Chapter 2

Web multiform data structuring for warehousing

J. Darmont, O. Boussaid, F. Bentayeb, S. Rabaseda, and Y. Zellouf
ERIC, Université Lumière Lyon 2, France

Abstract: In a data warehousing process, the data preparation phase is crucial. Mastering this phase allows multidimensional analysis or the use of data mining algorithms, as well as substantial gains in terms of time and performance when performing such analyses. Furthermore, a data warehouse can require external data. The web is a prevalent data source in this context, though the data broadcasted on this medium are very heterogeneous.

In this chapter, we propose a modeling process for integrating all these diverse, heterogeneous data into a unified format. Furthermore, the very schema definition provides first-rate metadata in our data warehousing context. At the conceptual level, a complex object is represented in UML as a superclass of any useful data source (databases, plain or tagged texts, images, sounds, video clips, etc.). Our logical model is an XML schema that can be described with a DTD or the XML-Schema language. Eventually, we have designed a Java prototype that transforms our multiform input data into XML documents representing our physical model.

Then, the XML documents we obtain are mapped into a relational database. We view this database as an ODS (*Operational Data Storage*), whose data will have to be re-modeled in a multidimensional way to allow their storage in a warehouse and, later, their analysis.

Key words: Web farming, Multiform data, Integration, Modeling process, UML, XML, Mapping, Data warehousing, Data analysis

1. INTRODUCTION

In the context of e-commerce, analyzing the behavior of a customer, a product, or a company consists in monitoring one or several activities (commercial or medical pursuits, patent deposits, etc.). The objective of multidimensional analysis, particularly OLAP (*On-Line Analytical Processing*), is to analyze such activities under the form of numerical data. The information is summarized and can be presented as relevant information (i.e., knowledge) allowing to couple OLAP with other analysis tools such as KDD (*Knowledge Discovery in Databases*) techniques (namely, data mining), whose objectives include understanding and predicting the behavior of one or several activities. Hence, the scope of analysis can be extended.

To be efficient in terms of quality and response time, analysis tools need their input data to be properly structured, acquired, and prepared in a previous step. These data are typically stored in databases aimed at decision support (such as data warehouses) that we call *Decision Support Databases* (DSDBs). These databases often necessitate external data sources. For instance, a company willing to support competitive monitoring cannot merely analyze only data from its own production databases. In this context, the web may be considered as a farming system (Hackathorn, 2000) providing input to a data warehouse. However, the data broadcasted on this medium are very heterogeneous and are not in a form suitable for data warehousing. A process of refining must be performed on such data before they can be loaded into a data warehouse. This renders their conceptualization in a data warehousing framework difficult. Nonetheless, the concepts of data warehousing (Kimball, 1996; Inmon, 1996; Chaudhuri and Dayal, 1997) remain valid in this approach (Thuraisingham, 2001). Measures, though not necessarily numerical, remain the indicators for analysis, and analysis is still performed following different perspectives represented by dimensions. Large data volumes and their dating are other arguments in favor of this approach (Kimball and Mertz, 2000).

The web then becomes a full data source. Data from the web may be used to consolidate the description of facts already warehoused by adding new values to a dimension, or by creating new dimensions or new fact tables to broaden the scope of analysis. Multiform data may also be loaded into the warehouse to be later analyzed with either OLAP or data mining techniques. This raises several issues:

– structuring multiform data from the web — databases, plain texts, multimedia data (images, sounds, video clips...), semi-structured data (HTML, XML, or SGML documents) — into a database;
– integrating these data into the particular architecture of a data warehouse (fact tables, dimension tables, data marts, data cubes);

– devising evolution strategies for the warehouse when new data pop up;
– physically reorganizing data depending on usage to improve query performance.

The aim of this chapter is to address the first issue and to expand the preliminary results presented in Miniaoui, Darmont, and Boussaid, 2001. We propose a modeling process for integrating the multiform data we need to store in a DSDB. We first designed a unified UML (OMG, 1999) conceptual model for a complex object representing a superclass of these data. Note that our objective is not only to store data, but also to truly prepare them for analysis. This is not a mere ETL (*Extracting, Transforming, and Loading*) task, which would only render data names and domains consistent. Then, we translated our UML conceptual model into an XML (Bray et al., 2000) schema definition that represents our logical model. Eventually, this logical model is instantiated into a physical model that is an XML document. The XML documents we obtain with the help of a Java prototype are mapped into a (MySQL) relational database with a PHP script. This database represents an ODS (*Operational Data Storage*), which is a temporary data repository that is typically used in an ETL process before the data warehouse proper is constituted.

The reasons why we selected XML as a pivot formalism are numerous. First, XML encapsulates both data and their schema, either implicitly or in a schema definition. This representation is also found in data warehouses, which store both data and metadata that describe the data. Hence, XML is particularly adapted for our purpose. Moreover, we benefit from the flexibility, the extensibility and the richness of the semi-structured data model. And since XML documents can easily be mapped into a conventional (e.g., relational) database (Anderson et al., 2000; Kappel, Kapsammer, and Retschitzegger, 2000), we also take advantage of well-structured data and query processing efficiency. Furthermore, XML-based databases like Lore (McHugh et al., 1997) are quickly expanding, and we could easily switch to such systems if needed. Hence, we get the best of both worlds (the structured and the semi-structured) by adopting the XML format.

The remainder of this chapter is organized as follows. Section 2 establishes a state of the art in the fields we studied before proposing our solution: federated, multimedia, and text databases (from an integration perspective), web collections, XML mapping, and XML querying. Section 3 presents our unified conceptual model for multiform data. Section 4 outlines how this conceptual model is translated into a logical, XML schema definition. Section 5 details how our input data are transformed into an XML document representing our physical model and presents examples of how our Java prototype transforms the data. The issue of mapping the obtained

XML documents in a relational database is also addressed. We finally conclude the chapter and discuss future research issues in Section 6.

2. RELATED WORK

2.1 Data integration

We identified three main ways to integrate heterogeneous data into a data warehouse. The first approach relates to federated databases (Hsiao, 1992a; Hsiao, 1992b; Busse et al., 1999; Bertino, Catania, and Zarri, 2001). Federated databases are distributed and heterogeneous databases constituted from data sources of various natures: different kinds of databases (object, relational, object-relational...), HTML or XML documents, and so on. However, they must provide users with an integrated view of the data. The casual architecture for a federated database is layered in three levels:

– *presentation:* components allowing to formulate queries in the federated database language;
– *mediation:* mediators in charge of collecting queries issued by users in the presentation components and translating them in the proper language of each data source;
– *adaptation:* components allowing communication between data sources and mediators.

The second approach consists in capturing the common characteristics of the different data types we need to integrate. In order to propose a unified data model, we took interest in how data is structured, stored, and indexed in textual and multimedia databases. Indexing strategies in textual databases include reversed lists of significant terms with their frequency of appearance in each document, signatures obtained by hashing keywords, and relative frequency matrix of words present in a set of documents (Rakow, Neuhold, and Löhr, 1995). Multimedia databases may adopt the following characteristics to index images: signatures derived from (manually captured) visual descriptors describing the image, color, texture or brightness distributions, etc. Eventually, video and audio data are handled in a similar way. They are typically described by texts and metadata regarding the content of the frames constituting the continuous document. They may also be described by sample frames (Thuraisingham, 2001).

The last approach concerns metadata and relates to web collections (Hopmann, Berkun, and Hatoun, 1997). Web collections are an XML application that describes the properties of objects in a common metadata format and provides a hierarchical structure for the data to describe.

Therefore, each collection corresponds to a set of metadata. The mainspring of web collections is a good basis for the development of a core set of metadata. The main goal for defining such a common metadata set is the multiplicity of possibilities for fielded searches as well as for sorting and filtering results from a database. To each collection corresponds a specific profile that specifies an identifier for the collection (e.g., a WebPage profile would state that a collection is a web page) and properties such as the page title and the author name. In short, a collection is simply an association of field names to values. This description may be in a separate document or enclosed inside the original HTML document (a block of XML code may appear anywhere inside the HTML document enclosed between `<XML>` `</XML>` tags).

2.2 XML Mapping

Semi-structured data such as XML documents are characterized by the lack of any fixed schema, although the data have some implicit structure. The main problem is to uncover this structure. Semi-structured data are typically stored in file systems that provide limited support for organizing, searching, and operating them. Current database systems are inappropriate for semi-structured information because they require the data to be translated into their data model.

On the other hand, XML provides many database functions: storage, schemas, query languages, programming interfaces, etc. However, it lacks many of other important tools found in true databases: efficient storage, indices, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, etc.

There are three possible approaches to store semi-structured data:

– build a special-purpose database system such as the research prototypes Rufus (Shoens et al., 1993), Lore (McHugh et al., 1997), and Strudel (Fernandez et al., 1998); or the commercial product Lotus Notes[1];
– exploit the rich data modeling capabilities of object-oriented database system (Christophides et al., 1994; Zwol, Apers, and Wilschut, 1997);
– map XML data into relational tables (Anderson et al., 2000; Kappel, Kapsammer, and Retschitzegger, 2000).

In theory, special purpose-systems should work best, but in practice, they are not mature yet. Current object-oriented systems perform poorly when evaluating queries on very large databases. Relational database systems are mature and scale very well. Moreover, in a relational database, XML data and traditional structured data can co-exist. However, the requirements for

[1] Lotus Notes homepage: *http://www.lotusnotes.com*

processing XML data are different from the requirements for processing traditional data. Recent work has concentrated on models and algorithms to extract schemas from XML documents (Nestorov, Abiteboul, and Motwani, 1998).

2.3 XML query languages

The role of an XML query language is to allow an application to extract precisely the information it needs from one or several XML data sources through structural and content-based queries (Deutsch et al., 1999b). XML data are fundamentally different from relational and object-oriented data. Hence, neither SQL nor OQL is appropriate for XML querying. In database models, every data instance has a schema that is separate and independent. In XML, the schema coexists with the data as tag names. XML is self-describing and can naturally model irregularities that cannot be modeled by relational or object-oriented data (Deutsch et al., 1999a).

Two approaches are drawn: (1) consider a repository of XML data, where XML documents constitute a database; or (2) store XML documents in a relational database, where XML data are shredded into rows in relational tables. Semi-structured query languages help querying the data stored in XML documents. They include X-Query (Fernandez, Marsh, and Nagy, 2001), XML-QL (Deutsch et al., 1999a; Deutsch et al., 1999b), Lorel (Abiteboul et al., 1997), which is an extension of OQL, XQL (Robie, Lapp, and Schah, 1998), proposed by the W3C, which uses path expressions for navigating in the XML nested structure, XML-GL (Ceri et al., 1999), which is a graphical query language, and WEBL, which is a scripting language for HTML and XML documents with a markup algebra.

On the other hand, to query XML documents mapped into a relational database, semi-structured queries (specified in a semi-structured query language) over XML document must be translated into SQL queries over the corresponding relational tables, and the results must be converted back to XML (Florescu and Kossmann, 1999; Shanmugasundaram, 1999; Shanmugasundaram, 2001). For example, the STORED approach uses a combination of relational and semi-structured techniques (Deutsch, Fernandez, and Suciu, 1999).

3. UML CONCEPTUAL MODEL

The data types we consider (text, multimedia documents, relational views from databases) for integration in a data warehouse all bear characteristics that can be used for indexing. The UML class diagram shown in *Figure 1* represents a complex object generalizing all these data types. Note that our goal here is to propose a general data structure: the list of attributes for each class in this diagram is willingly not exhaustive.

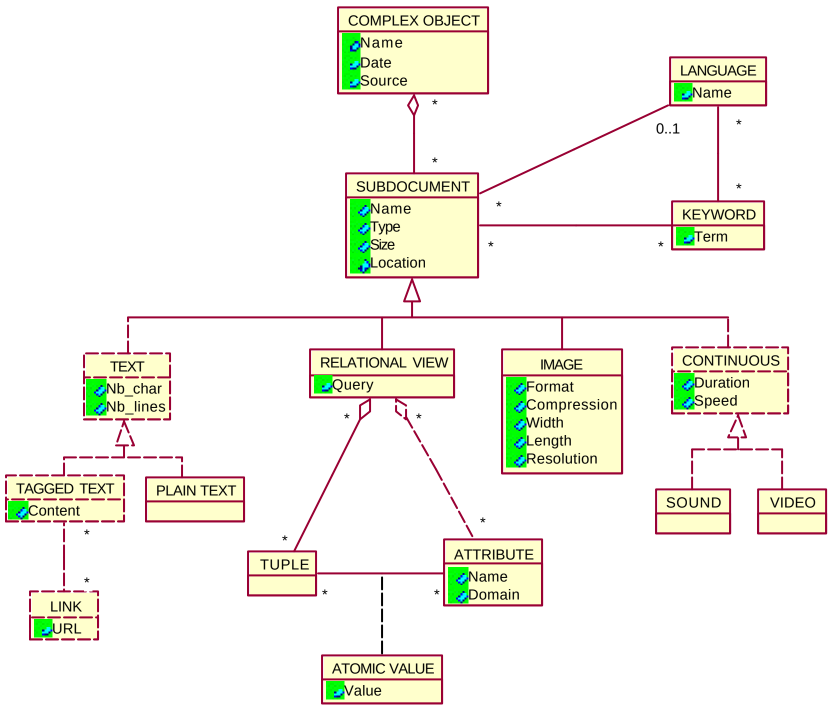

Figure 1. Multiform conceptual model

A complex object is characterized by its name and its source. The date attribute introduces the notion of successive versions and dating that is crucial in data warehouses. Each complex object is composed of several subdocuments. Each subdocument is identified by its name, its type, its size, and its location (i.e., its physical address). The document type (text, image, etc.) will be helpful later, when selecting an appropriate analysis tool (text mining tools are different from standard data mining tools, for instance). The language class is important for text mining and information retrieval purposes, since it characterizes both documents and keywords.

Eventually, keywords represent a semantic representation of a document. They are metadata describing the object to integrate (medical image, press article...) or its content. Keywords are essential in the indexing process that helps guaranteeing good performances at data retrieval time. Note that we consider only logical indexing here, and not physical issues arisen by very large amounts of data, which are still quite open as far as we know. Keywords are typically manually captured, but it would be very interesting to mine them automatically with text mining (Tan, 1999), image mining (Zhang, Hsu, and Lee, 2001), or XML mining (Edmonds, 2001) techniques, for instance.

All the following classes are subclasses of the subdocument class. They represent the basic data types and/or documents we want to integrate. Text documents are subdivided into plain texts and tagged texts (namely HTML, XML, or SGML documents). Tagged text are further associated to a certain number of links. Since a web page may point to external data (other pages, images, multimedia data, files...), those links help relating these data to their referring page.

Relational views are actually extractions from any type of database (relational, object, object-relational — we suppose a view can be extracted whatever the data model) that will be materialized in the data warehouse. A relational view is a set of attributes (columns, classically characterized by their name and their domain) and a set of tuples (rows). At the intersection of tuples and attributes is a data value. In our model, these values appear as ordinal, but in practice they can be texts or BLOBs containing multimedia data. The query that helped building the view is also stored. Depending on the context, all the data can be stored, only the query and the intention (attribute definitions), or everything. For instance, it might be inadequate to duplicate huge amounts of data, especially if the data source is not regularly updated. On the other hand, if successive snapshots of an evolving view are needed, data will have to be stored.

Images may bear two types of attributes: some that are usually found in the image file header (format, compression rate, size in pixels, resolution), and some that need to be extracted by program, such as color or texture distributions.

Eventually, sounds and video clips are part of a same class because they share continuous attributes that are absent from the other (still) types of data we consider. As far as we know, these types of data are not currently analyzed by mining algorithms, but they do contain knowledge. This is why we take them into account here (though in little detail), anticipating advances in multimedia mining techniques (Thuraisingham, 2001).

4. XML LOGICAL MODEL

In a data warehouse, data come with metadata that describe their origin, the rules for transformations they may have undergone, and information regarding their usage (Wu and Buchmann, 1997). Identically, multiform data modeled as complex objects may be viewed as *documents*. It is then necessary to consider two kinds of data: data themselves and sets of information pieces, such as descriptive data that allow their identification, or status data describing their semantics.

The use of XML as an implementation tool for multiform data (i.e., complex objects) viewed as documents seems natural. This language indeed helps representing both the description and the content of any document. Within a modeling process, the translation of UML classes representing multiform data into XML constitutes a logical formalization phase. The obtained logical model can be as well mapped in a relational or object-relational database as stored in a native-XML database.

When integrating multiform data, we adopt a classical information system modeling process: first devise a conceptual model, and then translate it into a logical model. The UML class diagram from Section 3 is our conceptual model. We consider XML as a fine candidate for logical modeling.

The UML model can indeed be directly translated into an XML schema, whether it is expressed as a DTD (*Document Definition Type*) or in the XML-Schema language (Fallside, 2001). We considered using XMI (Cover, 2001) to assist us in the translation process, but given the relative simplicity of our models, we proceeded directly.

The schema we obtained, expressed as a DTD, is shown in *Figure 2*. We applied minor shortcuts not to overload it. Since the *LANGUAGE*, *KEYWORD*, *LINK*, and *VALUE* classes only bear one attribute each, we mapped them to single XML elements, rather than having them be composed of another, single element. For instance, the *LANGUAGE* class became the *LANGUAGE* element, but this element is not further composed of the *Name* element. Eventually, since the *ATTRIBUTE* and the *TUPLE* elements share the same sub-element "attribute name", we labeled it *ATT_NAME* in the *ATTRIBUTE* element and *ATT_NAME_REF* (reference to an attribute name) in the *TUPLE* element to avoid any confusion or processing problem.

```
<!ELEMENT COMPLEX_OBJECT (OBJ_NAME, DATE, SOURCE, SUBDOCUMENT+)>
   <!ELEMENT OBJ_NAME (#PCDATA)>
   <!ELEMENT DATE (#PCDATA)>
   <!ELEMENT SOURCE (#PCDATA)>
   <!ELEMENT SUBDOCUMENT (DOC_NAME, TYPE, SIZE, LOCATION, LANGUAGE?,
   KEYWORD*, (TEXT | RELATIONAL_VIEW | IMAGE | CONTINUOUS))>
      <!ELEMENT DOC_NAME (#PCDATA)>
      <!ELEMENT TYPE (#PCDATA)>
      <!ELEMENT SIZE (#PCDATA)>
      <!ELEMENT LOCATION (#PCDATA)>
      <!ELEMENT LANGUAGE (#PCDATA)>
      <!ELEMENT KEYWORD (#PCDATA)>
      <!ELEMENT TEXT (NB_CHAR, NB_LINES, (PLAIN_TEXT | TAGGED_TEXT))>
         <!ELEMENT NB_CHAR (#PCDATA)>
         <!ELEMENT NB_LINES (#PCDATA)>
         <!ELEMENT PLAIN_TEXT (#PCDATA)>
         <!ELEMENT TAGGED_TEXT (CONTENT, LINK*)>
            <!ELEMENT CONTENT (#PCDATA)>
            <!ELEMENT LINK (#PCDATA)>
      <!ELEMENT RELATIONAL_VIEW (QUERY?, ATTRIBUTE+, TUPLE*)>
         <!ELEMENT QUERY (#PCDATA)>
         <!ELEMENT ATTRIBUTE (ATT_NAME, DOMAIN)>
            <!ELEMENT ATT_NAME (#PCDATA)>
            <!ELEMENT DOMAIN (#PCDATA)>
         <!ELEMENT TUPLE (ATT_NAME_REF, VALUE)+>
            <!ELEMENT ATT_NAME_REF (#PCDATA)>
            <!ELEMENT VALUE (#PCDATA)>
      <!ELEMENT IMAGE (COMPRESSION, FORMAT, RESOLUTION, LENGTH,
      WIDTH)>
         <!ELEMENT COMPRESSION (#PCDATA)>
         <!ELEMENT FORMAT (#PCDATA)>
         <!ELEMENT RESOLUTION (#PCDATA)>
         <!ELEMENT LENGTH (#PCDATA)>
         <!ELEMENT WIDTH (#PCDATA)>
      <!ELEMENT CONTINUOUS (DURATION, SPEED, (SOUND | VIDEO))>
         <!ELEMENT DURATION (#PCDATA)>
         <!ELEMENT SPEED (#PCDATA)>
         <!ELEMENT SOUND (#PCDATA)>
         <!ELEMENT VIDEO (#PCDATA)>
```

Figure 2. Logical model (DTD)

5. XML PHYSICAL MODEL

We have developed a prototype capable of taking as input any data source from the web, fitting it in our model, and producing an XML document. We view the XML documents we generate as the final physical models in our process.

5.1 Transformation algorithm

The general algorithm for integrating multiform data into our unified model is provided in *Figure 3*. Its principle is to parse the schema introduced in *Figure 2* recursively, fetching the elements it describes, and to write them into the output XML document, along with the associated values extracted from the original data, on the fly. Note that, when reading a DTD line, the current element we refer to is the one which is being described, e.g., *TEXT* in the `<!ELEMENT TEXT (NB_CHAR, NB_LINES, (PLAIN_TEXT | TAGGED_TEXT))>` DTD line. We also suppose that sub-elements are defined in the same order they are declared in their parent element. Missing values are currently treated by inserting an empty element, but strategies could be devised to solve this problem, either by prompting the user or automatically.

```
// Initialization
Write XML document prologue
Read DTD line
Push root element
// Main loop
While stack not empty do
   Pop element
   // Positioning on the current element description
   While element not found in the DTD and not EOF(DTD) do
      Read DTD line
   End while
   If element was found then
      For each value of the element do
      // For elements with + or * cardinality}
         If element is atomic then
            Write elementBeginTag, elementValue, elementEndTag
         Else // Composite element
            Write elementBeginTag
            Push element // Necessary to later write end tag
            For each sub-element (in reverse order) do
               If sub-element does not belong to a selection then
               // If element not in a list
               // of the form (PLAIN_TEXT | TAGGED_TEXT)
                  Push sub-element
               Else
                  If sub-element was selected then
                  // If the DTD document type matches
                  // the actual document type
                     Push sub-element
                  End if
               End if
            End for
         End if
      End for
   Else
      Write elementEndTag // Close composite elements
   End if
End while
```

Figure 3. Multiform data integration algorithm

5.2 Implementation

Our prototype for data transformation, `web2xml`, has been coded in Java, for portability purposes. Its full code is freely available on-line[2].

[2] `web2xml` Java prototype download URL: *http://bdd.univ-lyon2.fr/download/web2xml.zip*

5.2.1 Architecture

The architecture, i.e., the classes in our application, is displayed on *Figure 4*. The lower part of the class diagram represents our internal data structure and reuses the classes introduced in *Figure 1*. The interface of the application appears in the upper part of the diagram. *APPLICATION2* is the main class of the Java program, the one that is started up by the user. All the **_INTERFACE* classes correspond to graphical user interfaces for the specification of complex objects (name, source), subdocuments (name, type, language, and keywords), relational views (query and JDBC Data Source Name, which was not present in the conceptual model, but is necessary at the physical level), images (compression rate and resolution), and continuous documents (duration and speed), respectively. Other classes and/or attributes, such as texts and hyperlinks in web pages, are treated automatically.

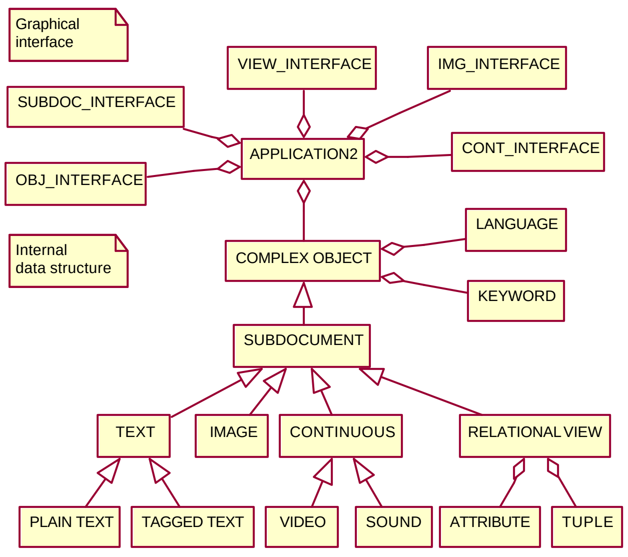

Figure 4. Java prototype architecture

5.2.2 Attribute extraction

The first step of our approach consists in extracting the attributes of the complex object that has been selected by the user. A particular treatment is applied depending on the subdocument class (image, sound, etc.), since each

subdocument class bears different attributes. We used three ways to extract the actual data:
- manual capture by the user, through graphical interfaces;
- use of standard Java methods and packages;
- use of ad-hoc automatic extraction algorithms.

Table 1 recapitulates how each attribute of each class in our conceptual model is captured. Our objective is to progressively reduce the number of manually-captured attributes and to add new attributes that would be useful for later analysis and that could be obtained with data mining techniques.

Note that, when processing the content of texts, only texts shorter than *N* lines are stored directly into the XML document, where *N* is a user-defined parameter. In the case of longer texts, only a reference to the file is stored.

Class	Attribute	Capture	Class	Attribute	Capture
COMPLEX	Name	Manual	RELATIONAL	Query	Manual
OBJECT	Date	Java	VIEW	JDBC DSN	Manual
	Source	Manual	ATTRIBUTE	Name	Java
SUB-	Name	Manual		Domain	Java
DOCUMENT	Type	Manual	ATOMIC	Value	Java
	Size	Java	VALUE		
	Location	Java	IMAGE	Format	Ad-hoc
LANGUAGE	Name	Manual		Compress	Manual
KEYWORD	Term	Manual		Width	Java
TEXT	Nb_char	Ad-hoc		Length	Java
	Nb_lines	Ad-hoc		Resolution	Manual
	Content	Ad-hoc	CONTINUOUS	Duration	Manual
LINK	URL	Ad-hoc		Speed	Manual

Table 1. XML document generation

5.2.3 XML document generation

The second and last step when producing our physical model consists in generating an XML document. This generation process strictly follows the algorithm provided in *Figure 3*. However, our internal data structure lies mainly on vectors. Hence, access to some attributes is indexed. It is thus necessary, when generating the XML document, to know which element is being processed to be able to fetch the corresponding information. Hence, the stack was modified to include the index of each pushed element.

5.3 Output

At this point, our prototype is able to process all the data classes we identified in *Figure 1*. *Figure 5* and *Figure 6* first illustrate how single documents (namely, an SGML tagged text and an image) are transformed using our approach. Then, *Figure 7* shows the output of our prototype when applied to a composite, synthetic document. We designed this document specifically for demonstration purposes. It is actually a web page which content includes XML data, data from a relational database, and an audio file. After processing, all these data are finally described in a single XML document. Note that all the keywords are still manually captured.

SGML document	XML model
<!DOCTYPE lewis SYSTEM	<?XML version=1.0?>
"lewis.dtd">	<!DOCTYPE MlfDt SYSTEM "mlfd.dtd">
<REUTERS TOPICS="YES"	<COMPLEX_OBJECT>
LEWISSPLIT="TRAIN"	<OBJ_NAME>PressRelease</OBJ_NAME>
CGISPLIT="TRAINING-SET"	<DATE>2001-05-15</DATE>
OLDID="12509" NEWID="326">	<SOURCE>Reuters</SOURCE>
<DATE>2-MAR-1987 06:41:06</DATE>	<SUBDOCUMENT>
<PLACES><D>france</D></PLACES>	<DOC_NAME>SGMLdoc</DOC_NAME>
<COMPANIES>SNCF</COMPANIES>	<TYPE>Tagged text</TYPE>
<TEXT>	<SIZE>820</SIZE>
<TITLE>SNCF ISSUING THREE BILLION	<LOCATION>reuter.sgml</LOCATION>
FRANC DOMESTIC BOND</TITLE>	<LANGUAGE>English</LANGUAGE>
<DATELINE>March 2</DATELINE>	<KEYWORD>France</KEYWORD>
<BODY>The French state railway	<KEYWORD>SNCF</KEYWORD>
company, the Ste Nationale des	<TEXT>
Chemins de Fer Francaise (SNCF),	<NB_CHAR>790</NB_CHAR>
is issuing a three billion French	<NB_LINES>12</NB_LINES>
franc domestic bond in two	<TAGGED_TEXT>
tranches, the bond issuing	<CONTENT>*The document could*
committee said. Details of the	*be reproduced here as CDATA*
issue will be announced later and	</CONTENT>
it will be listed in the Official	</TAGGED_TEXT>
Bulletin (BALO) of March 9.	</TEXT>
REUTER </BODY> </TEXT>	</SUBDOCUMENT>
</REUTERS>	</COMPLEX_OBJECT>

Figure 5. Sample physical model for a tagged text

Image	XML model

User-prompted keywords:
– scissors
– black
– white
	`<?XML version=1.0?>`
	`<!DOCTYPE MlfDt SYSTEM "mlfd.dtd">`
	`<COMPLEX_OBJECT>`
	` <OBJ_NAME>Sample image</OBJ_NAME>`
	` <DATE>2001-06-15</DATE>`
	` <SOURCE>Local</SOURCE>`
	` <SUBDOCUMENT>`
	`  <DOC_NAME>Scissors</DOC_NAME>`
	`  <TYPE>Image</TYPE>`
	`  <SIZE>24694</SIZE>`
	`  <LOCATION>sciss.bmp</LOCATION>`
	`  <KEYWORD>scissors</KEYWORD>`
	`  <KEYWORD>black</KEYWORD>`
	`  <KEYWORD>white</KEYWORD>`
	`  <IMAGE>`
	`   <FORMAT>Bitmap</FORMAT>`
	`   <COMPRESSION/>`
	`   <WIDTH>256</WIDTH>`
	`   <LENGTH>192</LENGTH>`
	`   <RESA>100dpi</RESA>`
	`  </IMAGE>`
	` </SUBDOCUMENT>`
	`</COMPLEX_OBJECT>`

Figure 6. Sample physical model for an image

```
<?XML version=1.0?>
<!DOCTYPE MultiformData SYSTEM "mlfd.dtd">
<COMPLEX_OBJECT>
   <OBJ_NAME>Sample composite document</OBJ_NAME>
   <DATE>2002-01-15</DATE>
   <SOURCE>Local</SOURCE>
   <SUBDOCUMENT>
      <DOC_NAME>XML Data</DOC_NAME>
      <TYPE>Tagged text</TYPE>
      <SIZE>675</SIZE>
      <LOCATION>tagdoc.xml</LOCATION>
      <KEYWORD>Tag</KEYWORD>
      <KEYWORD>Document</KEYWORD>
      <TEXT>
         <NB_CHAR>675</NB_CHAR>
         <NB_LINES>22</NB_LINES>
         <TAGGED_TEXT>
            <CONTENT>tagdoc.xml</CONTENT> <!-- Too large to store -->
         </TAGGED_TEXT>
      </TEXT>
   </SUBDOCUMENT>
   <SUBDOCUMENT>
      <DOC_NAME>DB Data</DOC_NAME>
      <TYPE>Relational View</TYPE>
      <SIZE>184320</SIZE>
      <LOCATION>bd1.mdb</LOCATION>
      <KEYWORD>Courses</KEYWORD>
      <RELATIONAL_VIEW>
         <QUERY>select * from course</QUERY>
         <ATTRIBUTE>
            <ATT_NAME>Course</ATT_NAME>
            <DOMAIN>VARCHAR</DOMAIN>
         </ATTRIBUTE>
         <ATTRIBUTE>
            <ATT_NAME>Teacher</ATT_NAME>
            <DOMAIN>VARCHAR</DOMAIN>
         </ATTRIBUTE>
         <TUPLE>
            <ATT_NAME_REF>Course</ATT_NAME_REF>
            <VALUE>Maths</VALUE>
            <ATT_NAME_REF>Teacher</ATT_NAME_REF>
            <VALUE>Smith</VALUE>
         </TUPLE>
         <TUPLE>
            <ATT_NAME_REF>Course</ATT_NAME_REF>
            <VALUE>Computer Science</VALUE>
            <ATT_NAME_REF>Teacher</ATT_NAME_REF>
            <VALUE>Wesson</VALUE>
         </TUPLE>
      </RELATIONAL_VIEW>
   </SUBDOCUMENT>
   <SUBDOCUMENT>
      <DOC_NAME>Sound sample</DOC_NAME>
      <TYPE>Sound</TYPE>
      <SIZE>31773</SIZE>
      <LOCATION>jazzsong.wav</LOCATION>
      <LANGUAGE>English</LANGUAGE>
      <KEYWORD>Music</KEYWORD>
      <KEYWORD>Jazz</KEYWORD>
      <CONTINUOUS>
         <DURATION>20</DURATION>
         <SPEED/>
      </CONTINUOUS>
   </SUBDOCUMENT>
</COMPLEX_OBJECT>
```

Figure 7. Sample physical model for a composite document

5.4 Mapping into a relational database

The mapping of XML documents into a (MySQL) relational database is achieved with a prototype baptized `xml2rdb`[3]. This PHP script operates in two steps. First, a DTD parser exploits our logical model (*Figure 2*) to build a relational schema, i.e., a set of tables in which any valid XML document (regarding our DTD) can be mapped. To achieve this goal, we mainly used

[3] `xml2rdb` PHP prototype URL: *http://bdd.univ-lyon2.fr/xml2rdb*

the techniques presented in Anderson et al., 2000, and Kappel, Kapsammer, and Retschitzegger, 2000. Note that our DTD parser is a generic tool: it can operate on any DTD. It takes into account all the XML element types we need, e.g., elements with +, *, or ? multiplicity, elements lists, selections, etc. The last and easiest step consists in loading a valid XML document into the previously build relational structure.

6. CONCLUSION AND FUTURE ISSUES

We presented in this chapter a modeling process for integrating multiform data into a Decision Support Database such as a data warehouse. Our conceptual UML model represents a complex object that generalizes the different multiform data that can be found on the web and that are interesting to integrate in a data warehouse as external data sources. Our model allows the unification of these different data into a single framework, for purposes of storage and, maybe more importantly, preparation for analysis. Data must indeed be properly "formatted" before OLAP or data mining techniques can apply to them.

Our UML conceptual model is then directly translated into an XML schema (DTD or XML-Schema), which we view as a logical model. The last step in our (classical) modeling process is the production of a physical model in the form of an XML document. XML is the format of choice for both storing and describing the data. The schema indeed represents the metadata. XML is also very interesting because of its flexibility and extensibility, while allowing straight mapping into a more conventional database if strong structuring and retrieval efficiency are needed for analysis purposes.

The perspectives opened by this work are numerous. First, in our next step, we will have to integrate the documents produced by our application into a data warehouse. Now that the XML documents we produce are mapped into an ODS, we have to integrate them in the particular architecture of a multimedia data warehouse (Thuraisingham, 2001).

Our XML modeling could also be improved by taking advantage of the features proposed in XML-Schema (Fallside, 2001) that are not supported by DTDs, such as typing and inheritance, which we have not taken into account yet. In other respects, our XML formalization may also be considered as first-level logical modeling. A multidimensional representation with dimensions and measures would make up the second level and allow the warehousing and analysis of multiform data. This second modeling level has not been discussed in this chapter, but it is one of our main goals for the integration of web data in a data warehouse. Note that it is difficult to devise dimensions and measures from multimedia documents without some knowledge of their content. As far as we know, extracting the semantics of a multimedia document is still largely an open problem and lots of research focus on this issue. The results of this research shall help us a lot in our modeling task.

Next, we do not envisage data mining as a front-end tool only. We believe integrating multiform data in a data warehouse requires more than a simple ETL: it requires intelligence. Using data mining techniques may be envisaged along three axes.

First, data mining can help building a multimedia data warehouse. For instance, it can outline the relevance of indices or materialized views. The physical reorganization of warehoused multiform data can also be automatically triggered if their frequent usage is monitored. For instance, new disk clusters could be devised to improve the performance of multidimensional queries. Eventually, mining some multimedia data may help building indicators (measures) to be analyzed along certain axes (dimensions). Viewed as a pre-processing phase, this task could also save time for ulterior analysis.

Second, the diversity of multiform data necessitates operators that are adapted to their nature. One key question is: how to aggregate non-additive data such as multimedia data? OLAP operators could be extended to include data mining tasks able to achieve such aggregation. Clustering techniques could be used, for example.

Lastly, the analysis of multiform data with data mining techniques could be reinforced by enhancing the navigation into these data (in a multidimensional context) with the help of OLAP-like operators.

REFERENCES

Abiteboul, S., et al. (1997). The Lorel query language for semistructured data. *International Journal on Digital Libraries* 1(1), 68–88.

Anderson, R., et al. (2000). *Professional XML Databases*. Wrox Press.

Bertino, E., Catania, B., and Zarri, G.P. (2001). *Intelligent Database Systems*. Addison Wesley.

Bray, T., et al., Eds. (2000). Extensible Markup Language (XML) 1.0 (Second Edition). *http://www.w3.org/TR/2000/REC-xml-20001006*.

Busse, S., et al. (1999). Federated Information systems: Concepts, Terminology and Architectures. *Forschungsberichte des Fachbereichs Informatik* 99(9).

Ceri, S., et al. (1999). XML-GL: a graphical language for querying and restructuring XML documents. *International World Wide Web Conference*, Canada.

Chaudhuri, S., and Dayal, U. (1997). Data Warehousing and OLAP for Decision Support. *ACM SIGMOD International Conference on Management of Data (SIGMOD 97)*, Tucson, USA, 507–508.

Christophides, V., et al. (1994). From Structured Documents to Novel Query Facilities. *ACM SIGMOD International Conference on Management of Data*, Minneapolis, USA, 313–324.

Cover, R. (2001). XML Metadata Interchange (XMI). *http://xml.coverpages.org/xmi.html*.

Deutsch, A., Fernandez, M., and Suciu, D. (1999). Storing semistructured Data with STORED. *ACM SIGMOD International Conference on Management of Data*, Philadelphia, USA, 431–442.

Deutsch, A., et al. (1999). XML-QL: A Query Language for XML. *International World Wide Web Conference*, Canada.

Deutsch, A., et al. (1999). Querying XML Data. *IEEE Data Engineering Bulletin* 22(3), 10–18.

Edmonds, A. (2001). A General Background to Supervised Learning in Combination with XML. Technical paper, Scientio Inc. *http://www.metadatamining.com*.

Fallside, D.C., ed. (2001). XML Schema. *http://www.w3.org/TR/xmlschema-0/*.

Fernandez, M., et al. (1998). Catching the Boat with Strudel: Experiences with a Web-Site Management System. *ACM SIGMOD International Conference on Management of Data*, Seattle, USA, 414–425.

Fernandez, M., Marsh, J., and Nagy, M., Eds. (2001). XQuery 1.0 and XPath 2.0 Data Model. *http://www.w3.org/TR/query-datamodel/*.

Florescu, D., and Kossmann, D. (1999). Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin* 22(3), 27–34.

Hackathorn, R. (2000). *Web farming for the data warehouse*. Morgan Kaufmann.

Hopmann, A., Berkun, S., Hatoun, G. (1997). Web collections using XML. *http://www.w3.org/TR/NOTE-XMLsubmit.html*.

Hsiao, D. (1992). Federated Databases and Systems: Part I – A Tutorial on Their DataSharing. *VLDB Journal* 1(1), 127–179.

Hsiao, D. (1992). Federated Databases and Systems: Part II – A Tutorial on Their Resource Consolidation. *VLDB Journal* 1(2), 285–310.

Inmon, W.H. (1996). *Building the Data Warehouse*. John Wiley & Sons.

Kappel, G., Kapsammer, E., and Retschitzegger, W. (2000). X-Ray – Towards Integrating XML and Relational Database Systems. *19th International Conference on Conceptual Modeling*, 339–353.

Kimball, R. (1996). *The data warehouse toolkit*. John Wiley & Sons.

Kimball, R., and Mertz, R. (2000). *The Data Webhouse: Building the Web-enabled Data Warehouse*. John Wiley & Sons.

McHugh, J., et al. (1997). Lore: A Database Management System for Semi-structured Data. *SIGMOD Record* 26(3), 54–66.

Miniaoui, S., Darmont, J., and Boussaid, O. (2001). Web data modeling for integration in data warehouses. *First International Workshop on Multimedia Data and Document Engineering (MDDE 01),* Lyon, France, 88–97.

Nestorov, S., Abiteboul, S., and Motwani, R. (1998). Extracting Schema from Semistructured Data. *ACM SIGMOD International Conference on Management of Data*, Seattle, USA, 295–306.

OMG. (1999). *Unified Modeling Language Specification*, Version 1.3. Object Management Group, Inc.

Rakow, T.C., Neuhold, E.J., and Löhr, M. (1995). Multimedia Database Systems – The Notions and the Issues. *Datenbanksysteme in Büro, Technik und Wissenschaft BTW*, GI-Fachtagung, Dresden, 1–29.

Robie, J., Lapp, J., and Schach, D. (1998). XML Query Language (XQL). *http://www.w3.org/TandS/QL/QL98/pp/xql.html*.

Shanmugasundaram, J., et al. (1999). Relational Databases for Querying XML Documents: Limitations and Opportunities. *25th International Conference on Very Large Data Bases (VLDB 99),* Edinburgh, Scotland, 302–314.

Shanmugasundaram, J., et al. (2001). A General Technique for Querying XML Documents using a Relational Database System. *SIGMOD record* 30(3), 302–314.

Shoens, K., et al. (1993). The Rufus System: Information Organization for Semi-Structured Data. *19th International Conference on Very Large Data Bases*, Dublin, Ireland, 97–107.

Tan, A.H. (1999). Text Mining: The state of the art and the challenges. *PAKDD 99 Workshop on Knowledge discovery from Advanced Databases (KDAD 99),* Beijing, China, 71–76.

Thuraisingham, B. (2001). *Managing and Mining Multimedia Databases*. CRC Press.

Wu, M.C., and Buchmann, A.P. (1997). Research Issues in Data Warehousing. *BTW '97*, Ulm.

Zhang, J., Hsu, W., and Lee, M.L. (2001). An Information-Driven Framework for Image Mining. *12th International Conference on Database and Expert Systems Applications (DEXA 2001),* Munich, Germany; *LNCS* (2113), 232–242.

Zwol, R., Apers, P., and Wilschut, A. (1999). Modelling and querying semistructured data with MOA. *Workshop on Query processing for semistructured data and non-standard data formats.*