AXIS – a Java implementation of a data exchange architecture among heterogeneous data sources using XML documents

Sérgio Mergen[1], Carlos Alberto Heuser[1], Eduardo Kroth[2]

[1] Universidade Federal do Rio Grande do Sul Av. Bento Gonçalves, 9500
Porto Alegre - RS -Brasil
{mergen, heuser}inf.ufrgs.br
http://www.inf.ufrgs.br
[2] Universidade de Santa Cruz do Sul Av. Independência, 2293
Santa Cruz do Sul – RS – Brasil
kroth@dinf.unisc.br
http://www.unisc.br/deptos/dinfwww

Abstract. XML is quickly becoming the industry standard data exchange format for business applications. Since most of the data is stored in relational tables, there must be a way to transform relational data into XML. For a XML document to be readable for every business associate relational database, a mapping must be provided to specify how a XML document fits a specific database model. This paper proposes AXIS (Automatic XML Interchange System), which is an architecture implemented in Java that allows the creation of such mapping and, afterwards, the conversion of relational data to XML and back to relational data.

1. Introduction

This paper deals with the data exchange among enterprises that use different schemas and databases. AXIS is an architecture designed to provide the exchange of data among enterprises. This kind of application is convenient in the scenario where heterogeneous databases must be able to import and export data to each other [1].

Since the databases schemas are usually expected to have differences in the data representation, it is not a trivial operation to translate the semantics from one database data to the others.

One of the possible solutions to overcome that situation is to write database specific applications to handle the conversion of data from one database to another. However, this ad hoc solutions are only suited for small, non-scalable business partnerships. If the net of partners tends to grow over time, it is advisable to choose a more flexible, database schema independent solution.

Independence is the key of order when it comes to data exchange. It is important to highlight that not only database schema independency is important in that manner.

Platform independence is also a very desired feature, once it is not uncommon to come with business partners' applications running in different operational systems.

Independence of database technology is another fundamental aspect that must be accomplished by data exchange tools, i.e. data conversion must be possible regardless of the database used by the parts involved in the exchange process.

To achieve all the previous mentioned purposes, one possible approach is to work with a standard data format, used as an intermediary level of communication among different databases. This data format is defined in a common agreement by all enterprises, thus providing the enterprises with the ability to comprehend the whole set of documents that may be generated from any of the enterprises.

To deal with the other issues, like the platform independency, the most common solution is to write applications using a platform independent language.

The combination of Java portability & XML [2] was the choice adopted by AXIS to match the requirements demanded above. XML has become a de facto standard data format [3], and it allows the definition of grammars that can be used to validate documents across enterprises. The grammars are most commonly written in DTDs [2] or XML schemas [4].

The AXIS architecture makes extensive use of DTDs in the process of exchange. In fact, DTDs are a fundamental piece in the architecture, since they are used to map the structure allowed in the XML documents to the database structure. The mappings, as well as the DTDs structures and the databases structures, are store in the AXIS Meta Model.

The information stored in this Meta Model are then consulted when XML documents must be generated from the data of a given database, and when database data must be generated from the data of a given XML document.

The exchange of data does not require that all of the involved parts use the architecture proposed. Since XML is the data format chosen for the intermediary level, and all parts understand the same DTD, each enterprise may use whatever tool is available for the data conversion. The fig. 1 shows the overall architecture proposed by AXIS.

The paper is organized as follows: the next section shows related work about the paper subject Section 3 describes the Data Model of the AXIS architecture. Section 4 presents the software implemented that are used to generate XML documents. Section 5 contains the conclusions.

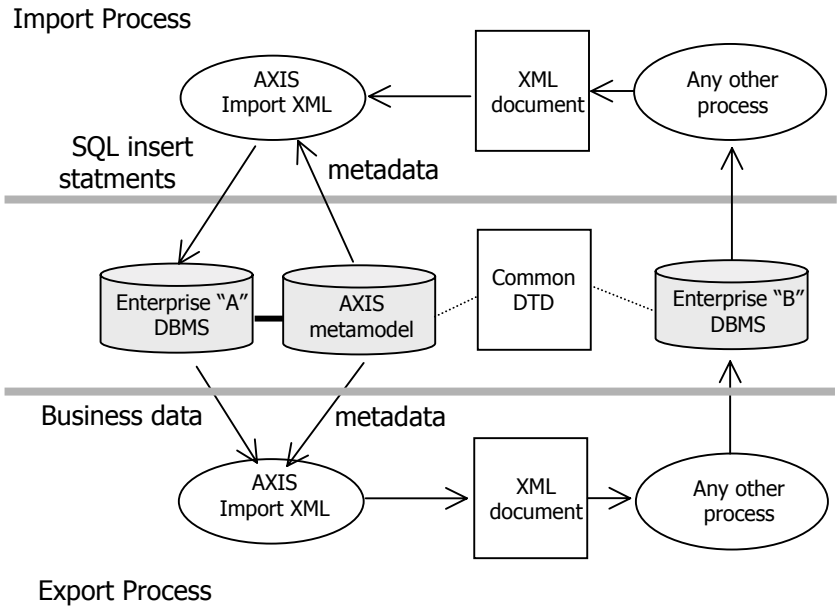


Fig. 1. AXIS: a view of data exchange architecture

2. Related Work

Many of the database vendors are integrating XML capabilities in their products. The so-called XML-enabled databases are mainly concerned in improving the searching performance across structured and unstructured data sets [5].

Oracle's way to manage the conversion of XML data into and out of databases is XML SQL Utility (XSU)[6]. This utility provides an internal set of rules that determine how the transformation is performed. Because of this fixed mapping rules, it is not possible to transform the data if the database and the XML are asynchronous, i.e., have differences in the schemas. In such cases, either the database object-relational model is adjusted through the creation of views, or the XML document is adjusted using XSLT[7].

Microsoft's SQL Server[8] mapping rules use different syntax for updating the database and generating XML documents. The mapping is specified inside the SQL statement. It means that the mapping must be explicitly defined each time a conversion is necessary, and there is not an integrated mechanism for the transformation.

XMLDBMS[9], which is a middleware for transferring data between XML documents and relational databases, employs an architecture similar to the one used in AXIS. It is different because the mapping is stored in a XML document, and it allows the association between XML elements with database tables. The inclusion of database tables in the association limits the integration of database and XML documents with different structures.

IBM's XML Extender[10] allow arbitrary mappings, because, like in AXIS, only columns of database can be mapped to elements/attributes of a DTD. This kind of mapping makes it possible to integrate database schemas and XML schemas with different structures.

The tools described above present different solutions to handle XML document transformation to/from a database, but they all share a concept, that is the mapping between XML documents and databases. This mapping is responsible for various aspects of the data transformation, like flexibility and performance.

3. AXIS Model

This section presents the description of AXIS sub-models (fig. 2): (i) the meta model that stores metadata from XML documents (*XMLMD*), (ii)the meta model that stores metadata from relational databases(*RDBMD*), and the model that maps the first two meta models(*Correspondence Model*).

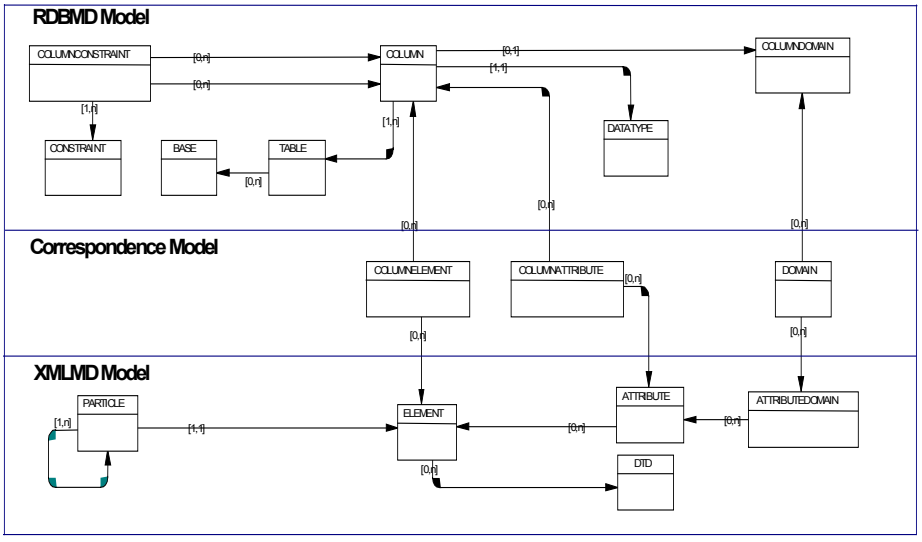


Fig. 2. The AXIS Model

The XML metadata is obtained from the DTDs that are used to validate the XML documents. The *XMLMD* can be basically populated with the elements and the attributes of a DTD.

As will be seen in section 4.1, before a DTD is converted to the *XMLMD*, it must be visible to the *XMLMD* converter algorithm. This is achieved by creating a memory representation of the DTD. This memory representation (a DTD object) is then transformed in relational data. However, the "impedance mismatch"[11] between the object paradigm and the relational paradigm makes the transformation a little bit harder.

Considering the DTD classes' hierarchy, it was decided to create one data entity per class. That was the case for classes like *Element* and *Attribute*, which became respectively the tables *ELEMENT* and *ATTRIBUTE*. However, some other classes demanded more caution. The interface *Particle*, for instance, could be extended to a *Group* class or to a *Reference* class. *Reference* classes simply refer to an *Element* class. *Group* classes are responsible for storing a group of elements. A group is considered to be every collection of elements delimited by round brackets. In the object representation, a *Group* is a collection of *Particles*.

Since a *Group's* collection may contain either another *Group* or a *Reference*, there was no way to create separated tables for the *Group* class and the *Reference* class. So, the two classes were unified in the *PARTICLE* table.

Besides storing the contents of a DTD file, this model can also support the definition of extra values, used to better identify a DTD, such as its description and version.

The relational database metadata is obtained from the schema of the database. The *RDBMD*(fig 2) can be basically populated with the tables and the columns of a RDB. More detailed information, like foreign key references can also be stored in the model.

The last (but not least) sub-model makes use of the metadata stored in the *RDBMD* and in the *XMLMD* to provide a mapping between the two representations of schemas. The *Correspondence Model* maps the columns of tables of the *RDBMD* with the elements/attributes of the *XMLMD*.

The mapping was conceived in a way that only the pieces of structure capable of containing data can be mapped. From the XML perspective, only the elements and attributes can contain data, and from the database perspective, only the fields of tables can contain data.

4. AXIS Java Applications

This section presents an overview of the modules responsible for the generation of XML documents. All the applications were written in Java. Besides its portability, it is a very known programming language, which make of it the best choice for building application with good development support and the aid of powerful utility tools (middleware, wrappers, etc…), developed by third parties, and made available for the java community around the world.

The applications developed in the AXIS project can access any database that provides a JDBC driver. Another alternative is to use the JDBC ODBC Bridge to connect to a database via ODBC. However, the database creation scripts may vary from one database to another. Each database vendor has its own way to deal with some kind of operations, like the definition of auto-increment columns.

The fig. 3 shows how the applications communicate with their APIs to perform its tasks.

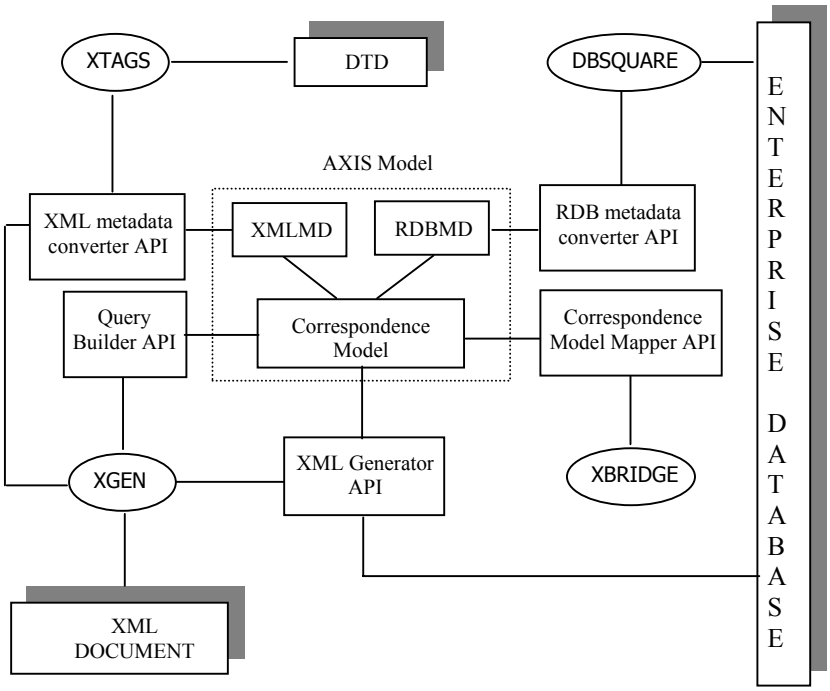


Fig. 3. The modules of the AXIS architecture

4.1 XTAGS and DBSQUARE

Using the AXIS architecture, the user can begin by running the XTAGS software. This software allows the manipulation of DTDs, through the use of a visual editor. Next, these DTDs metadata can be exported to the *XMLMD*.

All the actions performed by XTAGS take place in the XML Metadata Converter API. Besides the conversion of the DTD from XTAGS to *XMLMD* and vice versa, this package also comprises a DTD Parser, used to convert the DTD from a memory representation to its textual description and vice versa. The DTD Parser API [12] was designed by Ronald Bourret. All the other APIS describe in fig. 3 were developed by the AXIS researchers.

Besides the population of the *XMLMD* model, the *RDBMD* model must also contain metadata from a database so both models can be mapped. The DBSQUARE software is used for that purpose. The RDB Metadata Converter API is responsible for collecting the metadata from the Enterprise Database and passing it to the *RDBMD* model.

4.2 XBRIDGE

The XBRIDGE software provides a visual interface that helps the user in the construction of the mappings. This interface hides the complexity of the mapping rules from the user, leaving him with a high level of abstraction of the AXIS model. The mapping rules are defined as follows:

- Content Components: Only the components of a RDB and a XML document that can contain text data are allowed to be mapped, i.e. the elements and attributes of DTDs and the fields of tables of relational databases. It is important to note that not every element of DTDs can contain literal values. EMPTY elements are included in that group. ANY elements are suspicious due it's ambiguity, and therefore are also included in this group.
- Duplicate Mappings: A component can't be mapped twice. E.g. an element "bookName" cannot be mapped to the field "name" of table "book" and to the field "name" of table "author". Usually, cases of duplicate mappings occur due to bad modeling designs or bad data interpretation from the user, which was the case showed in the above example.
- Nullable Components: A component whose content may be null cannot be mapped to a component whose content is mandatory. E.g. an element with a 0-1 frequency (the "?" symbol) can't be mapped to a field of a table whose content is set to NOT NULL.

An attempt to perform any of the mappings described above results in an error message. However, some mappings are allowed by the software, which may result in an incomplete XML generation. Such problems occur when multi-valued components are mapped to mono-valued components. In that kind of occasion, only the first match is translated, and all the others are ignored.

The software does not address this problem, because the generated document is not invalid, only incomplete.

4.3 XGEN

XGEN is the software responsible for generating XML documents taking data from a relational database. The options of the XGEN software are reduced, since most of the work was already performed in XBRIDGE. Therefore, XGEN is considered as a final user tool.

One of the user's assignments is to decide from which enterprise database and from which DTD should the generation actuate. Once this decision is made, the Query Builder is activated. The Query Builder generates a SQL select statement based a set of rules applied over the information collected from the *Correspondence Model*. Such rules are:

- Every fields of tables presented in the mapping are included in the statement. These fields are the ones that will generate the XML content.
- For every table included in the statement, the table's primary keys are also included. Because of the mapping flexibility, it is possible that the fields from the same table are associated to elements belonging to different hierarchy level. So, these primary keys are used by the XML Generator algorithm to separate the data already converted from the data yet to be converted.
- The fields are ordered based on the elements hierarchy on the DTD. Considering the DTD Definition "*<!ELEMENT book (name, year)>*"**,** and knowing that element "name" is mapped to field "name" of table "book" and element "year" is mapped to field "pubYear" of table "book", the select statement ordering is defined as: "*order by book.name, book.pubYear*". This rule is considered as a preventive measure, since it assures the XML document is integrally generated. The omission of this rule could lead to a fragmentized XML document.

The XML Generator API was designed in a way it can be extended. Through the implementation of the *XMLGenerator* interface, the programmer can decide what to do with the data that comes from the Enterprise Database. The *XMLGenerator* interface provides the following methods:

```
public Object getXMLDocument();

public void startDocument();

public void startElement(String elementName);

public void finishElement(String elementName, int
nodeType);

public void setElementValue(String elementValue);

public void setAttribute(String attributeName, String
attributeValue);

public void enterNode();

public void leaveNode();
```

Two implementation of *XMLGenerator* are available from AXIS. The *XMLDOMGenerator* and the *XMLTextGenerator*. The first generates a DOM[13] Tree, while the second generates a string representation of a XML document.

Another task that can be executed by the user is to apply filters on the SQL statement. The filters are used to restrict the data returned from the Enterprise Database.

4. Conclusions

The *Correspondence Model* propitiates a very flexible mapping, allowing that the data from any database schema can be converted in a XML restrained by any DTD schema, and vice versa.

However, such flexibility comes with a performance cost. The AXIS architecture is being improved so the performance does not becomes a burden for enterprise exchange of XML documents.

Unfortunately, it is difficult to overcome the problem of the databases heterogeneity. The *Correspondence Model* handles problems issuing Domain Conflicts, Semantic Conflicts and Structure Conflicts, but some limitations rise, like the data type heterogeneity, e.g. a column that accepts integer values in one database and floating values in another database.

The XML Schema is emerging as a rich schema capable of dealing with this kind of problems. Once the XML Schema specification is completely defined, the *Correspondence Model* will be extended to allow the mapping between database schema and XML Schema.

References

1. Elmagarmid, A.; Rusinkiewicz, M.; Sheth, A.: Management of Heterogeneous and Autonomous Database Systems. San Francisco, California: Morgan Kaufmann Publishers, Inc., (1999). 413p.

2. Martin, D; et al.: Professional XML. Wrox Press. (2000). 1266p.

3. Bradley, N. : The XML Companion. 2.ed. Harlow, USA: Addison-Wesley Longman Limited, (2000). 435p.

4. W3C Consortium. XML Schema. Available at http://www.w3.org/XML/Schema

5. Liefe, H.; et al. Efficient View Maintenance in XML Data Warehouses. Department of Computer and Information Science. University of Pennsylvania, http://www.cis.ipenn.edu/~liefke/papers/

6. Oracle. XML-SQL Utility (XSU), Available at http://otn.oracle.com/docs/tech/xml/oracle_xsu/doc_library/adx04xsu.html

7. Kay, M.: XSLT Programmer´s Reference. Wrox Press Ltd. (2000) . 777p.

8. Microsoft. OPENXML, http://msdn.microsoft.com/library/default.asp?url=/library/enus/tsqlref/ts_oa-oz_5c89.asp

9. Bourret, R. XML-DBMS Middleware for Transferring Data between XML Documents and Relational Databases, http://www.rpbourret.com/xmldbms/

10. IBM. IBM DB2 XML Extender- An end-to-end solution for storing and retrieving XML documents, http://www4.ibm.com/software/data/db2/extenders/xmlext/xmlextbroch.pdf

11. Ambler, S. Mapping objects to relational databases Available at http://www-106.ibm.com/developerworks/library/mappng-to-rdb/

12. Bourret, R. - DTD Parser and Schema Converters. Available at: http://www.rpbourret.com/schemas/index.htm

13. Document Object Model. Available at http://www.w3.org/DOM