bzip)	11,120	0.90	285,088	0.98
XMLZip	18,561	0.54	471,203	0.59
BiM	45,264	0.22	-	-
XFlavor	10,048	1.00	278,583	1.00

Table 1: Compression effectiveness (CE) for different XML compressors. CE = size of original data (bitstream) / size of compressed XML data.

In summary, if storage or transmission of data is the main concern, then the original bitstream should be used. Upon the time of processing, the bitstream can be converted into an XML data, in whole or in part. Then, the processed data can be converted back into its binary form. Additionally, the compressed data can be directly manipulated by any bitstream-syntax compliant applications. By allowing media data to be switched back and forth between XML and binary representations, XFlavor enables application developers to take advantage of the benefits offered by both representations.

3.4 Related Work

Bitstream Syntax Description Language (BSDL) [28, 29] is a new language, based on XML Schema, that has been introduced in MPEG-21 for describing the structure of bitstreams. Its purpose is to describe the high-level syntax of scalable bitstreams, and using the BSDL schema, the corresponding bitstreams can be converted back and forth between the binary and XML representations using generic software tools.

Once a scalable media data is in XML form, a generic software tool can be used to parse and manipulate such content regardless of its corresponding bitstream syntax. More importantly, XSLT can be used to easily manipulate the content. The goal of BSDL is to provide generic approach to scalable data manipulation so that, depending on the conditions related to the user, terminal, environment, and etc., a modified data that satisfies the conditions can easily be provided.

Though, the software tools in BSDL seem to provide the same functionalities as those of XFlavor, there are some differences between the two. Using BSDL, each one of the supported bitstream syntaxes must be hand-described. This is not a problem for the given purpose of BSDL, which is to describe only the high level structure of a given bitstream. But for a full description of the bitstream, due to the nature of the language, the description can get quite verbose and the process of generating the description can be error-prone and tedious. As a solution, XFlavor can be used to automatically generate XML schema corresponding to the given Flavor bitstream description. The Flavor description is much easier to write and in certain cases where the syntax is already described using Flavor (e.g., MPEG-4 Systems [4] and Advanced Audio Coding [37]), it is just a matter of copying the description and pasting it into a text file.

Additionally, BSDL uses generic software to convert bitstream descriptions into XML documents. The software is very slow in the conversion and for real-time applications, there is the need to provide content-specific (syntax-specific) software tools so that fast conversion can take place. For each bitstream syntax description, XFlavor provides the code to generate corresponding XML document much faster than the generic software tool.

4 Using XFlavor

Figure 6 illustrates the features and some of the possible applications of XFlavor. Starting with a Flavor description of a media bitstream, Flavor File A, XFlavor can generate an XML document generating code or an XML schema. Then, applications can use the generated schema to create new media data in the specified XML form (e.g., Media App A). Additionally, media bitstreams can be converted into corresponding XML documents that can be used for easier and more flexible processing. Once data is in XML form, different applications can easily manipulate it (e.g., Media App C and D) using a generic XML parser. On top of this, all the XML related technologies and tools could readily be used to process the data. Notably, media data in XML form can be manipulated by simply feeding stylesheets containing XSLT rules into a generic XSLT processor. Also, for space and time critical applications, the XML representation of the data can be converted back into the original bitstream representation for applications like Media App B.

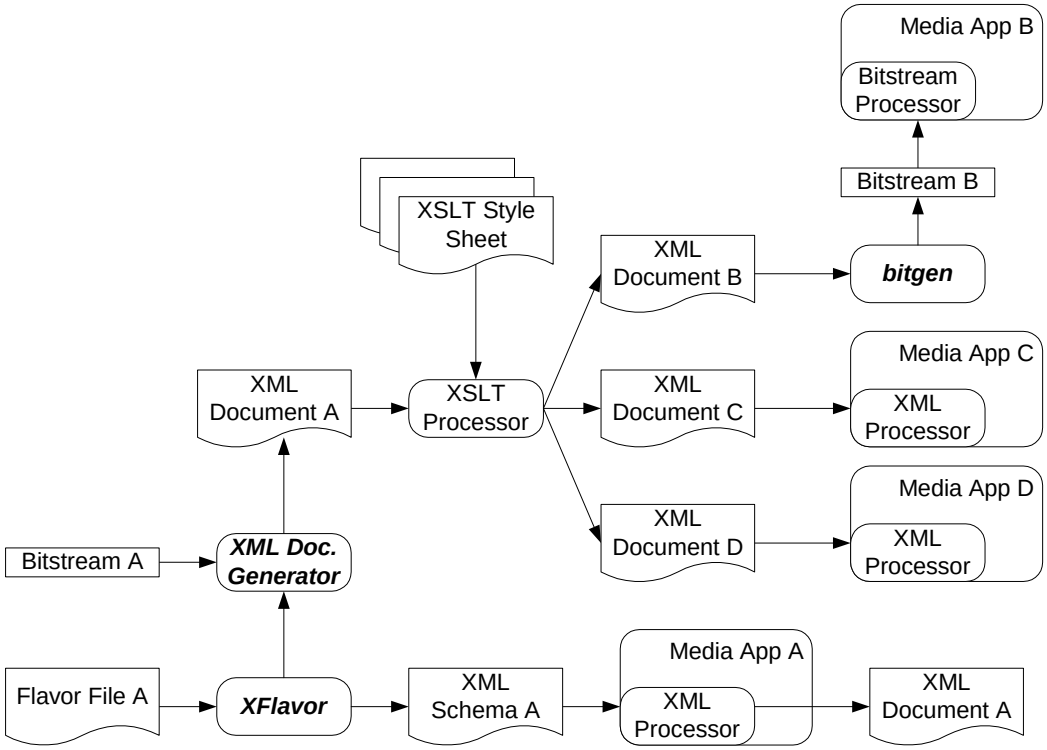

Figure 6: The functions of XFlavor used for different applications.

XFlavor is developed as a framework, assisting developers in building media applications/systems. As such, it can be used for a wide variety of applications and systems. As an example, we have built a prototypic system consisting of a database (DB) and media applications, and Figure 7 illustrates such system. The DB can contain media data in various formats, but for our example, we have chosen to store just video clips in two different formats: MPEG-1 and MPEG-2. Then, we generated two modules (one for MPEG-1 Video and another for MPEG-2 Video), using XFlavor, for converting the video bitstreams into XML documents. The DB can be considered to be in a different location from the modules and the applications. The key point is to keep the data in compressed bitstream format for storage and transmission. Once the data is the client computer, it can be converted into an XML document, enabling different XML-aware applications to process the data. The client can have a wide variety of applications for editing, transcoding, searching for a particular image/frame in a video clip, etc. Each one of the applications needs to be able to parse the data, and because the data is in XML form the parsing is automated by using a generic XML parser. Using a generic parser, we were able to parse both MPEG-1 and MPEG-2 Video clips in XML form: the value of any specific element (e.g., the width and height of an image, DCT values, motion vector residuals, etc.) was easily obtained via DOM [38, 39] or SAX [40] API. Note that by having the data in XML form, not only have we made the applications bitstream-format independent, but we have also enabled the applications to interoperate.

A large number of applications are already built that operate exclusively in XML-based data. In particular, a lot of content analysis, data mining, archiving, and searching applications are being recast as XML applications. Major databases all offer XML front-ends, so that data can be converted to and from the desired XML format. Even in standards like MPEG-7, XML has been adopted for the description of the structural and semantic aspects of content. It is thus obvious that a lot of the value-added content associated with media will be created and processed by an XML-aware array

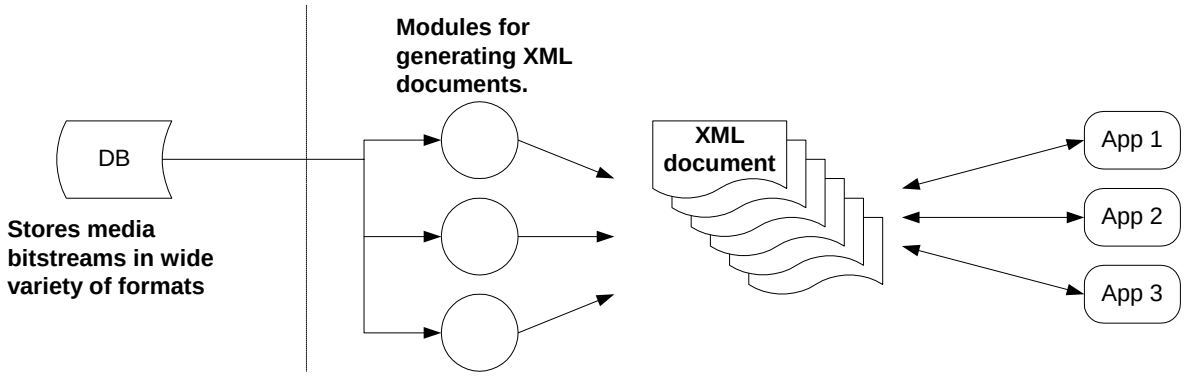

Figure 7: A prototypic system utilizing XFlavor.

of software tools. These XML-aware tools can also be made to process the actual media data by making the data available in XML format. Additionally, developers of XML-aware applications are already familiar with DOM/SAX API, and as a result, the data can be accessed effortlessly and they only need to focus on the actual processing part of the applications. All in all, we have allowed integration of media data to modern software applications with minimal or no additional effort by the part of the developer.

5 New XFlavor Architecture

The downside of the system described above is that the bitstreams must be converted into XML documents for the XML-enabled applications to be able to access the data. Though there are advantages to having a media file in the XML form (e.g., XSLT can be used to easily manipulate the data), it may not be practical for certain applications to create the intermediate file.

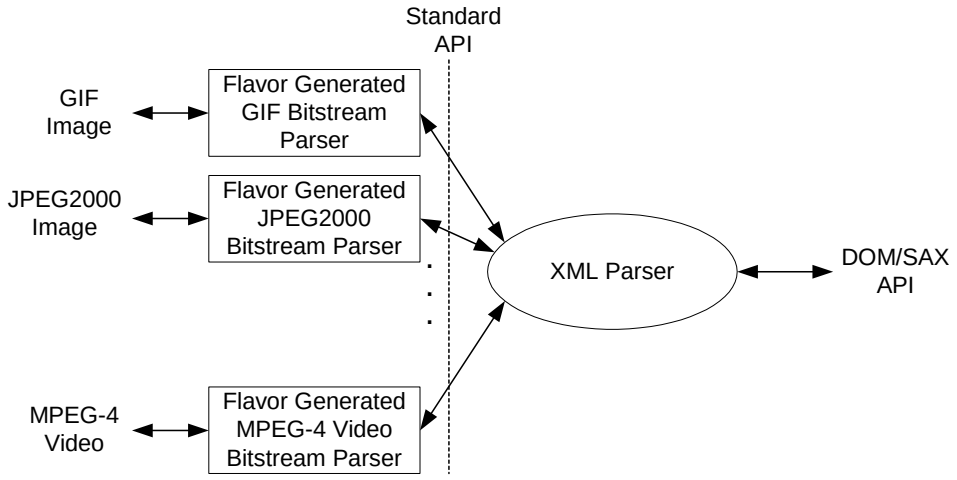

Figure 8: A generic bitstream parser supporting DOM/SAX API.

We are currently enhancing XFlavor so that the XML features can be more practically used. More specifically, we are implementing a generic parser, similar to an XML parser that can read and write different bitstreams and provide DOM and/or SAX API to applications. An XML parser (the Xerces XML parser – `http://xml.apache.org`) and the Flavor translator are being modified so that a standard API can be used between the modified XML parser and the translator-generated

bitstream parser. Consequently, application developers can use the new parser in exactly the same way as they would use any XML parser; however, the new parser will be used to access and modify bitstreams. This will significantly facilitate creating generic applications that process media data in different bitstream formats. As shown in Figure 8, the modified XML parser can use the code generated by the Flavor translator to access the bitstreams and convert them into XML form in memory. We will provide some control parameters so that depending on the memory requirement, for huge video bitstreams, only chunks of the data are read in at a time and converted into XML form, as needed. Note that this new generic parser is different from the generic software tools provided with BSDL, as it directly manipulates bitstreams without using a BSDL schema. Additionally, for each supported bitstream format, the code for accessing corresponding bitstreams can be plugged into the parser. This makes accessing bitstreams faster than using the generic BSDL software tools.

We also plan to modify an XSLT processor in the similar fashion so that bitstreams can be processed with simple XSLT stylesheets. Since most XSLT processors access XML documents using XML parsers, it will be a straightforward task to take the modifications to XML parsers and apply them directly to an XSLT processor. Figure 9 illustrates a modified XSLT processor.

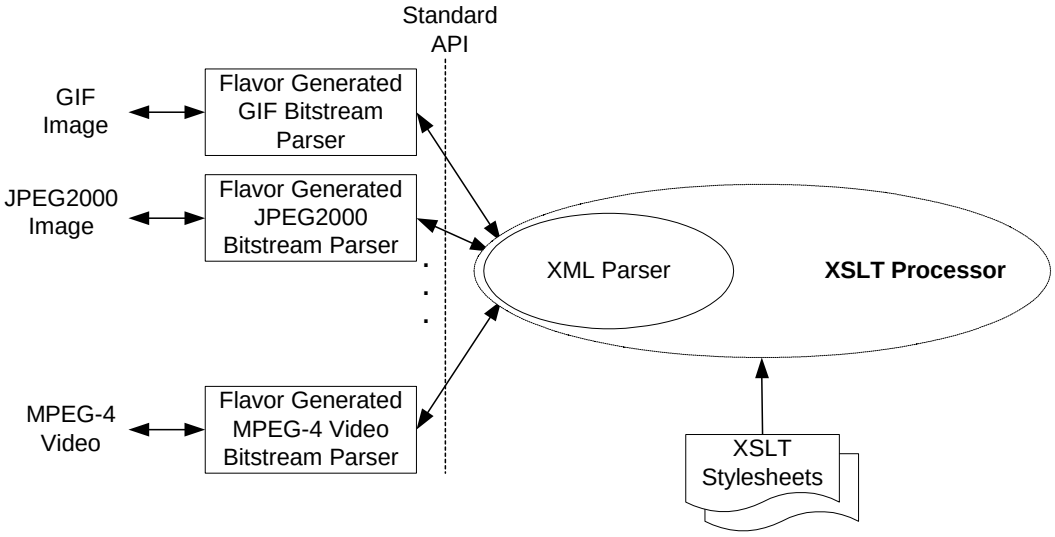

Figure 9: A generic, simple bitstream processor based on a generic XSLT processor.

6 Conclusion

We provide a framework, XFlavor, for converting media data encoded in a binary form into an equivalent XML document, and vice-versa. Multimedia communities (e.g., JPEG, MPEG, H.26x) have a tradition of preferring binary formats because they represent data in the most compact form. Though much less efficient in compression, XML representation offers some benefits that are not available in binary representation. Our work helps to show the advantages of using the textual representation, and helps to work with both representations at the same time. For storing or transporting data, the original coding format should remain the preferred format to be used for its efficiency. Also for memory-efficient, real-time applications, processing bitstreams yields better result than processing corresponding XML documents. To facilitate development of such applications, Flavor can be used. On the other hand, for simpler, more flexible, and general manipulation of media data, the XML representation can be used.

An XML document can be viewed as an abstract layer being added on top of a bitstream. This

hides the bitstream syntax and makes the semantic values of the data directly available. In other words, with an XML document, due to its self-describing nature, programs do not have to know the details of the binary representation such as byte and bit-offsets. As a result, software tools can be easily developed. Equally importantly, one generic tool can be used to process media data in various formats: rather than generating the necessary code for each bitstream format, each bitstream can be converted into an XML document, allowing a generic XML parser to access the data.

MPEG-1, 2 and 4, along with other widely used standards, provide the necessary information for creating media content. Then with the tremendous growth of available media data, MPEG-7 is developed to provide a uniform way for describing media content through standardized description schemes. This facilitates content searching, organizing, managing, etc., which are valuable applications. Then, we have MPEG-21 that defines a framework so that resources across wide range of networks and terminals can be flexibly combined for flexible distribution of media content. In summary, there are many media standards that provide the necessary framework for content creation, description, distribution, consumption, etc. However, tools that process media data must be developed according to the standards and there is no easy way to create them. As a solution, we present Flavor and XFlavor.

Note that the concept of XFlavor is different from the concept of MPEG-4/7 meta data descriptions. The latter is used to describe the information carried by the content whereas the former is used to describe the structure of the content.

References

[1] Y. Fang and A. Eleftheriadis. "A Syntactic Framework for Bitstream-Level Representation of Audio-Visual Objects". In *IEEE Int. Conf. on Image Processing*, Proceedings, pages II.426–II.432, 1996.

[2] A. Eleftheriadis. "Flavor: A Language for Media Representation". In *ACM Int. Conf. on Multimedia*, Proceedings, pages 1–9, 1997.

[3] Y. Fang and A. Eleftheriadis. "Automatic Generation of Entropy Coding Programs Using Flavor". In *IEEE Workshop on Multimedia Signal Processing*, Proceedings, pages 341–346, 1998.

[4] ISO/IEC 14496-1 International Standard. *Information technology – Coding of audio-visual objects – Part 1: Systems*, 2001.

[5] J. Levine, T. Mason, and D. Brown. *lex & yacc*. O'Reilly, 2^{nd} edition, 1992.

[6] R. Stevens. *Unix Network Programming*, volume 1. Prentice Hall, 2^{nd} edition, 1998.

[7] R. Stevens. *Unix Network Programming*, volume 2. Prentice Hall, 2^{nd} edition, 1999.

[8] IETF RFC 1014. *XDR: External Data Representation Standard*, 1987.

[9] IETF RFC 1832. *XDR: External Data Representation Standard*, 1995.

[10] ISO/IEC 8824. *Abstract Syntax Notation One (ASN.1)*, 2002.

[11] ISO/IEC 8825. *ASN.1 Encoding Rules*, 2002.

[12] W3C Recommendation. *Extensible Markup Language (XML) 1.0 (Second Edition)*, 2000.

[13] W3C Recommendation. *XSL Transformations (XSLT) Version 1.0*, 1999.

[14] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[15] CompuServe Inc. *Graphics Interchange Format*, 1987 and 1989.

[16] ISO/IEC 10918 International Standard (JPEG). *Information technology – Digital compression and coding of continuous-tone still images*, 1994.

[17] ISO/IEC 11172 International Standard (MPEG-1). *Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbits/s*, 1993.

[18] ISO/IEC 13818 International Standard (MPEG-2). *Information technology – Generic coding of moving pictures and associated audio information*, 1996.

[19] B. G. Haskell, A. Puri, and A. N. Netravali. *Digital Video: An Introduction to MPEG-2*. Chapman and Hall, 1997.

[20] ISO/IEC 14496 International Standard (MPEG-4). *Information technology – Coding of audio-visual objects*, 1999.

[21] *Signal Processing: Image Communication*, volume 15 no. 4–5 of *Tutorial Issue on the MPEG-4 Standard*, January 2000.

[22] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2^{nd} edition, 1993.

[23] K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language*. Addison-Wesley, 3^{rd} edition, 1996.

[24] S. W. Smoliar and H. Zhang. "Content-Based Video Indexing and Retrieval". *IEEE Multimedia Magazine*, 1994.

[25] S. F. Chang. "Some New Algorithms for Processing Images in the Transform Compressed Domain". In *ACM Int. Conf. on Visual Commnunications and Image Processing*, Proceedings, 1995.

[26] J. R. Smith and S. F. Chang. "VisualSEEk: A Fully Automated Content-Based Image Query System". In *ACM Int. Conf. on Multimedia*, Proceedings, 1996.

[27] ISO/IEC JTC1/SC29/WG11 N4801, Fairfax. *MPEG-21 Overview v.4*, May 2002.

[28] S. Devillers and M. Caprioglio. *Bitstream Syntax Description Language (BSDL)*. ISO/IEC JTC1/SC29/ WG11 M7433, 2001.

[29] S. Devillers, M. Amielh, and T. Planterose. *Bitstream Syntax Description Language (BSDL)*. ISO/IEC JTC1/SC29/WG11 M8273, 2002.

[30] W3C Recommendation. *XML Schema Part 0: Primer, XML Schema Part 1: Structures, XML Schema Part 2: Datatypes*, 2001.

[31] A. Avaro and P. Salembier. "MPEG-7 Systems: Overview". *IEEE Trans. on Circuits and Systems for Video Technology*, 11(6):760–764, 2001.

[32] http://www.expway.tv. *BiM*.

[33] W3C Note. *WAP Binary XML Content Format*, 1999.

[34] H. Liefke and D. Suciu. "XMill: An Efficient Compressor for XML Data". In *2000 ACM SIGMOD Int. Conf. on Management of Data*, Proceedings, pages 153–164, 2000.

[35] http://www.xmls.com. *XMLZip*.

[36] M. Girardot and N. Sundaresan. "Efficient Representation and Streaming of XML Content over the Internet Medium". In *IEEE Int. Conf. on Multimedia and Expo*, Proceedings, pages 67–70, 2000.

[37] ISO/IEC 14496-3 International Standard. *Information technology – Coding of audio-visual objects – Part 3: Audio*, 1999.

[38] W3C Recommendation. *Document Object Model (DOM) Level 1 Specification*, 1998.

[39] W3C Recommendation. *Document Object Model (DOM) Level 2 Specification*, 2000.

[40] http://www.saxproject.org. *Simple API for XML (SAX)*.