Bidirectional Conversion between XML Documents and Relational Data Bases

Marta H. Jacinto[1], Giovani R. Librelotto[2], José C. Ramalho, Pedro R. Henriques
Departamento de Informática – Universidade do Minho
Campus de Gualtar – 4710-057 – Braga – Portugal
marta.jacinto@itij.mj.pt, {grl,jcr,prh}@di.uminho.pt

Abstract

Relational databases have been used as support to the information storage since a long time ago. Maintenance and the natural growing of information systems imply the reuse of existing databases but, many times, information is not stored on the most convenient structure or in the desired platform; then it becomes necessary to export it into another system or to transform its structure. These tasks should be executed in a fast and safe way, and as much automatized as possible.

In the last years, XML was accepted as the neutral format for information representation. This is due, mainly, to two factors. On one hand, XML documents are purely textual files, structured and independent of any hardware or software platforms. On the other hand, more and more public domain tools are available, to help users transforming XML documents.

The paper focus on the use XML as an interchange format between database management systems and presents an XML language, DBML, that allows the definition of the structure and the content of a relational database. It also introduces a methodology to convert a database into a DBML document and two tools that allow transforming DBML documents, into SQL and back.

DBML and the two conversion functions (dbsql2dbml and dbml2dbsql) provided the basis to support the idea of designing a language to specify database transformations and developing a generator to automatically produce the transference processors; we finish the paper with the proposal of such a system.

1. Introduction

The work reported in this paper started some time ago in the context of a national research project called *metamedia*. This project aims at the definition of a schema to describe a multimedia archive, handling data in different formats like: text, image, sound, and video. While one of the partners conceived and defined the model in UML and used a relational database to implement the information repository, our participation was guided by the XML approach and the link between both.

In the sequence of this project, we pursued the investigation in this promising area. The idea of using the XML technology came out from the need to establish a dialogue between some applications supporting similar information in different formats. As it will be seen, XML supplies a neutral platform for the information description. As the information was stored in relational databases, we have decided to start creating tools to export the information from a database into XML, and to import XML documents into any relational database management system. These two tools enabled us to transform data from one database into another using XML as the intermediate representation.

This methodology is not specific of the metamedia application that was in its origin. In fact, it is generic and may be applied to other situations where the interchange of information between different applications is needed.

The article starts with a contextualization about XML and its importance in the information world (section 2). In section 3, we will address the problem of the relationship between XML and relational databases (RDBs). The discussion will focus the concepts of structured and half-structured information. In the following section (section 4, the core of the paper), the DBML language is introduced, as well as the transformations of a DB into an XML document (*dbsql2dbml*: SQL => DBML phase) and the reverse process that rebuilds the DB from an XML description (*dbml2dbsql*: DBML => SQL phase). The article finishes with a section that briefly synthesizes the presented work, enhancing the results achieved, and describes the goals for future work; in this concluding section we propose a generation system to automatize the transference of data between databases that can be applied to update legacy systems.

2. Concepts

This section describes the XML standard approach to documents' markup (method and metalanguage). It

[1] Currently working for ITIJ — the Computer Department of the Ministry of Justice
[2] Sponsored by CNPq grants 200339/01-0 - Brazil

gives a perspective of its usefulness in the integration with other traditional applications.

2.1. XML

XML, Extensible Markup Language, is a subset of the Standard Generalized Markup Language (SGML) defined in the ISO standard 8879:1986. XML was designed to make it easy to interchange structured documents over the Internet. XML files always clearly mark the start and the end of each of the logical parts (called elements) of an interchanged document. It also defines how Internet Uniform Resource Locators can be used to identify component parts of XML data streams.

Defining the role of each valid element of a text using a formal model, known as Document Type Definition (DTD), XML users can check that each component of a concrete document occurs in a valid place within the interchanged data stream.

An XML DTD allows computers to check, for example, that users do not accidentally enter a third-level heading without having entered first a second-level heading, something that can not be checked using the HyperText Markup Language (HTML).

However, unlike SGML, XML does not require the presence of a DTD. If no DTD is available, either because all or part of it is not accessible over the Internet or because the user failed to create it, an XML system can assign a default definition for undeclared components.

XML allows users to provide processing control information to support programs, such as document validators and browsers; bring multiple files together to form compound documents; identify the places where illustrations are to be incorporated into text files, and the format used to encode each illustration.

It is important to note, however, that XML is not a predefined set of tags (similar to those defined in HTML) to markup all the existing documents. XML was not designed to be a standard way of coding text: in fact it is impossible to devise a single coding scheme that would be suitable for all applications.

Instead, XML is a formal language that can be used to pass information about the component parts of a document; that is, XML is a metalanguage to define special purpose markup systems. XML is flexible enough to allow the description of any logical text structure (a form, letter, report, book, dictionary or database, etc.).

2.2. XML: the connector link in the information society?

Nowadays, in the area of information systems, almost all phases of an application's life cycle are automatized. At this point, a problem emerged from the great variety of support applications, each one having its format, usually proprietary.

Let us take, for instance, a functional architecture consisting of a case tool like *Rational Rose* (to draw UML diagrams), a development tool like *Visual Age for Java*, a database management system like *SQL-Server2000*, documentation generated in PDF, and a repository of software components written in C.

If we want all the applications to dialogue with each other, we will have to develop *n*n-n* converters, where *n* is the number of different applications. On the other hand, if we have a neutral intermediate representation, we only need *n*2* converters.

This fact is not new and motivated the creation of a specific group inside the OMG (Object Management Group) to study the problem [7]. As a result of this effort, a standard called XMI (XML Metadata Interchange) was proposed [3]. XMI is not more than an XML language.

After studying that proposal, we verified that it was very complex and it was still in an embryonic state for it only establishes a meta-information skeleton that shall involve the information to be interchanged. In what concerns the information itself, nothing is implied, the user is supposed to do what he/she wants.

This lack of a simple and clean way to describe a DB structure in XML and to translate automatically a DB into an XML document in both directions, motivated the work described in this article.

3. XML and RDBs: what is the relation?

As it will be shown, concerning information representation, XML is more embracing than a RDB. This means that it is always possible to represent the information contained in a RDB in an XML document, but the reverse is not always true.

To justify this point of view we will discuss and compare two important concepts: *structured information* and *half-structured information*.

3.1. Structured Information

RDBs can be pointed as the best example of structured information. In a RDB, the information is structured in tables which are, by its turn, organized in lines (records) and columns (record fields or attributes).

This rigid structure provides some advantages such as: the access is easy and fast; the information can be validated and reused for different kinds of results.

Nevertheless, this organization has some drawbacks. For the present work, the most important one is the lack of order in the record fields; in the description of the structure of a RDB, nothing indicates their order. For the typical database applications, this aspect does not matter; yet, when we are storing data which obeys to a linear order, and this order has to be preserved, we face a problem.

3.2. Half-Structured Information

Usually, the concept of half-structured information appears associated to documents. In the context of descriptive markup, of which XML is a practical application, a document has a logical structure made explicit by the inclusion of marks in the text. Although a document, as a whole, is a structured piece of information, many of its parts are half-structured, that is, they are composed by unstructured text and tagged-elements interleaved in an almost free way.

To clarify these concepts, we present the DTD for a class of documents known as poem (Example 1).

```
<!ELEMENT poem (title, author, body, date) >
<!ELEMENT title (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT body (quatrain|tercet)+>
<!ELEMENT quatrain (verse, verse, verse, 
verse)>
<!ELEMENT tercet (verse, verse, verse)>
<!ELEMENT verse (#PCDATA|name)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT date (#PCDATA)>
```

Example 1: DTD for a simple poem

This specification describes the structure that the documents of type poem must follow: the poem has a title, followed by the author, the body and, finally, the date; the title, the author and the date are free text components; the body is, by its turn, a structured part composed by one or more *quatrains* or *tercets*, that are made up of verses.

Apparently we are before a rigid structure. But this is false: there is an element which definition allows a mixed-content; a *verse* is made up of free text in the middle of which *name* elements may occur, in any number and position.

These mixed-content elements and the free text elements allow us to state that an XML document is a half-structured information container.

At this point we may conclude that there are two impediments to the transposition of the information from any XML document into a RDB. The first one is the existence of elements with a mixed-content. The other one has to do with the nature of text. A text has a linear order which we need to keep; changing this order will either change or destroy its meaning.

The RDBs do not have efficient mechanisms to deal with linear order nor with half-structured data. On the other hand, the XML coexists with structured information, half-structured information, and even linear orders. We are, therefore, before a strong candidate for the constitution of an information interchange platform.

Another factor that supports the use of XML as an interchange platform is the huge amount of available tools to validate and process XML documents; also the fact that new tools can be easily developed.

4. Converting XML Documents from and into RDBs

The conversion problem shall be divided in two parts: the conversion of a RDB into XML; and the conversion of a specific XML (*DBML – Data Base Markup Language*) into a RDB.

The second can be solved generically. The XML is a neutral format and provided the DBML document contains all the information needed, it is possible to generate the SQL text that replicates the original RDB in any Database Management System (DBMS) that accepts SQL.

To solve the first sub-problem there are two basic approaches, besides, it can only be generalized in methodological terms. Once DBMSs use different internal representations, it is necessary to create a specific *front-end* for each one of the internal representations. In the work here reported, we developed a *front-end* for the SQL-Server2000, but plan, in short term, to define *front-ends* for Access, Oracle, MySQL and PostGres.

As discussed in [10], we have two basic choices to represent a RDB in XML: the first one (adopted in this project) consists in the definition of a generic DTD that contains the appropriate elements to describe the data and also the DB structure (that is, the tables and the columns – names and properties); on the opposite side, the second approach consists in the definition of a specific DTD for each DB – that DTD will contain the necessary elements to represent the data already with the specific table and column names and the DTD itself describes the DB structure (there are elements which names describe the tables and columns for that specific DB).

In the first case, the conversion process is more complex and heavier because the XML document that will be produced is longer (before representing the values stored in the record fields of the DB tables, the description of the DB itself must be generated). However, the DTD is always the same (defined just once) and all the tools can be re-used. In the second alternative, the conversion is more efficient (we just need to translate the data) but a new DTD must be tailored for each different DB structure.

In the next subsections, we will illustrate the proposed methodology, with a small example: both the conversion and the DBML language developed will be introduced.

4.1. From a RDB into XML

Whenever we intend to represent an entity in XML, we must remember that, we not only want to represent the information itself but also to describe its properties (the so-called meta-information). In that case, we want also to describe the way data is stored in the RDB.

A RDB has two components: structure and information. Our final XML document will have, therefore, two parts, one to describe the structure of the RDB and the other one to store the information of that

RDB. Therefore, the XML skeleton of the final document will be:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<DB name="XXX" date="today">
    <STRUCTURE>
        ...
    </STRUCTURE>
    <DATA>
         ...
    </DATA>
</DB>
```

Example 2: XML Skeleton: general structure

The conversion of a RDB into XML will, then, consist of two parts:

- **Structure:** To fill the first part of the XML document it will be necessary to access the structural information of the RDB and to convert that information into an adequate XML text. Each Database Management System has a particular way to keep the structural description of a database. Thus, it will be necessary to create a converter for each platform. In the rest of this paper, we assume SQL-Server2000 as the working platform.
- **Data:** The transference of information from a RDB to XML may be a more generic process. Every Database Management System allows to download the information of a database to a pure text file using pre-defined field and record separators. Therefore, the conversion problem to solve is the one of converting data from those text files to the second part of the final XML document.

Lee Buck [4] and Ronald Bourret [1] describe two possible approaches to data codification. However, there is no information available on the structure description; even in commercial projects, like CARD, nothing is said about the philosophy and methodology adopted. In our case, we decided to develop an XML markup language (DBML) to describe the structure. We develop these two topics in the following subsections.

4.1.1. Structure Conversion. As told previously, this conversion consists in processing the text description of the database structure generated by the RDBMS. These descriptions have the general form shown bellow.

```
... CREATE TABLE [dbo].[District] (
    [code] [int] NOT NULL
    [name] [nvarchar] (50) ...
    [country] [int] NOT NULL
) ON [PRIMARY] GO ... ALTER TABLE
[dbo].[Districts] WITH NOCHECK
ADD
    CONSTRAINT [PK_Districts] PRIMARY KEY
    CLUSTERED
    (
        [code]
      ) ON [PRIMARY]
   GO ...
```

Example 3: Internal description of a DB

Christiansen and Torkington [5] developed, in Perl, a parser which, from a file like the one shown above, generates the correspondent description in XML.

In Example 4, we describe the translation scheme adopted by the parser.

Tables: Each table is mapped into an element named *TABLE*, which has an attribute *NAME*.

```
<TABLE NAME="Districts">
   ...
</TABLE>
```

Example 4: XML Skeleton: Tables

Columns: Each column will be mapped to a *COLUMN* element that also has a *NAME* attribute, where the column's name is saved. Other properties like the data type for the values of that column and the characteristic of being empty or not, are stored in attributes *TYPE* and *NULL* associated to the *COLUMN* element.

As a table contains more than one column, it is necessary to include in the XML document another element, *COLUMNS*, to aggregate all the *COLUMN* instances.

```
<COLUMNS>
   <COLUMN NAME="code" TYPE="int" NULL="no"/>
      ...
</COLUMNS>
```

Example 5: XML Skeleton: Columns

Primary and Foreign Keys: Keys are defined inside a table definition; so it will be described as a sub-element of the *TABLE* element. An aggregate element, *KEYS*, has to be introduced to gather the various keys of a table. The set of *KEYS* shall also be divided into Primary and Foreign Keys; so *PKEY* and *FKEY* were introduced as sub-elements of *KEYS*.

Moreover, a primary key in the relational model may be single (just one column) or compound (more than one column). To distinguish these two cases, an attribute *TYPE* was associated to *PKEY* element, as shown below.

As foreign keys (just of single type) relate one table with another one, the *PKEY* element, shall be associated to the attributes *IN* (identifier of the destination table) and *REF* (identifier of the linked fields in the destination table).

```
...
  <KEYS>
    <PKEY TYPE="simple">
      <FIELD NAME=""/>
    </PKEY>
    <PKEY TYPE="compound">
      <FIELD NAME=""/>
      <FIELD NAME=""/>
    </PKEY>
    <KEY NAME="" REF=""/>
    ...
  </KEYS>
```

Example 6: XML Skeleton: Primary Keys

To illustrate the translation schema just described we will use the structure of the classical DB of Products and Suppliers that contains three tables: two represent the entities Products and Suppliers and the third implements the *N:N* relation “suppliers” between them.

The primary key for tables Products and Suppliers is single and implemented by the field (column) CODE, in both cases; there are no more keys. Concerning the third table, p2s, its primary key is composed and the field elements are *cod-p* and *cod-s*; it also has two foreign keys, *cod-p* and *cod-s* that establish the links to the other two tables. The result of the conversion is shown in the Example 7.

```
<?xml version="1.0" ?>
 <DB>
  <STRUCTURE>
   <TABLE NAME="products">
    <COLUMNS>
     <COLUMN NAME="code" TYPE="nvarchar"
      SIZE="10" NULL="no"/>
     <COLUMN NAME="description" TYPE="nvarchar"
      SIZE="50" NULL="no"/>
     ...
    </COLUMNS>
    <KEYS>
     <PKEY TYPE="simple">
      <FIELD NAME="code"/>
     </PKEY>
    </KEYS>
   </TABLE>
   <TABLE NAME="p2s">
    <COLUMNS>
     <COLUMN NAME="cod-p" TYPE="nvarchar"
      SIZE="10" NULL="no"/>
     <COLUMN NAME="cod-s" TYPE="nvarchar"
      SIZE="10" NULL="no"/>
    </COLUMNS>
    <KEYS>
     <PKEY TYPE="composite">
      <FIELD NAME="cod-p"/>
      <FIELD NAME="cod-s"/>
     </PKEY>
     <FKEY NAME="cod-p" IN="products"
      REF="code"/>
     <FKEY NAME="cod-s" IN="suppliers"
      REF="code"/>
    </KEYS>
   </TABLE>
   <TABLE NAME="suppliers">
    <COLUMNS>
     <COLUMN NAME="code" TYPE="nvarchar"
      SIZE="10" NULL="no"/>
     <COLUMN NAME="name" TYPE="nvarchar"
      SIZE="60" NULL="no"/>
     ...
    </COLUMNS>
    <KEYS>
      <PKEY TYPE="simple">
        <FIELD NAME="code"/>
      </PKEY>
    </KEYS>
   </TABLE>
  </STRUCTURE>
  <DATA>
    ...
  </DATA>
</DB>
```

Example 7: The Products and Suppliers DB Structure translated into DBML

4.1.2. Data Conversion. As it was told in the beginning of this section, the transference of a DB to a DBML document has two parts: the structure description that was explained in the previous specification; and the data description that will be discussed now. There are two approaches, studied and published [4, 1]:

Via attributes: Each line of the table is mapped into one element and the values of its columns (fields) are mapped into attributes of that element;

Via elements: Each line of the table is mapped into one element and the values of its columns (fields) are mapped into child elements, one for each value.

The two proposals might look equivalent [6], and are in terms of the information representation, but concerning the processing, the options are quite different. The processing of an XML documents is structure-oriented, and the structure is given by the elements. The attributes play a secondary roll. So, the second approach seems to be much more effective. Thus, in this project, the choice was the second one - *Via elements*.

To convert the information from a RDB, into DBML, the following translation schema is used:

1. For each table, an element with the correspondent name is created.

2. For each line (record), an element with the table name and the suffix “-REG” is created.

3. For each column (field) an element with the column name is created. Its content will be the value of that field. Empty fields give origin to elements without content.

The next example (Example 8) illustrates the data conversion principle, using the same Products.Supliers database.

```
...
  <DATA>
    <products>
      <products-REG>
        <code> a122 </code>
        <description> milk </description>
        ...
      </products-REG>
      <products-REG>
        ...
      </products-REG>
    </products>
    ...
  </DATA>
...
```

Example 8: The Products and Suppliers DB data to DBML

4.2. From XML (DBML) into a RDB

Given a DBML document, describing a database structure and content, the problem of regenerating the original database is solved by generating a SQL file.

The SQL statements will then be interpreted by a DBMS and the unique DB will be created.

There are two approaches to XML processing:

Procedural: using a programming language like Omnimark, Balise [9] or XML::DT (a Perl module) [8].

Declarative: using XSLT [2], a declarative language (defined in XML) and designed for XML transformations.

In the context of this project, we have experimented both approaches. So we developed a conversor in XML::DT, as well as another one in XSL.

The second solution has the advantage of being implemented in a neutral and standard platform (an XSL stylesheet is an XML document). It can, therefore, be ported and installed in various operating systems, and also, it satisfies a great community of XML users with some background in XSL. It is possible to have a complete software package to process DBML developed inside of the same paradigm. Below, we show part of the XSL stylesheet used.

```
... <xsl:template match="DATA">
  <!-- For each Table -->
    <xsl:for-each select="child::*">
      <!-- For the first Record -->
        <xsl:for-each select="child::*">
          <xsl:if test="position()=1">
            <!-- For each Field -->
              <xsl:for-each select="child::*">
                <xsl:value-of select="name()"/>
...
```

Example 9: DBML into SQL Translation: Part of XSL stylesheet

So that conversion, in the inverse direction of the one discussed above, from DBML into SQL, is just a traditional-case of XML documents processing: nothing specially new should be developed.

The final result in SQL looks like the following:

```
INSERT INTO products (code, description, ...)
   VALUES('a122', 'milk', ...)
INSERT INTO products (code, description, ...)
   VALUES('a115', 'milk', ...)
```

Example 10: DBML to SQL Translation: SQL Code Generated

In the same way that we developed a processor and a stylesheet to transform a DBML document into SQL code, it is possible to develop processors and stylesheets for different purposes like the transformation of the database in another one with a different structure or filtered content. This project seems to be challenging and very useful in the information systems area; it will be presented in section 5 as future work.

5. Conclusion

The work presented in this paper was developed in the *metamedia* context. In that context, some of the technics and tools developed so far are being applied to support the information interchange between different applications.

The concrete results of the work here reported are:

- a descriptive markup language, DBML, to describe the structure and the content of a database (in the paper we just presented fragments of DBML documents, along the discussion of the mapping – conversion – process, but that language is formally defined by an appropriate DTD);
- a methodology to export a relational database to a DBML document;
- a processor that converts a database (exported in SQL by the SQL-Server 2000) into a DBML document;
- a processor that converts DBML documents back into SQL code.

The possibility to transform both the content and structure of a database, when these are represented in a DBML document, points out a new research direction: as future work, we intend to develop a DB transformer generator, as described in the diagram of Figure 1. We will define an XML language, XDBTL, to describe the desired transformation; then we will develop a generator to compile that description in order to produce an XSL stylesheet to translate the DBML file (corresponding to the source DB) into the new DBML file (corresponding to the target DB).

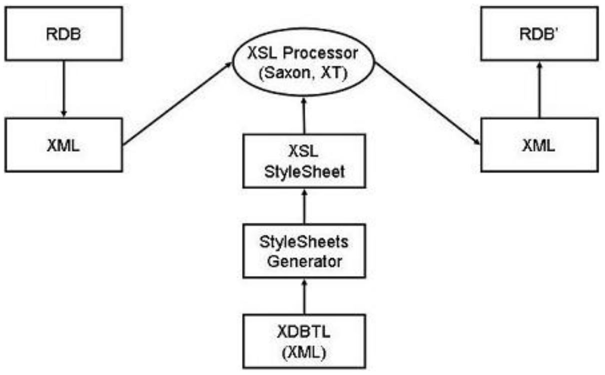

Figure 1: XDBTL and DB Transformer Generator

6. References

[1] R. Bourret. *XML and Databases*. http://www.rpbourret.com/xml/XMLAndDatabases.htm, Nov. 2000.
[2] N. Bradley. *XSL Companion*. Addison-Wesley, 1999.
[3] S. Brodsky. *XMI opens application interchange*. IBM white papers, 1999.
[4] Lee Buck. “Data models as an XML Schema development method”, *XML 99*, Phyladelphia, Dec. 1999.
[5] T. Christiansen and N. Torkington. *Perl Cookbook*. O’Reilly, 1998.
[6] E. Kimber. *Designing a DTD: Elements or attributes?* http://www.oasis-open.org/cover/attrKimber9711.html.
[7] OMG. *XMI production of XML Schema*. Request for Proposal, 2000.
[8] J.C. Ramalho and J.J Almeida. “XML::DT - a Perl down-translation module”, *XML Europe’99*, Granada - Spain, April 1999.
[9] AIS Software. Balise and XML. http://xml.coverpages.org/baliseWhitePaper.html, 1998.
[10] M.H. Jacinto.*”Validação Semântica em Documentos XML”*, MSc Thesis, Universidade do Minho - Portugal, 2002. (to be published)