Managing large sound databases using Mpeg7

Max Jacob[1]

[1]*Institut de Recherche et Coordination Acoustique/Musique (IRCAM), place Igor Stravinsky 1, 75003, Paris, France*

Correspondence should be addressed to Max Jacob (max.jacob@ircam.fr)

ABSTRACT

Sound databases are widely used for scientific, commercial and artistic purposes. Nevertheless there is yet no standard way to manage them. This is due to the complexity of describing and indexing audio content and to the variety of purposes a sound database might address. Recently there appeared Mpeg7, a standard for audio/visual content meta-data that could be a good starting point. Mpeg7 not only defines a set of description tools, but is more generally an open framework allowing to host specific extensions for specific needs in a common environment. This is crucial since there would be no way to freeze in a monolithic definition all the possible needs of a sound database. This paper tries to line out how the Mpeg7 framework can be used, how it can be extended and how all this can fit into an extensible database design, gathering three years of experience during the CUIDADO and SemanticHIFI projects at IRCAM.

1. AN OVERVIEW ON MPEG7

Mpeg7 is a standard (ISO/IEC 15938) for audio-visual content description. It's purpose is to provide a common way multimedia content can be described, allowing better interchanges between people and applications and setting the basis for richer content retrieval techniques.

In order to design multimedia content descriptors, Mpeg7 has specified a Descriptor Definition Language (DDL), which has actually been used to design all the standard Mpeg7 descriptors. The Mpeg7 DDL is basically the XML Schema language [1], what means that Mpeg7's descriptors are defined as *XML Schema types* and Mpeg7 descriptions are *XML documents* [2]. For this reason many questions about storing and indexing Mpeg7 involve more generally questions about storing and indexing XML.

The XML Schema of Mpeg7 is mainly divided in four parts. The first one adds some numerical type extending those already built in the XML Schema language, such as vectors, matrices and a few others. The other three parts are called *audio*, *video* and *Multimedia description schemes* (MDS). The first two define a set of descriptors specific for audio and video content. The MDS instead defines the framework where audio and video descriptors can be embedded, managing relations, content segmentation, semantic descriptions and so on.

The audio and video descriptors are mainly numerical values, while the MDS defines complex, object oriented data structures. For this reason they raise quite different problems.

2. STORING MPEG7 PART I: THEORY

To store XML is at first sight a very trivial issue. You can simply use a file system and store Mpeg7 descriptions as text files. This is of course possible, and also completely satisfying from a pure informational point of view. But unfortunately the management of a large Mpeg7 database implies some more constraints, which are mainly the following:

- validation
- management of very large Mpeg7 documents
- efficient searches

We will have a look on them.

2.1. Mpeg7 validation

Usually an XML document is subjected to syntactical and structural constraints defined by a Document Type Definition (DTD) or, as for Mpeg7, by

an XML Schema. The process that ensures a single XML document meets such a specification is called *validation*.

But although XML Schema is a very powerful language, there are some aspects of the Mpeg7 specification it is not able to represent, as, for example, the consistency between the dimension of a matrix and its content etc. For this reason a real Mpeg7 validator should do a few things more than a standard XML Schema validator does.

2.2. Management of large Mpeg7 documents

Of course an Mpeg7 database should not only store descriptions but also allow to modify them or parts of them. For example you might want to rewrite the textual annotations about a music piece, or add a tempo value. And of course each of these operations must ensure the document still remains *valid*.

The easiest way to perform validation on insert and updates is to parse each time the whole document, update it if you are updating, pass everything through a validator and re-dump it to the database. This can be fine if you are talking about small documents. But during the CUIDADO project at IRCAM, I had once to load inside the database a document of 28 Mb with roughly 450.000 nodes. It was the description of a large sound collection, with about 20.000 sounds organized in 16 sub-collections. With a powerful computer it is possible to parse such a monster in memory. But what if you want to correct a type error on a title of one of those 20.000 sounds? Imagine an application where, just to retype a twenty character long text, it can take you several hours.

So unfortunately the *re-parse and dump* approach can not be adopted for a professional solution. The system must be smart enough to manipulate and validate small pieces inside the whole document separately. This means, from a technical point of view, that you can not store each Mpeg7 document as a *whole* but that you will have to decompose it into nodes.

2.3. Efficient searches

The W3C consortium has developed a language called XPath [3] that is a simple but very powerful way to address nodes in an XML tree. I believe that with XPath, especially with XPath 2.0, it is possible to do more or less all searches you might want to perform on an Mpeg7 database. But this is true only from a *formal* point of view. The reality is a bit harder.

To perform fast searches, DBMSs (Database Management Systems) relay on *indices* which are special storage techniques beside the normal *table* approach, focusing on a particular search task. And of course also searches in an Mpeg7 database will need indices, but they might not follow exactly the same logic than in traditional relational databases.

As already said, a part of the Mpeg7 standard, mainly the MDS, defines complex object oriented data structures. To do this, it takes advantage on the XML schema *inheritance* mechanism (see [1]). As it happens in object oriented programming languages (like C++ or Java), you can define general data types that are *specialized* by other more specific types recursively. Each type inherits data structures (and possibly behaviors) from its *ancestor* types. For example, in Mpeg7, we have the `SegmentType` that is specialized (among others) by the `AudioSegmentType` and the `VideoSegmentType`. According to the XML Schema specification, where a `SegmentType` is required, you can instantiate also any of its *descending* types. Of course segments have all some common feature, for example they can have sub-segments. I can manage this segment decomposition in a general way, regardless on the actual segment type, maintaining, for example, an index that links the sub-segments to the root segment, increasing the performance on some operation. Once i have defined this index, i don't want to re-define it for each child type of the `SegmentType` but would expect the system to be smart enough to apply it automatically.

This is crucial especially for managing *Mpeg7 extensions*. As many meta-data sets, Mpeg7 is extensible in order to allow a single application to use the standard adding some application-specific feature. The XML Schema type mechanism provides a very clean way to build new descriptors on top of the existing Mpeg7 types, and an Mpeg7 enabled database should support this possibility. If for example you define new specialization of the `SegmentType` you should not only be allowed to use it wherever a `SegmentType` occurs, but also have all associated indices working properly.

3. STORING MPEG7 PART II: PRAXIS

Now the question is: is there already a software on the market that does all we have seen till now?

The answer is unfortunately: *not yet.*

Let's see what is the state of the art for each point:

validation There are DBMSs that support XML Schema validation, but we had many troubles in loading the very complex Mpeg7 schema into some of them. Furthermore the validation process should be customizable in order to add Mpeg7 specific validation logic, and we did not find this feature anywhere.

large documents Some DBMSs can store XML decomposing it into nodes, which is the prerequisite for large documents management. On these systems it would probably also be possible to add the Mpeg7 specific validation using *triggers*, but only if they support XML Schema validation as well. And this is only partially true, if true at all, on all systems we have checked out.

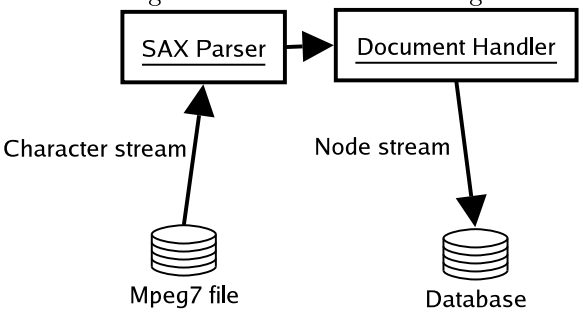
searches On many systems it would be hard to implement customized indices at all. In any case there seem to be no system allowing to index data in a way that can be understood by the XPath processor. And no system seems to be type-centric enough to allow something like *index inheritance*.

So far the bad news. The good news are that there are lots of tools and libraries that you can use (and customize if needed), providing XML parsing and validation. Furthermore there are lots of DBMSs with a nice API (Application Programming Interface) allowing you to write extensions. And this is what we actually did for the CUIDADO project at IRCAM.

4. THE CUIDADO MPEG7 DATABASE

During the European project CUIDADO (2001-2003) I was in charge at IRCAM for the development of an Mpeg7 based database system for Mpeg7 based applications. So I had to address all the problems we have seen in the previous chapters. After a lot of investigations and trials on existing software we finally decided to adopt an open source database system (PostgreSQL [4]) having a well documented C API and providing lots of advanced features, like *views*, *transactions* and a procedural language interpreter.

Fig. 1: Streamed XML loading



4.1. Streaming document loading

XML documents are loaded into the database using a SAX (Simple API for XML) parser. This is an *event based* parser, who reads the XML file and notifies a *document handler* each time it finds structural entities, like a start tag, an end tag or character data. This makes it possible to process the document while it is read, without any need to load it all in memory. So at least from this point of view there is no size limit for XML documents that can be loaded.

Each element of the XML tree is stored separately in the database, in order to *dump* data to disk while they are processed. This completes the streaming process as shown in figure 1.

A further benefit of storing each XML element separately in the database, is that you can move, update or remove it in a quite easy way and independently on the size of the document it belongs to.

4.2. Event handlers

We have also set up a *trigger mechanism* that allows to associate *event handlers* (or *triggers*) to database operations on the XML documents. These handlers are associated to either an element type or an element name or both and are called automatically by the system taking into account the XML Schema inheritance logic. The handled events are:

After insert Called after the insertion of an element.

Before remove Called before the removal of an element.

Before move Called before an element is moved on another tree or to another place in the same tree.

After move Called after an element has been moved.

Before update Called before an element is updated.

After update Called after an element has been updated.

The event handlers allow the creation and maintaining of indices. Indices are in this case tables where data are stored in a more *database friendly* way, allowing fast searches and easy retrieval of some particular data the application needs without having to browse the whole XML tree. Event handlers can be written in C or in the database procedural language, but we always used the latter since it is much easier and most times almost as fast as C.

Since the trigger system takes into account type inheritance, it is possible to *safely* extend Mpeg7 with a new type specializing the standard `ReferenceType`: the validating event handler will still be called also on instances of the new custom type.

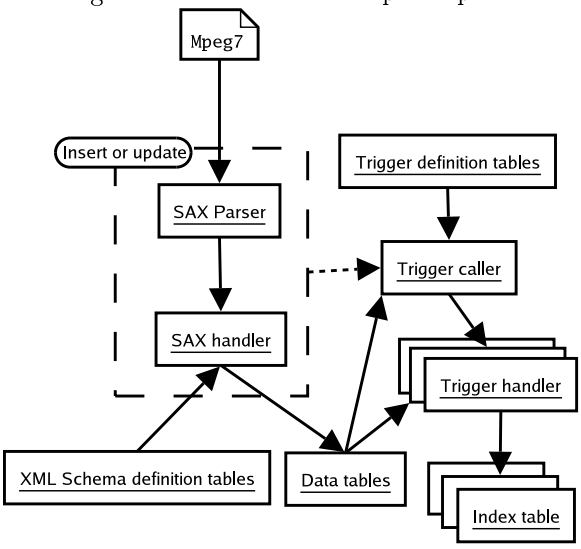
4.3. Validation

The standard XML Schema definitions are stored in a set of database tables allowing to verify for each inserted element whether it complies the schema definition or not. This operation is performed by the SAX document handler during the document insertion or update, as well as at *move* and *remove* operations.

This validation process is also important since it provides *type detection*. In an XML document the type declaration of each element can be omitted where the context allows to infer it. The validating process has to analyze the context and detect the actual type of each element, which is crucial at least for consistent trigger calls.

But, as said, Mpeg7 requires some validation rule that is not included in the XML Schema definition.

Fig. 2: The insertion and update process



This additional validation stage is, once again, performed by the triggers. For example the mpeg7 `ReferenceType`, which is supposed to point to an existing element, has an associated *after insert trigger* that checks this condition and throws an exception if it is fails.

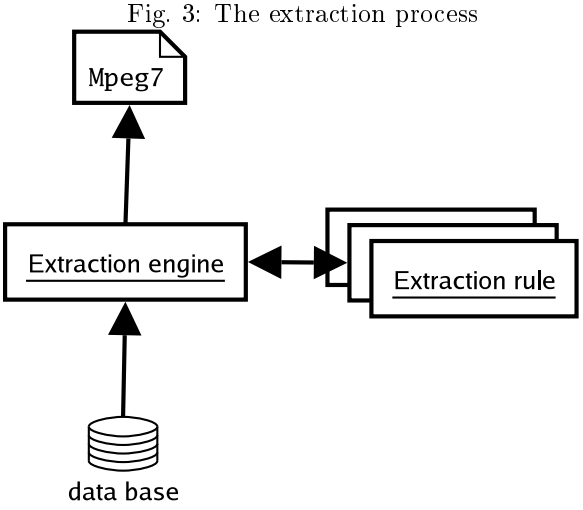
Figure 2 gives an overview on what happens when XML data are inserted or updated.

4.4. Extraction rules

When an XML document, or a branch of a document, is extracted from the database, an *extraction engine* browses the corresponding XML tree calling for each element an *extraction rule* (see figure 3). By default this rule rebuilds the corresponding XML code, but you can associate to each type *customized* extraction rules. We have found out that this mechanism is very useful for applications, since they might need the XML data packed in a different way than the Mpeg7 source code.

For example an application might need the XML data of a collection including its sub-collections, but without the contained audio entities (sounds), in order to display a file-system like tree. In this case the mechanism is not only useful, but even crucial, since a collection might contain thousands of sounds. Another example is on *referenced elements*. In Mpeg7 you can reference an element, for example, by its identifier. In this case the XML code would contain

Fig. 3: The extraction process



only the identifier of the referenced element, while an application will want to display something like a label or a title. All this can be managed directly in the database with customized extraction rules.

5. TOWARDS AN MPEG7 QUERY LANGUAGE

Mpeg7 defines a meta-data model, but it does not specify how an Mpeg7 database should be *queried*. And this is quite understandable, since such a logic is closely related to the purposes of each single application. And there can be huge differences. An application might perform statistical analysis on low level descriptors, another one could be a sound file management system, or an on line music store and so on.

The most natural hypothesis for a kind of *Mpeg7 query language* seems something relying on XPath, which is general enough to provide a large spectrum of possible uses. Such a language would make sure different Mpeg7 databases behaving exactly the same way (although performing differently) even if they was conceived for different purposes, allowing different applications to *really* interoperate.

Anyway this is very far from being a trivial issue. An implementation of such a system should not only provide a query language parser, but also a dedicated index definition mechanism the query interpreter understands. For the moment this does not exist even for standard XML + XPath, so we will probably have to wait for a while.

An alternative to an XPath based query language would be a dedicated Mpeg7 query language focusing on known use cases. An Mpeg7 database could just implement a part of the language. This would not allow *any* Mpeg7 based application to exchange data with *any* other, but ensures they can as long as they was built for similar purposes, which seams quite reasonable. During the CUIDADO project we have also conceived and implemented a query language that covers *our* use cases, and perhaps this could be further developed. This topic is still quite undefined but we hope to get some interesting result during the continuation of the SemanticHIFI project.

6. REFERENCES

[1] *XML Schema.*
http://www.w3c.org/XML/Schema

[2] *Extensible Markup Language (XML) 1.0.*
http://www.w3.org/TR/REC-xml

[3] *XML Path Language (XPath).*
http://www.w3.org/TR/xpath

[4] *The PostgreSQL web site.*
http://www.postgresql.org