XML as a means of control for audio processing, synthesis and analysis

David García Garzón, Xavier Amatriain
{david.garcia,xavier.amatriain}@iua.upf.es

Music Technology Group
Institut Universitari de l'Audiovisual
Universitat Pompeu Fabra
http://www.iua.upf.es/~mtg

Abstract

This paper discusses about benefits derived from providing XML support to the component based framework for audio systems that we are developing. XML is used as data format for persistency, visualization and inter-application interface. Direct XML support is a very useful feature for an audio framework because of the popularity of the XML format as data interchange format, and the introduction of MPEG7 standard, an XML based description format for multimedia content. Formatting task has been distributed along the system objects in a compositional way, making easy to format a single object from its parts. The system minimizes the overhead added to a class and the programmer effort to support XML I/O. A default XML implementation has been provided for most of the future data structures, giving the chance to customize it. The system has been designed to be reused with other formats with a minimal impact on the system.

1 Introduction

A new component based C++[28] framework to implement complex audio systems is being developed by the Music Technology Group at the Pompeu Fabra University. Its purpose is to share all the programming effort done over the many applications developed within the group, and it provides a common way of doing and reusing.

The framework models audio systems as a set of processing entities which process data objects. Data objects types on the system (frames, descriptors...) are very heterogeneous in order to allow complex processing. Such heterogeneity is managed by providing uniform handling of data objects.

Often, those data objects are to be stored persistently for later processing by another application. The persistent format for data exchange has to be agreed between applications. XML not only provides an standard format template, but also standard ways to perform adaptations between different concrete XML formats.

2 XML as an application independent data format

Markup languages, like HTML, have been spread around the net. Their main advantage is their capability to be exchanged between heterogeneous systems whatever the application and even the platform. This interoperability is guaranteed by the partnership that establishes most of the standards that are used on the Internet content: The W3 Consortium (http://www.w3.org).

So HTML has the ability to interchange information between different platforms and applications but it was limited to web pages presentation. Why not extend this inter-operation ability to other applications and other types of documents? That is the aim of the XML standard.

XML is an extendible markup language that allows storing structured data conforming your own document structure. It is a hierarchical format, so, it can store compositional data. It is also a text based format, so, it is readable and modifiable by humans.

Several initiatives have taken profit from XML in music systems. Most of them are related to music notation [19, 11, 27, 23, 29], music generation [26, 5], multimedia synchronization [6, 7, 23] and multimedia databases [24].

3 How does an XML document look like

As an example, you can take a look to the following XML document:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<mainElement>
  <subelement1 attribute1="attribute value">
    plain content here
    <subsubelement>
      plain content
    </subsubelement>
    plain content here
  </subelement1>
  <subelement2 attribute2="attribute value">
    <subsubelement>
       plain content
    </subsubelement>
    <subsubelement>
       plain content
    </subsubelement>
    <subsubelement>
       plain content
    </subsubelement>
  </subelement2>
  <emptyelement attribute="foo" />
</mainElement>
```

Excluding the first line which is the XML document identifier, you can see the three main kinds of objects in an XML document: Elements, attributes and plain text content.

XML elements are the main hierarchical organizers. They start with an opening tag (the element name between angles) and they end with a closing tag (like the opening tag but having a slash before the name).

```
<ElementName>
        ...
</ElementName>
```

An element can contain others elements and plain text content between the opening and closing tags.

```
<ElementName>
        plain content
        <ASubElement>
                sub element content
        </ASubElement>
</ElementName>
```

An element can also contain attributes inside the opening tag. XML attributes have a name and a value. Attribute names cannot be repeated inside a single element while element names can.

```
<ElementName attributeName="value" />
```

The slash at the end of the previous tag is a shortcut for closing elements when there is neither plain text content nor sub-elements but only attributes.

You can take a look to the XML specification [10] for a more formal definition of the XML format.

XML is just a format template. You can represent the same information in many possible XML formats. So, storing XML, by itself, does not guarantee interconnection. Fortunately, there is an standard, XSLT [13], that describes the way to performs translations between two different XML formats. XSLT is very useful to fit XML based interfaces.

Being XML a template for other formats, concrete XML formats must be specified. Two standards are defined by the W3 Consortium in order to specify concrete XML formats: One deprecated specification language is Document Type Definitions (DTD). Lately, DTD has been superseded by the more versatile XML-Schema [16]. XML Schema allows to define XML data objects in a constructive way.

4 Implementation

4.1 Goals

The goals for the XML support for the framework can be summarized on the following ones:

- Provide XML support for new system objects with a minimum programmer effort.
- Allow quick automatic ways to make XML able classes
- Give mechanisms to customize XML representation to a different one that the automatically offered
- Allow the Integration of other formats (SDIFF, tagged text...)

4.2 Tools

XML support has not be done from scratch. The W3C Consortium defined a language-neutral API to deal with XML documents in memory. This API is the DOM, the Document Object Model [22][21][20].

We have used DOM API because is the accepted standard thought it has some major problems. The first one is that, in order to be language-neutral, implementations on the same language can slightly differ. It also provides a redundant interface so that both, object oriented and procedural languages can implement it, but, this has the counter part that libraries implementors tend not to provide the full API, only one of the redundant functions.

The second one is that DOM comes from several proprietary API's and specification process is too slow, so that implementors provide its own solution for unsolved problems.

The last problem, clearly derived from the previous two, is that DOM level 2 still does not specify how to load, store and validate documents, this is left to the implementors of libraries. DOM Level 3[20] is on progress, it will specify loading and storing and it will introduce XML-Schema aided parsing and validation[12].

Because library related particularities are so high, they have been isolated from the rest of the system encapsulating library calls.

We use a library named *Xerces for C++*[3], that is a C++ DOM implementation. It uses SAX API[25] as load/store interface. We chosen Xerces because it is an open source and highly portable library that fulfills the DOM standard very closely and it is object oriented.

Alternatives to Xerces and the reason we did not choose them were:

- **Oracle's XML Developer Kit:** Both library and documentation were too oriented to deal with Oracle XML databases. [2]

- **Microsoft XML:** Being a MS-Windows specific library it was discarded because we wanted a multiplatform framework. [1]

- **Gnome XML:** It is a portable implementation. We did not choose it because is written in C, it comes from an older and messy specification and it did not implement the whole standard API. After the decision, this library has been improved and is becoming a great library. [4]

4.3 Storages and Storables

To achieve the goals, the designed solution is based on the definition of several interfaces that objects can fulfill: Storable, Storage and Component. This solutions its an adaptation from pattern solutions taken from bibliography about the serialization problem and compositional environments [17, 18, 9].

Storage interface: A storage encapsulates a place where the state of an object can be saved to or retrieved from. There are a different Storage implementation for each supported format, so, when you store an object on a `XMLStorage` (the Storage implementation for XML format) the object is stored as XML.

Storable interface: Interface for objects to be stored on a Storage. Also, an Storable sub-interface for each format has been provided. This sub-interface is used by the Storage to extract some format dependent information from the Storable.

For the `XMLStorable` the Storage sub-interface is `XMLable`. This interface allows to extract the kind of XML node (element, attribute or plain content), the name when the Storable is an element or an attribute, and a content string.

4.4 Components

Component interface: The Component is the base class for any object that wants to became a member in a object composition in a way known by the framework.

The framework can apply hierarchical operations to the composition using the Component interface without worrying about object particularities. A Component can contain or be contained by other Components. Components composition determines the application data structure which is normally mapped to an XML (or whatever format) structure.

When an Storage accepts an Storable as a valid one to store, it may check whether the Storable is also a Component and if so, it ask the component to store its components too in a recursive way.

4.5 Adapters for non-storable objects

Implementing the XML storable interface for any object in the system is not always possible for several reasons:

- Some simple C types cannot implement it because they are not redefinable classes.
- Some external classes can not be modified.
- Loading all classes with the overload associated to implement the `XMLable` interface is often undesirable.

Moreover, extending the mechanism to others formats different from XML implies that every object must implement the `MyOwnFormatStorable` interface, and this is far from convenient. It is better bounding the set of the classes affected by this kind of change.

For user confortability, a set of XML-adapters has been defined. An adapter implements the XMLable interface based on the adaptee data and some initialization values. It also places the XML related information (like the name and whether it is an XML element) out of the stored object unloading it.

Currently, three kinds of adapters have been implemented:

Simple types adapters: We consider a simple type that one that implements insertion and extraction operators (<< and >>) to C++ ostreams. Most basic C types define them (int, float, char, char* ...). Also objects created by the framework user can define its own insertion/extraction operators. Because simple type adapters are based on such operators to extract the XML content, they are usable with a large amount of objects not being MTG specific.

Collection adapters: This adapter is designed to adapt any C array of simple types without using a huge amount of adapters objects.

Component adapters: A `XMLComponentAdapter` is an adapter that it is itself a Component and adapts a Component. So, when the storage confirms that the Adapter is a Component and asks it to store its sub-items the adapter forwards the request to its adaptee.

Although you can adapt a Component with a simple XML adapter, the `XMLStorage` only sees that the adapter is not a Component and then it doesn't look for sub-items.

Given those adapters, take a component and implement an `StoreOn` method (the one that stores sub-items) is very easy by, adapting each sub-item and storing the adapter on the given Storage.

In a few words, each format is managed by a different kind of storage object which uses a format-dependent storable interface to extract and fill information from/on the objects. Because not all objects obey all format interfaces the system uses adapters objects that wraps non storable objects into the format-dependent interface.

4.6 XML and the Dynamic Types

The framework uses extensively Dynamic Types as base for Data Objects, Configuration Objects and so. Dynamic Types is an implementation of the Component interface used to define data types which can partly instantiate its attributes.

One of their main features is that dynamic types know about its structure, mainly, the type and the name of their attributes. So they can use them to select an suited adapter for each attribute which type is known and implement a general StoreOn method.

So, when you define a new Data Object, it has a default XML implementation. This implementation stores all the attributes being basic C data types or Components.

This is an example of the XML fragment generated by an Spectrum object:

```
<Spectrum>
  <Config>
    <Scale>Linear</Scale>
    <SpectralRange>22050</SpectralRange>
    <Size>513</Size>
    <Type>MagPhase</Type>
  </Config>
  <MagBuffer>
     1.05376e-011 0.198708
     ...
     0.198376 0.198051 0.197732
     0.197419 0.197112
  </MagBuffer>
  <PhaseBuffer>
     0 -0.0109383 -0.0218712
     -0.0327933 -0.0436993
     ...
     2.77518 2.77767 2.78015
  </PhaseBuffer>
</Spectrum>
```

This Spectrum is a dynamic type containing three instantiated attributes: `Config` which is a Dynamic Type containing four configuration parameters for the spectrum, and two attributes, `MagBuffer` and `MagFase`, containing magnitude and phase values.

Spectrums can contain more instantiated attributes which are indicated in the Type attribute of the configuration.

5 Applications

5.1 Handling MPEG-7 descriptors

MPEG-7 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group), the committee that also developed the standards known as MPEG-1, MPEG-2, MPEG-4.

> MPEG-7, formally named *Multimedia Content Description Interface*, aims to create a standard for describing the multimedia content data that will support some degree of interpretation of the information's meaning, which can be passed onto, or accessed by, a device or a computer application. [14]

In short, MPEG-7 standardizes how to represent semantic and syntatic data about multimedia content. For example, semantic data about a song could be its author, its interpret, the title, the lyrics, the score, the tonality, the mood, relationship with other media, recording conditions, spectral information, performance expressivity... XML is related to MPEG-7 because descriptors are coded as XML documents.

Most of those descriptors are human knowledge based, but many others can be obtained from a cute automated analysis. Automated analyses suits the needs of such systems that handles big amounts of multimedia resources that can not be assisted by operators.

In this sense, and because MPEG-7 descriptors are, in fact, XML documents, supporting XML on the framework helps the development of tools that perform such automated analyses and store the results as MPEG-7 descriptors.

The most direct application for MPEG-7 descriptors is information retrieval from multimedia databases. RAA[8], an MTG project focused on music recognition, will benefit directly from using MPEG-7 descriptors in this way.

But descriptors can also be used in creative tasks. For example, descriptors can be used as articulation tool in order to re-synthesize audio media. They manipulate parameters with human meaning in order to perform some processing on the original media.

Many other projects in our group are oriented toward this kind of manipulation. For example, CUIDADO and TABASCO projects address toward the analysis, modeling and manipulation of expressive performance parameters that can be encoded as MPEG-7 descriptors.

5.2 Debugging tool

Because direct framework users would be sound systems developers, XML becomes one interesting feature used as debugging tool by providing a readable and structured way of watching framework objects on run-time at development stage.

This feature has been very useful during the framework development and it will be useful too when users will develop its own application over the framework.

5.3 Passivating the processing state

Applications doing non-real-time audio processing often do long lasting batch calculations. The XML support provided to the framework allows a very direct implementation of a passivation mechanism that stores the current state of a computation.

This allows to interrupt long processing, when the system gets heavily loaded or the machine needs a re-boot, and then restart the process later.

Periodical computation backups in XML can secure many inverted time against computer crashes.

5.4 Modular processing

On a second stage on the framework development, a visual tool for constructing networks of interconnected processing entities will be developed. The XML support will allow saving the network as an XML file.

Once you have designed a network you can reuse it as a new processing entity in a different design. Having a library of XML files defining networks would allow rapid incremental design.

6 Conclusions

The XML support for the framework is done in a nearly transparent way, being customizable when the user wants to. This provides the framework with huge possibilities interfacing with other systems; not only systems developed within the group but also systems developed by any other teams working on audio systems.

We have keep open possibilities to extend this mechanism to other formats. A very useful one will be SDIF[15].

Future issues are supporting XML Schema by generating automatically schemes from data object definitions and by using them as aid to XML parsing.

7 Acknowledgements

The work reported in this paper has been partially funded by the IST European project CUIDADO and by the TIC national project TABASCO. Authors also would like to thank Pau Arumí for its contributions on the first drafts of the design and Dynamic Types XML integration.

References

[1] Microsoft xml sdk.
http://msdn.microsoft.com/xml.

[2] Oracle xml developer's kits.
http://www.oracle.com/xml.

[3] Xerces c++.
http://xml.apache.org/xerces-c.

[4] The xml c library for gnome.
http://xmlsoft.org.

[5] Xavier Amatriain.
Metrix: A musical description language for a spectral modeling based synthesizer, 1999.
http://www.iua.upf.es/~xamat/metrix.

[6] Stephen Arnold, Carola Boehm, and Cordy Hall.
Mutated.
January 2000.
http://www.pads.ahds.ac.uk/mutated.

[7] Jeff Ayars, Dick Bulterman, Aaron Cohen, Ken Day, Erik Hodge, Philipp Hoschka, Eric Hyche, Muriel Jourdan, Michelle Kim, Kenichi Kubota, Rob Lanphier, Nabil Layada, Thierry Michel, Debbie Newman, Jacco van Ossenbruggen, Lloyd Rutledge, Bridie Saccocio, Patrick Schmitz, and Warner ten Kate.
Synchronized multimedia integration language (smil 2.0).
August 2001.
http://www.w3.org/TR/smil20.

[8] E. Batlle and P. Cano.
Automatic segmentation for music classification using competitive hidden markov models.
In *International Symposium on Music Information Retrieval*, 2000.

[9] Kent Beck.
Smalltalk Best Practice Patterns.
Prentice Hall, October 1996.

[10] Tim Bray, Jean Paoli, C. M. Sperberg, and Eve Maler.
Xml 1.0 w3c recommendation.
October 2000.
http://www.w3.org/XML.

[11] Gerd Castan.
Musixml.
April 2000.
http://www.s-line.de/homepages/gerd_castan/compmus/MusiXML_e.html.

[12] Ben Chang, Andy Heninger, Joe Kesselman, and Rezaur Rahman.
Document object model (dom) level 3 abstract schemas and load and save specification.
June 2001.
http://www.w3.org/TR/2001/WD-DOM-Level-3-ASLS-20010607.

[13] James Clark.
Xsl transformations (xslt).
August 2001.
http://www.w3.org/TR/xslt11.

[14] Neil Day and José M. Martínez.
Introduction to mpeg-7.
July 2001.
http://www.darmstadt.gmd.de/mobile/MPEG7.

[15] M. de Boer, J. Bonada, and X Serra.
Using the sound description interchange format within the sms applications.
In *Proceedings of International Computer Music Conference 2000*, 2000.

[16] David C. Fallside.
Xml schema w3c recommendation.
May 2001.
http://www.w3.org/XML/Schema.

[17] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.
Design Patterns-Elements of Reusable Object-Oriented Software.
Addison-Wesley, 1995.

[18] Adele Goldberg and D. Robson.
Smalltalk-80: The Language and Its Implementation.
Addison-Wesley, 1983.

[19] Michael Good.
Musicxml.
2001.
http://www.musicxml.org.

[20] Arnaud Le Hors, Philippe Le Hgaret, Gavin Nicol, Lauren Wood, Mike Champion, and Steve Byrne.
Document object model (dom) level 3 core specification.
September 2001.
http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913.

[21] Arnaud Le Hors, Philippe Le Hgaret, Gavin Nicol, Lauren Wood, Mike Champion, Steve Byrne, and Jonathan Robie.
Document object model (dom) level 2 core specification.
November 2000.
http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113.

[22] Arnaud Le Hors, Robert Sultor, Chris Wilson, Scott Isaacs, Ian Jacobs, Gavin Nicol, Lauren Wood, Mike Champion, Steve Byrne, and Jonathan Robie.
Document object model (dom) level 1 specification.
September 2000.
http://www.w3.org/XML/Schema.

[23] ISO/IEC.
Standard music description language (smdl).
September 1999.
http://www.ornl.gov/sgml/SC34.

[24] Robert Kaye and Johan Pouwelse.
Musicbrainz metadata initiative.
http://www.musicbrainz.org/MM.

[25] David Megginson.
Sax 2.0, simple api for xml.
http://sax.sourceforge.net.

[26] Bert Schiettecatte.
A format for virtual orchestras: Flowml.
2000.
http://wendy.vub.ac.be/~bschiett/saol/FlowML.html.

[27] Jacques Steyn.
Music markup language.
2000.
http://www.mmlxml.org.

[28] Bjarne Stroustrup.
The C++ Programming Language.
Addison Wesley, second edition, 1991.

[29] Jeroen van Rotterdam.
Musicml, an xml experience.
1999.
http://www.tcf.nl/3.0/musicml.