X-Database. A middleware for collaborative video annotation, storage and retrieval

Iraklis Varlamis, Michalis Vazirgiannis

Dept of Informatics,
Athens University of Economics & Business,
Patision 76, 10434,
Athens, HELLAS
{varlamis,mvazirg}@aueb.gr

Panagiotis Poulos
Giorgos Akrivas, Spiros Ioannou
Dept of Electrical and Computer Engineering
National Technical University of Athens
9, Iroon Polytechniou Str., 157 73,
Athens, HELLAS
ppoulos@otenet.gr
{gakrivas, sivann}@ image.ece.ntua.gr

Abstract

Organising audiovisual content is a hard task for companies that own huge amounts of video information, such as television stations, studios, advertisement companies etc. The retrieval and re-use of useful information become unfeasible when the size of video information increases on a daily basis. The Panorama platform has been developed to cover such needs for manipulating video information, by attaching meta-information for both audio and visual content of video sources. This paper presents this meta-information model and the database interface developed in terms of the Panorama platform. The model works as keystone in the creation of the database and the database interface implements a mechanism for converting XML documents to relational data.

Keywords: Metadata, XML, Relational Databases, Video annotation

1. Introduction

Digitisation, annotation, meta-information attachment and organisation of video sources is a tedious process that requires time but offers many gains to companies that handle video content important: a) less storage space is required, b) longer endurance of digitised media is achieved, c) search is faster and more accurate when the media information is in digital format, d) parts of digitised programs can be easily retrieved and reused in other programs. Video meta-information such as information concerning the contributors, the technical specifications, the existing copies of a program and their maintenance state, the concept and content of an audiovisual program, is equally important to the digitised audiovisual program.

In order to cover this need for information recording and storage, a composite system is required that will control both the digitisation of audiovisual information, the extraction of content features and the recording of additional information, as well as the information retrieval and re-synthesis tasks. In the effort to support these requirements the Panorama platform is developed, to allow the video source owner to digitise existing programs to apply additional syntactic and semantic information and store it in a database, and to allow the end-user to search this information.

The video digitisation and annotation is a highly distributed process that requires information exchange among annotators and users. As a consequence, the information transferred among the platform users must follow a common information model, which in this case is expressed in XML-Schema. This information model follows the MPEG-7 definition for video metadata and is described in [1]. Furthermore all the audiovisual information transferred among the users, annotators and administrators is stored in a relational database management system in order to be available to other applications that do not use XML.

In this paper focus is given on the database part of Panorama. A short description of the whole system architecture is performed and a more detailed analysis of the database component follows.

Apart from the general requirements, a set of specific requirements concerning the database system has been addressed.

Submitted for the Research Track

- Information retrieval must be available for distant users
- The system must be available to users running different operating systems.
- Different user groups must have different levels of access to the database (Multilevel security)
- The database system must be able to handle audiovisual information

These specific requirements were taken under consideration during the selection of the database management system as well as during the design of the database component of the whole system.

In the following sections the platform architecture is discussed, the different modules are presented and the information model and the data transferred among the modules are explained (section 2). In Section 3 a detailed description of the module that serves as a database interface, namely the X-Database module, is performed. Certain issues that were considered during the database design are illustrated in the second part of this section. Section 4 presents related work from the area of video indexing and retrieval systems and attempts a comparison with commercial DBMSs in the way they handle XML. In section 5, some experimental results are presented and the final conclusions are drawn in section 6.

2. The platform architecture

2.1. Database

As mentioned in the introduction, the main objective was the design of a framework for extracting, describing, storing and querying video information that will be available both to remote (web) and local users. To accomplish this, the Oracle 8i database management system is used that supports multimedia data and handles efficiently the huge amount of data created by video decomposition and description process. Oracle Call Interface is used to communicate with the database, but an effort was made to create a database independent model, which will work with any underlying database management system using standard SQL commands.

2.2. Web & Local Interface

For enabling both remote and local users to access the information, two applications have been created: a web based application that uses ASP technology to create a user interface for querying and presenting the database contents and a local application developed in C++ that allows the media administrator to organize media and video content information into the database. Additionally, an application is developed to allow video annotators to work separately and save their work into files that are forwarded to the media administrator.

The database is accessed by the first two applications (web users and administrators), while the third application (annotators) produces data to be stored in the database but has no direct access to it. The web application performs only *select* statements to the database while the media administrator's application can perform *insert*, *update*, *delete*, *select* and in special cases *create* statements to the database. In addition to this, web application users need to have different access levels to the information stored in the database therefore several types of connection privileges are created.

2.3. Database Interface

According to the system requirements all the three applications should be independent of the underlying database system. In order to satisfy this design requirement one more application is built acting as an interface to the relational database management system. The application is implemented as a COM object developed in C++ that can be accessed both by web based and local applications.
A novelty for the system is that the information transferred among the applications follows a common model, an XML-Schema [2], which defines the structure of the information.

2.4. XML

The implemented data model is based in the directives of MPEG-7 [3] format for video metadata. The work of Jain and Hampapur [4] is also proved very useful in the development of the model. The emerging XML-Schema notation is used to express this model.

The use of XML induces many advantages to the system:
- It offers a common way of representing information transferred among applications
- Information produced by annotators can be saved in XML files, which can be reviewed and corrected at any time before being sent to the media administrator. The content of an XML document can be easily read or modified using a simple

text-editor and validated using the XML-Schema. Thus, the media administrator can easily review the information produced.
– The XML documents transferred through the system can be saved as files and presented to users, as informative documents concerning the videos.
– The analysis of the physical model of information can easily produce the XML-Schema. The XML language - which is object oriented - becomes a standard, and as a result object oriented analysis tools will soon provide the ability to export the model of information in an XML-Schema file. This XML-Schema can be used as input to our system.

Extending the capabilities of XML and XML-Schema, our framework maps the produced XML-Schema to a relational database schema and automatically creates the database. Several issues on mapping object-oriented information to a relational schema have been faced and solved during the development of the database interface. Problems and solutions considering the database architecture will be discussed in detail in the following.

Taking this one step further, the same database interface can be used in future applications. Once the information model is described in an object-oriented notation, it can be mapped into an XML-Schema and consequently into a database schema.

2.5. The information flow

The general architecture of Panorama is presented in fig.1. The flow of information among the various system components is described in the following.

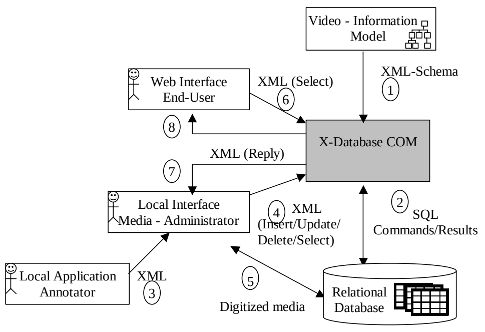


Fig1 (Short description of the system)

1) The administrator of the Video-Information Model, analyses the information to be stored in the database. He/she describes the different entities and relationships using an XML-Schema, thus providing a validating mechanism for the XML documents that will be created. This usually happens once at the beginning of the project, although tables and entities can be added into the database afterwards.
2) The X-Database interface receives the XML document created into step 1 (XML-Schema is also an XML document) and generates SQL commands to create the database.
3) The annotator provides descriptions of the programs' contents and generates an XML document containing content information for each video.
4, 5) The media administrator matches the digitised program descriptions with real media or media copies and adds all the media information related to each audiovisual object. The administrator inserts, updates, deletes and selects information from the database (step 4) and stores the digitised video files into the database (step 5)
6) The end-user can query the database through a web-based interface. User commands will be translated to appropriate XML documents before being sent to the X-Database interface.
7,8) The X-Database interface receives the XML document created during steps 4 and 6 and converts those to SQL commands. It also retrieves the query results and constructs an XML-Reply document, which is sent either to the media administrator (step 7) or to the end-user (step 8).

3. The X-Database (XML to Relational) component

The X-Database component can be divided in two parts. The first part is responsible for parsing XML-Schema documents and collecting information on the structure of XML documents that follow this schema definition. The second part parses XML documents and constructs the appropriate SQL commands that are processed by the database. It also takes database results and formulates valid XML documents as reply to user queries.

3.1. The XML-Schema explained

For a more detailed description of the first part of the component works, the XML-Schema used to describe video metadata, will be presented in the following. The XML-Schema definition contains three types of elements:
– complexTypes that represent the various entities of our video metadata model.

Complex types comprise of attributes, or group of attributes and sequences of elements.

- simpleTypes, which are used as enumerations of string values.
- attributeGroups, which only contain attributes and can be used for attributes that are common in many complex types.

Complex types describe the structure of XML elements. They contain:

- one ore more **<*xsd:attribute*>** tags (the ***use="required"*** attribute states if they are required or not
- one ore more **<*xsd:element*>** tags to describe sub-elements of the given element. Each sub-element has a name and a ***"type"*** or ***"ref"*** attributes. This means that an element can contain a sub-element as a whole, or can refer to the sub-element using its id.

Definition 1: The complex types that are not contained as ***"type"*** in other complex types are referred as ***Top-level*** types.

To give an example, the following XML-Schema fraction

```
<xsd:complexType name="AudioVisualDS">
  <xsd:attribute name="id" type="ID"
       use="required" />
  <xsd:attribute name="AVType"
       type="AVTypeD" use="required" />
 <xsd:sequence>
 <xsd:element maxOccurs="1" minOccurs="0"
   name="Syntactic" type="SyntacticDS" />
 <xsd:element maxOccurs="1" minOccurs="0"
   name="Semantic" type="SemanticDS" />
 <xsd:element maxOccurs="1" minOccurs="0"
   name="MetaInfoRef" ref="MetaInfoDS" />
 </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name="AVTypeD">
  <xsd:restriction base="string">
    <xsd:enumeration value="Movie" />
    <xsd:enumeration value="Picture" />
    <xsd:enumeration value="Document" />
  </xsd:restriction>
</xsd:simpleType>
```

denotes that an AudioVisualDS:

- has two attributes ("id" which is a number and "AVType" which can have one of the values Movie, Picture or Document),
- may contain two sub-elements namely Syntactic and Semantic (with their sub-elements) and
- may contain a reference to a MetaInfoDS element.

A valid XML file according to the above schema would be:

```
<AudioVisualDS id="1" AVType="Movie">
<Syntactic>
….
</Syntactic>
<Semantic>
….
</Semantic>
<MetaInfoRef>2<MetaInfoRef>
</AudioVisualDS>
```

In addition to this, since XML-Schema supports inheritance, certain entities of the schema extend the features of other entities. This is performed with the use of the **<*xsd:extension*>** tag. The extension entities contain all the features of their parent entity. The set of Complex types that represent information entities is stored in an XML-Schema file (namely xsdsource.xsd). This schema file also contains the descriptions of the available database commands.

The use of XML notation for describing the database commands allows applications to interact with the database using strictly defined XML documents. These well-defined documents not only contain the data that will be sent to the database but also describe the action that will be performed to the database. As a result a wrapping facility for the XML information that is sent or retrieved from the database is created.

Definition 2: The complex types that represent the information entities included in our model are referred as ***base elements***.
Definition 3: The complex types that are used to group base elements before they are sent for a transaction to the database are referred as ***extension elements***.

The extension elements used are:

- ***DBCommand*** that may contain many Insert, Update, Delete or Select elements,
- ***DBInsert, DBUpdate*** that contain the elements to be inserted or updated. In order to maintain the consistency of the database, only certain elements can be inserted or updated. The definition of ***DBInsert*** or ***DBUpdate*** in XML-Schema marks out which elements can be inserted or updated. For example if an element A contains an element of type B, only A can be sub-element of a ***DBInsert*** element. On the other side B can be inserted or updated in the database only as sub-element of A.

WRONG	CORRECT

`<Insert>`
`<B id="-2"/>`
`</Insert>`
	`<Insert>`
	`<A id="50">`
	`<B id="-2"/>`
	`</A>`
	`</Insert>`

- ***DBDelete*** that contains references to elements that can be deleted, so into a Delete element/command one can send the ids of many different elements to be deleted.

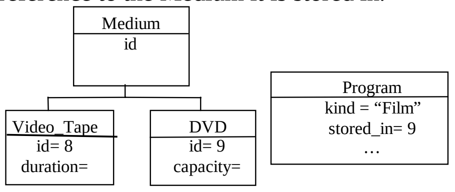
- ***DBReply***, which is the element used to enclose the retrieved information to applications. DBReply may contain all the base elements.
- ***DBSelect.*** This element has three parts: a) a ***"return"*** element that contains the base element(s) to be returned, b) a ***"from"*** element that contains the element(s) that will be used as criteria for the query and c) a ***"where"*** attribute that contains the query itself. Detailed examples of the Select process will be presented in the following paragraphs.

From the two groups of types only the base types are used for the database creation. The extension types are ignored during the database creation phase. They are used only for the validation of the commands that are sent to and from the database.

3.2. The Database structure

One of the basic requirements of the project was to retain the object-oriented nature of XML through the database transactions. This should allow users to access the database contents having in mind the structure of video information entities and not the structure of the database tables.

XML supports inheritance between types, which can be very confusing when using references. The following example describes a typical case of inheritance and reference. Entities "Video_Tape" and "DVD" are declared as extensions of entity "Medium" so they inherit all the attributes and elements of Medium in addition to their own attributes and elements. A "Program" entity makes a reference to the Medium it is stored in.



```
<xsd:complexType name="Program">
<xsd:attribute name="kind" type="string"/>
…
<xsd:sequence>
<xsd:element name="storedin" ref="Medium"/>
…
<xsd:sequence>
</xsd:complexType>
```

The reference is defined to the entity Medium but the referred id may be the id of a Video_Tape or a DVD. Therefore the relational database system must be adapted to support such features. In order to maintain such behaviour a table named OBJECTIDS is created in the database keeping record of the id and type of each element that may inherit or be inherited. A group of triggers certifies that a reference to children elements through their parent's name is stored correctly as a reference to the children element in the database.

Each ***"simpleType"*** in XML schema contains an enumeration of strings that represent the possible values an attribute can have. To give an example, the ***complexType*** AudioVisualDS has an attribute named AVType of type AVTypeD, where AVTypeD is a ***simpleType*** containing three different values (Movie, Picture, Document). This attribute in the database is mapped as a column AVType of type VARCHAR2 in table AudioVisualDS. A constraint is attached to the column:

AVType VARCHAR2(20) NOT NULL CONSTRAINT CHECK (AVType IN ('Movie','Picture','Document'))

Each ***"complexType"*** in XML-Schema is

```
  <xsd:complexType name="AudioVisualDS">
    <xsd:attribute name="id" type="ID"
         use="required" />
    <xsd:attribute name="AVType"
         type="AVTypeD" use="required"/>
     <xsd:sequence>…  </xsd:sequence>
   </xsd:complexType>

  <xsd:simpleType name="AVTypeD">
   <xsd:restriction base="string">
     <xsd:enumeration value="Movie" />
     <xsd:enumeration value="Picture" />
    <xsd:enumeration value="Document" />
   </xsd:restriction>
   </xsd:simpleType>
```

mapped into a table to the database. Additional tables are created to represent one-to-many relations between complexTypes. As already mentioned one complexType may refer to another in two ways: as ***"type"*** or as ***"ref".*** Additionally it can have zero, one or more than one (unbounded) references. The number of possible ***"ref"*** or ***"type"*** sub-elements is defined by the ***maxOccurs*** attribute. The ***minOccurs*** attribute in the schema defines if a field can be null (minOccurs=0) or not.

Combining the number of occurrences of a sub-element and the kind of references, four different types of relation between two complex types appear:

❖ Complex type A has ***exactly one reference*** to a complex type C.

This means that an element C must be created before an element A refers to it. The relation is one-to-one and there is no need for an intermediate table. So for complex types A and C two are tables created in the database. When an instance of C is destroyed then all the references to this must become null.

Constraints

table A:

CONSTRAINT A_C_ref FOREIGN KEY Cid REFERENCES C(id) ON DELETE SET NULL

table C:

CONSTRAINT C_Pkey id PRIMARY KEY.

❖ Complex type A has ***unbounded*** number of ***references*** to a complex type D.

This means that instances of D have been created before A refers to them. The relation is many-to-many, since two elements of type A may refer to the same elements of type D, so an intermediate table is needed (A_D_link), which will keep record of the order of the references inside A. So for complex types A and D three tables are created in the database. When an instance of A or D is destroyed then all related records in A_D_link are deleted.

Constraints

table A:

CONSTRAINT A_Pkey id PRIMARY KEY.

table D:

CONSTRAINT D_Pkey id PRIMARY KEY.
(if minOccurs=1 then Did NOT NULL.)

table A_D_link:

CONSTRAINT D_Pkey (Aid,Did) PRIMARY KEY.
CONSTRAINT A_D_ref FOREIGN KEY Aid REFERENCES A(id) ON DELETE CASCADE
CONSTRAINT A_D_ref FOREIGN KEY Did REFERENCES D(id) ON DELETE CASCADE

❖ Complex type A contains ***exactly one element*** of complex type B.

This means that the element B exists only inside A. The relation is one-to-one and there is no need for an intermediate table. So for complex types A and B two tables are created in the database. When an instance of A is destroyed then the B contained in it must be destroyed.

Constraints

table A:

CONSTRAINT A_Pkey id PRIMARY KEY.

table B:

CONSTRAINT B_Pkey id PRIMARY KEY.
CONSTRAINT B_A_ref FOREIGN KEY Aid REFERENCES A(id) ON DELETE CASCADE
CONSTRAINT Un_Aid Aid UNIQUE

❖ Complex type A contains ***unbounded*** number of ***elements*** of complex type E.

This means that elements of type E are created only after the container element A is created. The relation is one-to-many, since two different elements of type A have their own distinct sub-elements of type E. The order of appearance of elements E inside an A is stored inside table E. So for complex types A and E two tables are created in the database. When an instance of A is destroyed then all the B contained in it must be destroyed.

Constraints

table A:

CONSTRAINT A_Pkey id PRIMARY KEY.

table E:

CONSTRAINT E_Pkey id PRIMARY KEY.
CONSTRAINT E_A_ref FOREIGN KEY Aid REFERENCES A(id) ON DELETE CASCADE

3.3. Converting XML commands to SQL commands

The XML-Schema is used to create the database, but is also used to validate the XML documents that are transferred among the applications. During the creation of the XML-Schema all the physical relations among the various entities were expressed as relations of containment or reference among the respective complex types.

Therefore, when the complex types are converted into database tables their relations must be correctly expressed into database constraints. Additionally, attempts to insert, update, delete or select a complex type will result in a set of insert, update, delete or select queries that will handle all the information that relates to the complex type. A correct set of constraints, both database and programming ones, guarantee the integrity of the database into the aforementioned actions. Some of the constraints are:

a) Certain entities exist only as ***part of*** other entities. The respective complex types appear only as sub-elements of other complex types. As a consequence these complex types cannot appear inside a ***DBInsert*** command. To give an example: the syntactical information is related to an audiovisual medium, so SyntacticDS appears only inside an AudioVisualDS, so that nobody can store syntactic information into the database that does not belong to any audiovisual medium.
b) Certain entities can be ***re-used by more than one*** entity. The complex types that correspond to such entities must appear as sub-elements of ***DBInsert.*** When a complex type refers to another complex type, the latter must already have been inserted into the database.
c) Certain entities contain ***ordered instances*** of other entities, which in XML-Schema terms means that a complex type may have more than one sub-elements of another type. The order in which these sub-elements appear is important hence extra information must be

stored in the database during the XML document parsing.
d) One or more commands can be send into the X-database module at the same time. These commands usually evoke a reply from the database, so the DBReply module must be able to group the database replies for each command.

These constraints are incorporated into the XML-Schema in the definition of ***extension elements***. A pre-parsing of the XML-Schema document gives the X-Database module all the information needed for the parsing of the XML documents.

The four database commands supported by the module are:
1) Insert: One or more ***top-level*** *complex types* can be found inside a ***DBInsert*** element. The parsing starts from the top-level element and continues recursively to all sub-elements, thus generating and executing a series of SQL INSERT statements.
2) Update: One or more ***top-level*** *complex types* can be found inside a ***DBUpdate*** element. All elements having a ***negative id*** are inserted. The rest of the elements, which have a positive number as id (this is the id provided by the database) are updated. The ids are returned (as in ***DBInsert***)
3) Delete: Only ***top-level*** *complex types* with attribute ***"id"*** can be found inside a DELETE element. The database cascade deletes the information of all sub-elements.
4) Select:
A DBSelect element has three parts:
- A ***where*** part that contains the filter of the Select query.
- A ***from*** part where the elements that contain the attributes to be filtered appear.
- A ***return*** part that contains the element(s) to be returned.

The whole element is returned after a select statement. The following XML fraction is an example ***DBSelect*** command.

```
<Command>
<Select where="@-100@ = 1234">
<return>
<AudioVisualRef>-98</AudioVisualRef>
</return>
<from>
<AudioVisual id="-96" Name="Gladiator"
   AVType="Movie">
<MediaInfo id="-97">
<MediaProfile>
<MediumRef>-100</MediumRef>
</MediaProfile>
</MediaInfo>
 </AudioVisual>
 </from>
 </Select>
</Command>
```

This command selects all the AudioVisualDS entities that refer to a MediumDS with id=1234. The values of various attributes that appear inside the *"from"* tags are not taken into account during the document's parsing. These values are randomly assigned to the ***required*** attributes, to maintain the validity of the XML document. Only the value of –100 field is important and that is because –100 appears in the where clause.

The value of *where* attribute is the **"where_clause"** of our query. From the nested structure of elements inside the "from" part of the command, the parser is able to create all the ***"join conditions"*** among the tables that take part in the query. From the elements that appear in the *"from"* and *"return"* elements the ***"list_of_associated_tables"*** is generated.
The return element has a reference to the complex type(s) that must be selected. These references give the name of the table that corresponds to the ***"return_entity"***.
So the first query that is send to the database has the following structure:

SELECT *return_entity.id* FROM
"return_entity", "list_of_associated_tables"
WHERE *("join_conditions*")
AND (*"where clause")*

This query returns a list of ids of the entity to be returned. Using these ids a set of recursive select commands is addressed to the database to obtain all the information of the entities to be returned. The DBReply element contains the entities in their complete structure.

4. Related Systems

Several research and commercial systems provide automatic indexing and querying of visual contents. QBIC [5] creates a colour histogram for each image, detects shapes and considers spatial relationship of objects. Combined with IBM's DB2 [6] and its extenders it provides a powerful platform for handling visual contents but it is not very effective in audio contents. VIRAGE's [7] Audio and Video Logger provide a good tool for annotating audiovisual sources and building a retrieval mechanism upon these annotations. All the previous platforms provide solutions on the audiovisual content annotating and indexing, but unfortunately are not available for web applications. Nonetheless, they can be used as a basis for extracting information from the audiovisual sources that can be wrapped in our XML model.

As far as it concerns the XML part of the project, a great effort was made to maintain the validity of the transferred documents according the XML-Schema. The X-Database part of the platform achieved to keep the query mechanism fairly simple, by combining the default structure of XML documents with the simplicity of SQL "Select" commands. The query mechanism provided may not be very powerful as those of other platforms, like OQL-S of Ozone [8], or WebOQL [9] but has certain advantages. The query itself is an XML document, whose certain parts (***from, return***) have the same structure as the other XML documents that are inserted or updated and the where part can be easily expressed in an SQL-like manner. Most of the query categories proposed in [10] could be performed using our DBSelect element, such as Simple Visual Feature Query, Feature Combination Query, Query by Example etc.

Compared to the interfaces used by commercial relational database management systems, X-Database provides a complete solution in XML documents manipulation. DB2 XML extender [11] supports the use of XML DTDs only for describing the database schema but does not support XML-Schema, which is more powerful in schema definition. Microsoft SQL Server [12] uses specific template files to describe the database schema. Informix [13], Oracle [14] and Sybase [15] mainly support creation of XML files from database contents and do not support XML-Schema. All the above systems do not provide the ability to create the database schema based on the XML-Schema and moreover to use the same XML-schema to validate all the XML documents and commands that are forwarded to the database.

5. Experimental evaluation

In order to test the reliability but also the scalability of the X-Database module a series of test transactions is performed to the database. These transactions included database creation, multiple insertions and deletions, updates and selections of database contents. In all the transactions the total response time is measured. This includes the parsing of XML input documents the time for accessing the database and creating the XML reply document.

The system has been measured using a simple interface, where XML documents are inserted as text and the resulting XML is displayed in a web browser. The simulation was running in a Pentium II computer with 128MBytes of RAM and an IDE HD. The working version of Panorama platform uses a more powerful machine and as a consequence its performance is better.

The X-Database algorithm is tested using different kinds of application loads in a simulation. The performance of the module in creating a small or larger database schema, in handling multiple insert, update or delete commands and in processing less or more complicated select queries are evaluated.

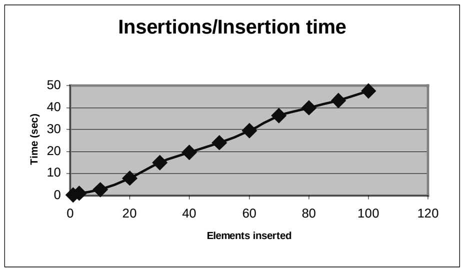
5.1. Database Complexity

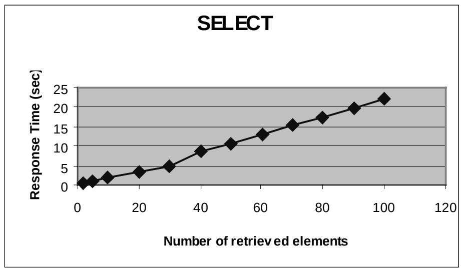
The most critical section in the X-Database work is the creation of the database. The module must analyse the structure of XML-Schema document and create the appropriate number of tables along with the required foreign keys and triggers that will guarantee the integrity of the database in the cascade insertions and deletions. Several parameters of the database schema have been measured, such as the number of tables created, the number of foreign key and triggers that guarantee the schema integrity, as well as the database creation time for XML-Schema files of different complexity. When the number of complexTypes, the number of references and the number of extensions in the XML-Schema increases, the number of tables in the database schema increases respectively. The results are presented in the following table.

Complex Types	**Unbounded refs**	**Extensions**	**Tables**	**Triggers**	**Foreign keys**	**Creation time (sec)**	**Drop time (sec)**
68	**35**	**18**	**140**	**77**	**260**	**52**	**31**
63	**31**	**14**	**111**	**51**	**197**	**41**	**20**
56	**29**	**14**	**99**	**44**	**172**	**38**	**20**
50	**27**	**13**	**88**	**40**	**154**	**31**	**20**
45	**26**	**9**	**81**	**36**	**139**	**29**	**11**
37	**23**	**8**	**69**	**30**	**93**	**19**	**9**
28	**18**	**5**	**50**	**21**	**64**	**15**	**8**
15	**9**	**5**	**26**	**10**	**31**	**7**	**4**
5	**5**	**0**	**11**	**6**	**12**	**3**	**3**

5.2. Insertion-Selection time

In order to test the efficiency of our system, the time needed for a set of insertion and selections from the database has been measured. The database schema used for the test was created using an XML-Schema file of increased complexity, with 68 complexTypes and 18 extensions. It is explicit in the following graphs that both insertion and selection time is linear to the number of retrieved elements given that all the elements are of the same type. When elements of different type are inserted, the number of consequent insertions differs and as a consequence the relation to time is not linear.





An audiovisual object stores its information into 34 different tables. As a result, an attempt to insert an audiovisual object into the relational database evokes 34 insert SQL commands, which are executed recursively into the database.

In the following figure, the tables where the information of an audiovisual object is stored are listed.

- ❖ AudiovisualDS
- ❖ ObjectIDs
  - SyntacticDS
    - SyntacticDS_ KeyFrameRef_link
    - ThemeDS
      - ThemeDS_ KeyFrameRef_link
      - ShotGroupDS
        - ♦ ShotGroupDS_ KeyFrameRef_link
        - ♦ ShotGroupDS_ ShotRef_link
      - ShotDS
        - ♦ KeyframeDS
        - ♦ MovingRegionDS
  - SemanticDS
    - SemanticDS_EventRef_link
    - SemanticDS_ObjectRef_link
    - SemanticDS_ EventObjectRelationRef_link
  - SyntacticSemanticLinkDS
    - SynSemDS
      - SynSemDS_ ObjectRef_link
      - SynSemDS_ EventRef _link
      - SynSemDS_ EventObjectRelationRef _link
  - MediaInfoDS
    - MediaProfileDS
      - MediaRecordDS
      - MediaProfileDS_Medium_link
  - Summarization
    - HierarchicalSummary
      - ThemeSummaryDS
        - ♦ ShotSummaryDS
          - ShotSummaryDS_KeyFrameRef_link
        - ♦ ShotGroupSummaryDS
    - SequentialSummary
      - SequentialSummaryDS_ShotRef_link
      - SequentialSummaryDS_KeyFrameRef_link

6. Conclusion and further work

Attempting to create a platform that handles digitised video data and meta-information, where different modules will access the same database, XML is chosen to be the wrapper of the information transferred among them. A set of rules formed as an XML-Schema document is then developed, to guarantee the validity of transferred documents. The explicitness of XML-Schema has been exploited to build the database where information is stored.

The X-Database module created plays the role of the database interface. Client applications interact with the relational database system only using XML documents without taking care of the underlying database schema. This provides a simple and database independent mechanism for storing and retrieving video meta-information that can be easily extracted to any kind of information. Fault tolerance is achieved by adding appropriate control procedures to the database, such as triggers that check the validity of inserted values, reference constraints that guarantee the cascade removal of an object and its content objects from the database. The last is very critical since information for an object may be stored in more than one table in our database.

Further research has to be focused into the retrieval and advanced query tasks. The retrieval process can be easily accelerated if the appropriate indexes are created. For this task the current XML-Schema can be enriched to precisely define the information that must be indexed, or even more the information entities on which to perform similarity search etc. Finding an efficient notation to describe such necessities in XML-Schema is crucial in creating a database schema that will serve advanced retrieval needs.

7. References

[1] G. Akrivas, S. Iwannou, E. Karakoulakis, K. Karpouzis, Y. Avrithis, A. Delopoulos, S. Kollias, I. Varlamis, M. Vazirgiannis, An Intelligent System for Archiving and Retrieval of Audiovisual Material Based on the MPEG-7 Description Schemes, Technical paper.

[2] “XML Schema Part 0: Primer,” W3C Working Draft, Sept. 2000 (http://www.w3.org/TR/xmlschema-0)

[3] ISO/IEC JTC1/SC29/WG11, “MPEG-7 Overview (v. 1.0),” Doc. N3158, Dec. 1999.

[4] R. Jain, A. Hampapur: Metadata in Video Databases. SIGMOD Record 23(4): 27-33 (1994)

[5] M. Flickner et al, Query by Image and Video Content: The QBIC System, IEEE Computer Vol. 28, No. 9, September 1995.

[6] DB2 Video extenders, (http://www.software.ibm.com/data/db2/)

[7] A. Gupta, Visual Information Retrieval Technology, A VIRAGE Perspective white paper, Virage, 1995.

[8] S. Abiteboul, J. Widom, T. Lahiri, A Unified Approach for Querying Structured Data and XML. The Query Languages Workshop, QL'98.

[9] G. Arocena, A. Mendelzon, WebOQL: Restructuring Documents, Databases, and Webs, Proc. ICDE'98, Orlando, February 1998.

[10] Y. Alp Aslandogan and Clement T. Yu, "Techniques and Systems for Image and Video Retrieval", IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, Jan.-Feb. 1999.

[11] IBM's DB2 extender for XML (http://www-4.ibm.com/software/data/db2/ extenders/xmlext.html)

[12] Microsoft SQL Server XML support, (http://msdn.microsoft.com/msdnmag/issues/ 0300/sql/sql.asp)

[13] Informix and XML, (http://www.informix.com/xml/)

[14] Steve Muench, Using XML and Relational Databases for Internet Applications, Oracle Corporation (http://technet.oracle.com/tech/xml/info/htdocs/ relational/index.htm#ID795)

[15] Sybase SQL server, (http://www.sybase.com/products/ databaseservers/ase/whitepapers/L01041.pdf)