Loading XML Data into Relational Databases Architecture of a System Including a (Semi-) Automatic Matching

Boubaker Kahloula[1], Karim Bouamrane[2]
SONATRACH Aval
[1]Division Recherche et Technologie
Oran, Algeria
bkahloula@avl.sonatrach.dz

[2]Université d'Oran, Département Informatique
Oran, Algeria
kbouamranedz@yahoo.fr

ABSTRACT: *Information systems that are widely distributed are characterized by inherent heterogeneity. This is mainly due to the fact that these systems have different purposes, they have been implemented by different developers and they run on different systems platforms. Data stored in these systems are imported and exported permanently from one system to another one to enable them to communicate and exchange information. These systems import also data regularly from other available web applications and services. Data are generally formatted into XML (eXtensible Markup Language) and the target system is most often a relational database. Data contained in XML files with a name and given format must be reformatted to fit with the target format and most of the time they need to be renamed to be stored and used at within the target schema of the database. In this paper, we present the architecture of a system that allows loading XML data into a relational database. We will focus more in detail on the data integration module that is responsible of matching the imported data with the local database schema.*

Keywords: Relational Databases, XML data, Database schema, Web services

Received: 11 November 2010, Revised 28 December 2010, Accepted 31 December 2010

1. Introduction

Modern exchanges of information over the web between databases, applications and services are done through XML, a file formatting language which prevails nowadays. Although XML presents many advantages over the so-called "flat files" used in the past, it does not have any clear solution with regard to the semantic processing of data and the way data schemas originating from two different databases match each other. XML data needs to undergo some processing before being loaded into the target relational database. In this paper we propose to present the overall architecture of a system that meets the following requirements:

• The input data are contained in an XML file. The XML file needs to be "well-formed", that is to say any open tag must have its corresponding closing tag. The file does not have to follow a specific DTD (Document Type Definition) nor a XSD (XML Schema Definition).

• The target database is a relational database. The data contained in the source file are dispatched over the corresponding attributes in the different tables of the target database.

• The processing and loading are done through SQL (Structured Query Language) queries.

• A matching between the input schema and the database schema must take place (semi-) automatically. The final mapping file is generated after validating the results of matching by the system user. It is worth mentioning that existing systems, proprietary or open source software, do not provide automatic matching, this aspect is still at the stage of prototypes such as Microsoft CUPID Research [1], TranScm [2] or DIKE [3].

• The portability of the system on another platform (operating system and database) is guaranteed as the programs are written in Java and the formulation of SQL queries inside the programs correspond to the SQL Standard (SQL'92) [4]. The system can be installed on any machine and is compatible with any relational database (Oracle, SQL-Server, DB2, MySQL, etc.).

We first present in this article the overall architecture of the system then we will give a more detailed description of each of its components. We will then discuss more specifically the matching and will consider the difficulties in case where it is done manually. Finally we will expose the (semi-)automated matching solution chosen for this system, partly based on the use of dictionaries and thesauri on one side, and the use of mapping decisions taken by the user in the past, on the other side.

2. Overall Architecture

The overall system architecture is depicted in Fig. 1. The system includes following modules:

• The Trigger, which automates the entire process

• The Matcher, whose function is to create a Mapping File

• The Loader, whose task is to load the XML file into a transient table

• The Rules Engine, which inserts the data contained in the transient table in the target database

• The Archiver, whose role is to archive a file already processed by the Loader.

The processing occurs as follows: First the XML file is loaded by the Loader into the transient table. The loading is done according to a mapping generated (semi-) automatically by the Matcher on the basis of a matching algorithm. In the mapping file is described the correspondence between elements of the XML file and the attributes of the transient table. As soon as the data are loaded into the transient table, the XML file is archived. In a second step, data contained in the transient table are, read and distributed among different tables in the target database. This action is accomplished by the Rules Engine. Throughout the process, information and / or error messages are recorded in a log file and eventually sent to the system administrator.

3. System components

The different system components are described below:

3.1 The Trigger

The trigger depends mainly on the operating system on which it is installed. It works as a daemon and a shell for UNIX (or Linux), otherwise if the operating system is Windows then the Trigger is a task planner and a batch file.

The daemon for UNIX/Linux or the task planner for Windows will launch the shell or the batch file with the desired frequency. The frequency is chosen by the user according to his needs. The shell or batch file includes the call sequence of the different programs: Matcher, Loader, Archiver and Rules Engine.

3.2 The Matcher

The role of the Matcher is to create, on the basis of a matching algorithm, the mapping file. Based on this mapping file the Loader will load the XML data into the columns of the transient table.

3.3 The Loader

The function of the loader is to load the XML data into a table, named “transient table”, of any supplier's relational database

(Oracle, SQL Server, DB2, MySQL, etc...). The term “transient table” is due to the fact that the data will reside in this table temporarily. The Loader relies on the mapping file, previously created by the Matcher, to load XML data into the corresponding columns of the transient table. The Loader uses the libraries of Hibernate framework [5]. An example of mapping file is given below:

```
<?XML version=”1.0"?><!DOCTYPE hibernate-Mapping PUBLIC
“-//Hibernate/Hibernate Mapping DTD 3.0//EN”
“http://hibernate.sourceforge.net/hibernate-Mapping-3.0.dtd”>
<hibernate-Mapping>
 <class entity-name=”catalog” node=”catalog” table=”CATALOG”>
   <id name=”catalogId” node=”catalogId” column=”CATALOGID” type=”string”/>
   <property name=”journal” node=”journal” column=”JOURNAL” type=”string”/>
   <property name=”publisher” node=”publisher” column=”PUBLISHER” type=”string”/>
   <property name=”edition” node=”edition” column=”EDITION” type=”string”/>
   <property name=”title” node=”title” column=”TITLE” type=”string”/>
   <property name=”author” node=”author” column=”AUTHOR” type=”string”/>
 </class>
</hibernate-Mapping>
```

Using Hibernate Framework requires only the existence of a mapping file that establishes the correspondence between the structure of the XML input and that of the table for receiving the data. Before loading data into the transient table, the Loader deletes in the table all remaining data from a previous loading. An SQL command is built for this purpose in the program.

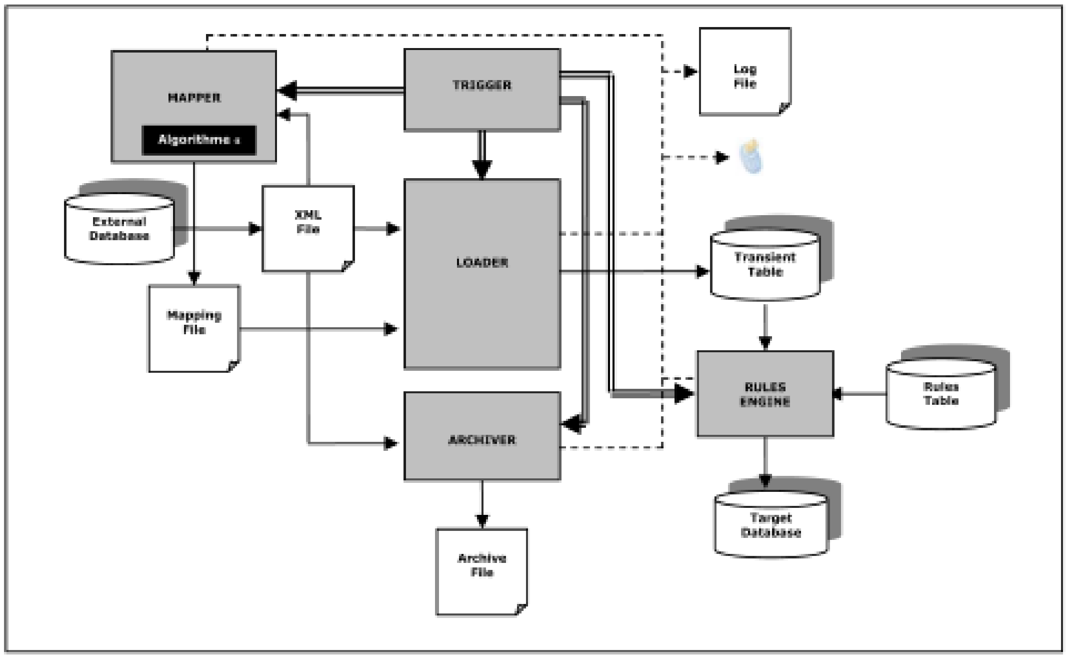


Figure 1. Overall Architecture

3.4 The Archiver

The function of the Archiver is to archive a XML file already processed by the Loader in a directory provided for this purpose. The name of the archived file is time stamped.

3.5 The Rules Engine

The Rules Engine is the component that, based on well-defined rules, inserts the data contained in the transient table into the target database. These rules, which are nothing but SQL commands, have themselves previously been entered into a table. The SQL syntax of these rules is that of the SQL corresponding to the database used: Oracle, MySQL, Sybase, DB2, etc.

Like the Loader, the Rules Engine is a program, which embeds SQL commands. These are used to read the rules and their execution. They address the table containing the rules or the target database. For reasons of portability, they must meet the ANSI SQL standard SQL-92, supported by all database providers.

In order to load an XML file, the loader is executed once. The Rules engine can be run by cons several times; it depends on the data model of the target database and the formulating of the SQL commands. The system administrator can decide, for technical reasons, to merge several rules into one single rule. He can also decide to break down a rule into several rules for the sake of a better reading of the rules and to ease the system maintenance.

Rule, i.e. the text of the SQL command is therefore stored in the table of rules. The structure of this table is as follows:

Column Name	Type	Length	Possible Values	Default Value
rule_code	CHARACTER	5		
rule_text	TEXT	LONGTEXT/ CLOB		
alternative_rule	TEXT	LONGTEXT/ CLOB		
if_rulefailure	CHARACTER	1	”R” for Rollback ”C” for r Commit ”A” for Apply alternative rule	”R”
if_alternativefailure	CHARACTER	1	”R” for Rollback ”C” for Commit	”R”
error_level	CHARACTER	1	”I” for Information ”W” for Warning ”E” for Error ”S” for Severe	”S”
comments	CHARACTER	255		
active_status	CHARACTER	1	”Y” for Yes ”N” for No	”N”

Table 1. `rule_code` is here an ID (Primary key)

A Rule can be any DML (Data Manipulation Language) command. The system administrator is responsible for its formulation. He has to ensure his logic, that is to say that the information is written in the right place in the target database and as required. As with any database relationships, the primary keys, the foreign keys and other integrity constraints (check constraints) are planned in the target database wherever required. Where a record does not meet the existing constraints, the Rules Engine will intercept the error message issued by the database, save it in the log file and eventually forward it to the system administrator.

The system administrator must also ensure the order in which rules are executed. In the case where an attempt to insert a “son” record takes place before the registration of the “parent”, the insertion fails. It may be recommended, for example, to use a change command (REPLACE) instead of an insertion command (INSERT). A “REPLACE” allows users to overwrite a record already existing in the database with a new record having the same primary key. An attempt to insert the record with a “INSERT” command would lead to an error.

A rule may be disabled. Its execution will not occur, although its code is transmitted to the Rules Engine. In this case an information message is written in the log file. The system administrator has also the ability to specify a “Commit”, a “Rollback” or to execute an alternative rule in case the execution of the rule fails. A “Commit” or “Rollback” will then be executed once again if the alternative rule could not be applied. An alternative rule could consist for example in writing rejected records in a table. The records of this table will allow the system administrator to investigate and decide what action to take for each record.

The Rules Engine is involved in the processing of primary keys. As for a Data Warehouse “processing of primary keys is an important task. During the transfer of data from multiple data sources to a data warehouse, primary keys can be ignored since they must be unique. As part of transformations, substitutes of source keys are created artificially.” [6]. The creation of substitutes is entrusted by the Rules Engine. A part can be stored in many stores; it will be appropriate to combine the code for the part with the code for the store and build in this way a unique code for the part.

Adapting the format of date data types while transferring data from the transient tables to the target database, is often inevitable. The extracted data from a character column of a transient table must be transformed into a date data type before it can be stored in a column of the same type in a table of the target database.

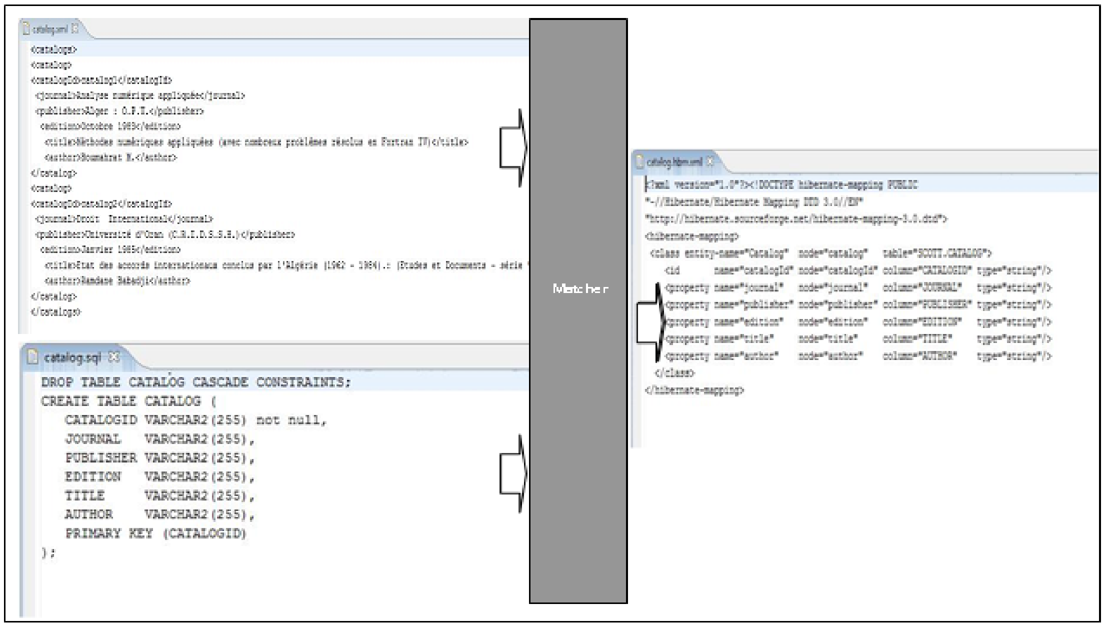
Numbers are also loaded from the XML file into transient table columns of type characters. These numbers may contain a point or comma as decimal separator. A proper transformation of the string for his storage in a column of decimal type has to take place in the rule. Other transformations can take place: currency conversion, conversion of measurements, scaling and aggregation. Any type of SQL command is permitted to do this, provided that the SQL command meets to the syntax allowed by the RDBMS (Relational Database Management System) in place and to the semantic data model. The business logic also has to be observed. Unlike syntax and semantic errors, no errors are reported in this case.

It should be noted that under a change in the structure of source data only the structure of the transient tables or the rules are to be amended but in no event should programs be updated.

4. “Matching” and “mapping” definition

The most common definition found in the literature of these two concepts is as follows: “Matching” is a process of comparison between two schemas and the result is “mapping”, that is to say a correspondence made between the nodes of the first schema and those of the second. This result is entered in a file that will be called “Mapping File”. Thus the definition given by Madhavan, Bernstein and Rahm [1] is formulated this way: “A schema consists of a set of related elements, such as tables, columns, classes, or XML elements or attributes. The result of a Match operation is a mapping. A mapping consists of a set of mapping elements, each of which indicates that certain elements of schema S1 are related to certain elements of schema S2.“

In our case it is a matching of XML to a relational database, that is to say making a correspondence between the elements of an XML file and the columns of a table which is the transient table, in order to load the data contained in the XML file into the transient table. As indicated in Fig. 2, the input information are the schema S1 which is the schema of data contained in the XML file and the schema S2 which is the structure of the transient table. The Matcher generates on the basis of this information the mapping file needed to load the XML data into the transient table.



```
<catalogs>
<catalog>
<catalogId>catalog1</catalogId>
 <journal>Analyse numérique appliquée</journal>
 <publisher>Alger : O.P.U.</publisher>
  <edition>Octobre 1983</edition>
   <title>Méthodes numériques appliquées (avec nombreux problèmes résolus en Fortran IV)</title>
   <author>Boumahrat M.</author>
</catalog>
<catalog>
<catalogId>catalog2</catalogId>
 <journal>Droit  International</journal>
 <publisher>Université d'Oran (C.R.I.D.S.S.H.)</publisher>
  <edition>Janvier 1985</edition>
   <title>Etat des accords internationaux conclus par l'Algérie (1962 - 1984).: (Etudes et Documents - série
   <author>Ramdane Babadji</author>
</catalog>
</catalogs>
```

```
DROP TABLE CATALOG CASCADE CONSTRAINTS;
CREATE TABLE CATALOG (
   CATALOGID VARCHAR2(255) not null,
   JOURNAL   VARCHAR2(255),
   PUBLISHER VARCHAR2(255),
   EDITION   VARCHAR2(255),
   TITLE     VARCHAR2(255),
   AUTHOR    VARCHAR2(255),
   PRIMARY KEY (CATALOGID)
);
```

```
<?xml version="1.0"?><!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
 <class entity-name="Catalog"  node="catalog"   table="SCOTT.CATALOG">
   <id        name="catalogId" node="catalogId" column="CATALOGID" type="string"/>
   <property name="journal"   node="journal"   column="JOURNAL"   type="string"/>
   <property name="publisher" node="publisher" column="PUBLISHER" type="string"/>
   <property name="edition"   node="edition"   column="EDITION"   type="string"/>
   <property name="title"     node="title"     column="TITLE"     type="string"/>
   <property name="author"    node="author"    column="AUTHOR"    type="string"/>
  </class>
</hibernate-mapping>
```

Figure 2. Input schemas and file mapping

4.1 Obstacles encountered during a manual matching

Manual matching can be very tedious, time consuming and may lead to errors because of various problems that may arise in cases such as the following:

• Dimension of schemas: The database tables can be numerous and contain a large number of attributes (in our case the number of transient tables and the number of columns that each can have). To establish a correspondence between the XML files nodes and the attributes of the database tables, it is necessary to know perfectly the meaning of all nodes and all attributes.

• Complexity of schemas: In all schemas part of the semantics is hidden in the structure. In a relational database, the primary or secondary keys can provide guidance on the semantics of the schema. This is even more accentuated in an XML file, given that semantic elements are available in the structure self of the file (tags).

• Linguistic problems: When attempting to establish correspondences between schemas, users may face further linguistic problems:

- Some items, although related, do are not recognized because of the use of synonyms. Some homonyms can be misleading. A correspondence can be established between elements bearing the same name but with different contents.

- A designation in a language other than local language may be used in the XML file

- The abbreviations make the matching even more difficult

• Cardinality problems: Cardinality is also an important point of matching. By cardinality we mean the ratio, how each element of the first schema links to the elements of the second schema.

• Redundancy: Redundancy information may appear when dealing with multiple data sources. In case of conflict, we have to decide which data source to adopt.

• Lack of information: A source of data does not always contain all the information required in the target schema. In this case either default values or the value null can be used for the missing data, or the target data can be deducted from calculations based on data sources. Some of the points above can be resolved after the matching process by the Rules Engine. It is possible to resolve, through a rule, the problems of redundancy or lack of information.

4.2 Automatic Matching

Because of the difficulties described in 4.1, it is appropriate to use automatic matching, or at least semiautomatic. The existing prototypes proceed with an analysis of XML files on the base of:

• The schema (meta data): All information gathered, which are not the result value of a concrete instance of the schema

• The instance (data): Data contained in a concrete instance of the schema can provide information on the meaning of elements of a schema [8].

They rely more on external sources such as:

• The thesaurus for finding synonyms, homonyms and resolution of ambiguities

• The language dictionaries, especially in international projects or in multinational companies, for the recognition of specific or local words, in linguistic analysis

• The mapping decisions taken in the past. Referring to decisions taken in the past speeds up the matching process. One can even imagine that this is the only way to make correct mapping decisions. For example: The value “Procurement” was often affected in the past to the element “Department”; if a schema S1 element has this value and an element of schema S2 has the name “Department”, it could be a correspondence between the two elements in case there is no other correspondence found.

A classification of existing approaches is given by Rahm and Bernstein in their article “A survey of Approaches to automatic schema matching” [7] according to which a match can be individual or the result of the combination of several algorithms. In the latter case it can be:

• Hybrid: The hybrid matching algorithms contain matching criteria that lead to joint matching decision

• Composite: matching algorithms are applied independently of each other; based on a weighting process the most likely element is then given as a result of the matching.

4.3 The Directory of Mapping Decisions Taken Previously (DMDTP)

For the system we present here we will use three linguistic resources: i) a thesaurus for the detection of synonyms and, following the language chosen by the user ii) a language dictionary and iii) a dictionary of abbreviations in this language. The eventual result of this search will be submitted for validation by the user. The user should, in case there is no result, enter the matching himself. This could be done by using a GUI (Graphical User Interface) by “drag & drop”. After the input or validation is complete, the corresponding mapping is stored in a “Directory of Mapping Decisions Taken Previously (DMDTP). From that moment the Mapping of these elements becomes automatic. The DMDTP is expected to take over time more and more volume and matching process to automate more and more cases. As defined in 4.2 this is an algorithm operating on the schema level, which does not require a weighting process, and using auxiliary information, including mapping decisions taken previously.

5. Conclusion

The system whose architecture is proposed here can be considered as an extension of the Hibernate framework, used here to load XML data into a transient table. The data in the transient table are “dispatched” to various tables of the target database according to the rules made by the user. The mapping between the input XML elements and those of the transient table is done on the basis of decisions taken by the user over time. This is a typical process which reaches stability after some time. It will automate more and more until user intervention is no longer required.

A graphical user interface (GUI), which will be called “Designer” offers more comfort in the input of rules, programming of trigger and creation of transient tables. This interface will also enable the user to enter by “drag & drop” or validate the correspondences between elements of the input XML file and columns of the transient table.

References

[1] Madhavan, Jayant ., Philip, Bernstein., A., Rahm, Erhard (2001). Generic Schema Matching with Cupid, *In*: Proc 27th VLDB Conference CD Rom.

[2] Milo,T., Zohar, S.(1998). Using schema matching to simplify heterogeneous data translation”, *In*: Proc 24th International Conference On Very Large Data Bases, p. 122–133.

[3] Palopoli,L., Sacca, D., Ursino, D. (1998). Semiautomatic, semantic discovery of properties from database schemas”. *In*: Proc Int. Database Engineering and Applications Symp. (IDEAS),IEEE Comput, p. 244–253.

[4] Chris, Date ., Darwen,J., Hugh (1996). A Guide to SQL Standard”, A (4th Edition), Addison-Wesley Professional, 4. Edition (November 18).

[5] Hibernate: http://www.hibernate.org/

[6] Andreas Bauer, Holger Günzel (2004). Data-Warehouse- Systeme”, Dpunkt Verlag, August.

[7] Rahm, Erhard ., Philip, A., Bernstein . A survey of approaches to automatic schema matching, *VLDB Journal* 10. 334–350.

[8] Wang,Q., Yu,J.,Wong,K. (2006). Approximate graph schema extraction for semi-structured data, *In*: Proc Extending DataBase Technologies, Lecture Notes in Computer Science, V. 1777. Springer, Berlin Heidelberg NewYork, p. 302–316.