Applying Principle of Locality to a Knowledge Base of Schema Mapping

BOUBAKER KAHLOULA[1]
KARIM BOUAMRANE[2]

Université d'Oran
Département Informatique
IGMO 31000 - Oran - Algeria
[1]bkahloula@gmail.com
[2]bouamrane.karim@univ-oran.dz

Abstract. Large amounts of information are posted on the web every day by thousands of enterprises, organizations and individuals. There is a daily exchange of data between companies, who also proceed to an extraction of data from web databases and documents in order to integrate them into databases. Since XML has become as the de facto standard in this area, the data contained in XML files are loaded into the database, usually relational, to be processed or analysed. In order to enable different systems to communicate with each other, it is necessary to match the data and create a mapping between the source schema and the target schema. Yet, in many companies, this mapping is manually built. Loading XML data, that were previously submitted to a manually constructed mapping, in databases, is subject to a prior itself automated schema matching. We propose in this paper a solution for loading (semi-)automatically XML data into relational databases. The solution we propose is to use previous mappings. These mappings are stored in what we call a Mapping Knowledge Base. We study the evolution of the volume of the Mapping Knowledge Base and we propose a strategy to use.

Keywords: Databases, Data Integration, Semi-structured Data, Schema Matching, Similarity Coefficients.

(Received October 29th, 2014 / Accepted September 7th, 2015)

1 Introduction

Because systems that exchange data have been developed in different environments, from different people and in different times and places, the structure of the data sources is often heterogeneous. To enable different systems to communicate with each other, it is necessary to match the data and create a mapping between the source schema and the target schema. Yet, in many companies, this mapping is manually built. Loading XML data, that were previously submitted to a manually constructed mapping, in databases, is subject to a prior itself automated schema matching.

The concept of "mapping" is closely related to the one of "matching" [26, 31]. A definition of these two concepts is given in [7]: "To cope with the heterogeneity and achieve interoperability, a fundamental requirement is the ability to match and map data across different formats. These two tasks are found in the literature under the names matching and mapping, respectively. A match is an association between individual structures in different data sources. Matches are the required components for every mapping task. The mappings are the products of the latter. A mapping, in particular, is an expression that describes how the data of some specific format is related to data of another. The relationship forms the basis for translating the data in the first format into data in the second".

We begin this article by introducing the state of the art in the area of schema matching and mapping. Their fields of application, the different approaches taken to date, as well as the tools and prototypes developed around this technique, will be presented in the first part.

The second part is about the solution that we propose for loading (semi-)automatically XML data into a relational database. The solution we propose is to use mappings established during previous handling. To load the data contained in XML files in the target database, we make a matching of XML elements. This allows us to achieve a mapping between the XML elements [6]. The mappings established during previous handling are stored in what we call a Mapping Knowledge Base (MKB). We study the evolution of the volume of the MKB and propose a strategy to use.

2 State of the art

The schema matching and mapping is applied in several areas: Data Integration [21], Data exchange [16], Messages exchange [8, 18], Schema Evolution [22] and Schema integration [4].

The cost of a manual matching can be very high, because it may require great efforts and time. Manual matching can also be a source of errors. The problems that appear in a manual schema matching have various causes. Many of these problems, that require a manual handling from the user, can be resolved if the matching process is automated. There are different approaches of the automatic schema matching. A taxonomy of these approaches was already developed by Rahm and Bernstein in 2001 [31]. The scale and complexity of the work generated by the schema matching, soon suggested the idea of developing a tool in order to assist the user in this task. Several prototypes, then called "mapping tools", have been developed so far: S-Match [17], Cupid [23], Clio [16], Coma++ [15, 3], etc. Some of these prototypes are generic; others are specific to a given branch (e.g. biomedical). An example of prototype evolved to a marketed product is Altova MapForce [2]. Some of these tools or prototypes make a Graphical User Interface (GUI) available to the user [3, 2], allow him to use an advanced mapping language [9], and offer the choice between advanced algorithms [29, 1]. Some of these tools and prototypes are specific to a definite type matching (e.g. XML-XML, XML-Relational Ontology-Ontology). Major software vendors are also engaged in the development of mapping tools: This is the case of IBM, with IBM InfoSphere Data Architect [19], Microsoft with Microsoft BizTalk Mapper [25], integrated into Microsoft Visual Studio, and BEA, with BEA AquaLogic [11].

Concerning us, we make a matching between XPaths. An example of XPath is: `/Orders/Customers/Address/City`. The matcher we use is a hybrid matcher [31], since XPath contains two pieces of information:

1. The node name
2. The path from the root to the node. The path to the node is an important feature of the node.

3 Proposed Architecture

3.1 Mapping Knowledge Base

What we call a Mapping Knowledge Base (MKB) is a table containing the history of the mappings previously inputted or validated by the user. It combines XPaths with column names of a table: XPath $\rightarrow$ column_name. The minimal structure of the MKB implies the existence of two columns: a first column containing XPaths, and a second column containing column names from a target table. Before loading the data contained in the XML into the table, we begin by searching in the MKB identical or similar XPaths to the XPaths extracted from the XML file. If we consider the XPaths as strings, there are several methods to estimate the similarity. The most popular indicators for the similarity, called indices, are: Jaccard, Cosine, Jaro-Winkler, and Dice-Sørensen. The column associated with each XPath in the MKB is given then to the user if the search is successful. The user can then validate the mapping. The mapping is stored in turn in the MKB. An unsuccessful search for an identical or similar XPath in the MKB requires the user to manually associate a column name to the XPath extracted from the XML file. The manually set mapping is also added to the MKB.

When using the MKB, a transitive operation is performed. Suppose that S_1 is the set of XPaths contained in the XML file and the MKB contains a mapping $S_2 \rightarrow S_3$, S_2 is a set of XPaths and S_3 is a set of column names. A matching between S_1 and S_2 occurs first and then the mapping $S_2 \rightarrow S_3$ is used. The result is a mapping $S_1 \rightarrow S_3$. This mapping is added in turn to the MKB after validation by the user (Figure 1). The volume of the MKB will progressively increase

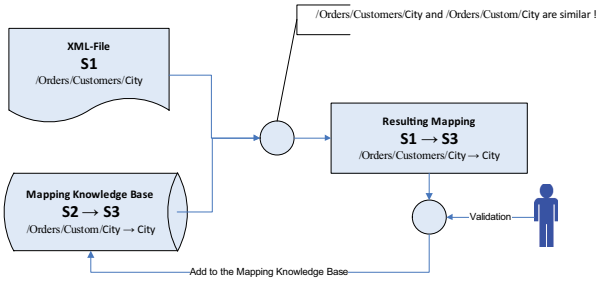

Figure 1: Using a Mapping Knowledge Base

as XML files are loaded into the target database. The number of XPaths increases, improving by this the probability of finding in it identical or similar XPaths as those extracted from an XML file. In order to illustrate this, let's consider an example where we have to load XML files, from which is extracted a constant number of XPaths equal 100. Let's suppose further that the percentage of identical XPaths found in the MKB is also constant and equal to 1. The result of a simulation based on these hypotheses is given in Figure 2. We note firstly that the number of XPaths contained in the MKB stabilizes after a period of filling, while the number of the manually set mappings decreases linearly. Since we use the XPaths contained in the MKB as historical information, the greater the volume of the MKB, the smaller the volume of the work done manually. We assumed that the percentage of XPaths

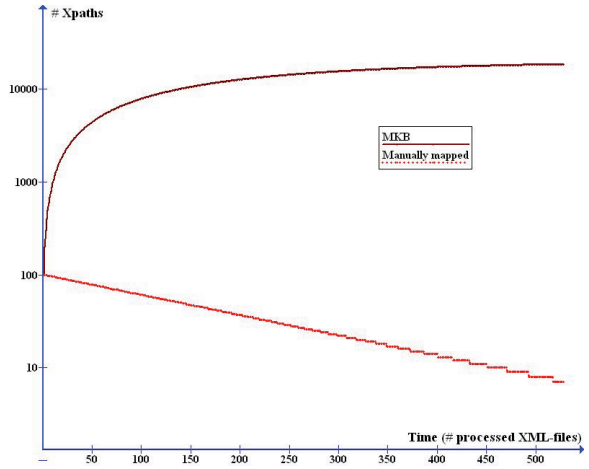

Figure 2: Evolution of the Mapping Knowledge Base

found in MKB is 1. XML Files can be very homogeneous if they belong to the same area but they are less homogeneous if they belong to various domains. The "filling period" of the MKB varies depending on the degree of homogeneity, but the volume of the MKB stabilizes always. The variation of the homogeneity is equivalent to the variation of the Precision. Precision (or positive predictive value) is a criterion used in the Information Retrieval (IR) that measures performance. This will be discussed in more details in 3.2.

3.2 Similarity between two strings

The Matcher contained in the prototype we developed [20] uses what is called in the literature "Name Matching" [31, 10]. The Name Matching is based on a comparison between the element's names. We estimate the similarity between each of the XPaths extracted from the XML file with each of the XPaths contained in the MKB. We determine by this comparison the value of the similarity index of the XPaths considered as strings. There are several indicators to estimate the similarity between two strings: Jaccard, Cosine, Jaro-Winkler, Dice-Sørensen, usw. The value of a similarity index is in general between 0 and 1, the value of 1 indicates that the strings are exactly the same. If we consider, for example, the two strings C_1=`"orderitem"` and C_2=`"order"`, the union of these two chains `{d,e,i,m,o,r,t}` has the length 7 and their intersection `{d,e,o,r}` has the length 4. The Jaccard index is calculated as following: S_J= |A∩B|/|A∪B| The Jaccard index for C_1 and C_2 is therefore: $S_J=\frac{4}{7}$=0,57

The efficiency and effectiveness of Jaro-Winkler index make it an indicator used in number of prototypes. Although we have considered in the prototype that we developed within the context of our research all the indicators mentioned above, it is the Jaro-Winkler index, which we use as default in matching operations. It gave, in terms of efficiency, the best results in the tests we made (Figure 5). Both Cohen [12] ("good distance metric is a fast heuristic scheme, proposed by Jaro and later extended by Winkler") and Da Silva [13] ("we performed experiments to assess the quality of eight similarity functions according to the discernibility function. The results show that, for the data set considered, the best function was Jaro-Winkler") confirm that Jaro-Winkler similarity is a good metric.

3.3 Sequential search

After calculating the similarity index, the resulting mapping from the matching that produced the highest value of the similarity index is proposed to the user for validation. The search in the MKB for identical or similar XPaths as those extracted from XML files is done

according to the pseudo-code below. As we can see the search in the MKB takes place sequentially.

```
// FP[i] are the XPaths extracted from the
// XML-file
// MP[j] are the XPaths contained in the MKB
// MCol[j] are the columns of the transient
// table mapped to the MP[j]
// We search for the FCol[i], the columns of the
// transient table to map to FP[i]
//
// Loop on the XPaths extracted from the
// XML-file
for(i=0; i<n; i++) {
    FCol[i] = null;
    maxSimCoef = 0.00;
    // Loop on the XPaths contained in the MKB
    for(j=0; j<m; j++) {
       simCoef[i]= calcSim(FP[i],MP[j]);
       if (simCoef[i] > maxSimCoef) {
          maxSimCoef = simCoef[i];
          FCol[i] = MCol[j];
          if (simCoef[i]== 1.00) break;
       }
    }
}
```

The matching, that is to say, the determination of the value of the similarity index of each XPath extracted from the XML file with each XPath contained in the MKB, is interrupted if the similarity index is equal to 1. This means that the XPath extracted from the XML file is exactly identical to the one found in the MKB. If we consider the algorithm that we have just given, we see that the calculation of the similarity index takes place inside two nested loops. In the worst case, that is to say, in case it is not found XPaths in the MKB exactly equal (similarity index = 1) to the XPaths extracted from the XML file, and therefore the inner loop has not been interrupted, the calculation of the similarity index must be calculated *n*m* times if *n* is the number of XPaths contained in the XML file to load and *m* the number of XPaths contained in the MKB.

As we have already said, the volume of the MKB will increase along with the XML files loading into the target database. Although stabilizing over time, the number of XPaths contained in the MKB can be several thousands. Regardless of the used similarity index, the matching scheme can be time consuming. We can distinguish by observing Figure 2 that shows the evolution of the volume of the MKB, two periods:

- A first period P_R, we say that it is the "filling period"

- A second period P_S during which the MKB is "sufficiently filled". Its volume is stable.

P_R + P_S is the lifetime of the MKB.

3.4 Exact search

If it is allowed to go sequentially through the the MKB during P_R, because the MKB is not sufficiently filled and the probability ξ of finding an XPath inside it, that is exactly identical to the XPath extracted from the XML file, is small, this approach is certainly "oversized" for the P_S period. We can, in fact, during the second period make an exact search, since the probability ξ is high. The exact search is described in the following pseudo-code. The function `seek` here means an exact search through the use of a Database Index.

```
// Loop on the Xpaths extracted from the
// XML-file
for(i=0; i<n; i++) {
    FCol[i] = null;
    // Exact Search for FP[i] in the MKB
    seek FP[i];
    if found {
       // MP[j] = FP[i] found and its
       // corresponding MCol[j]
       FCol[i] = MCol[j];
    }
}
```

Our strategy is therefore, to go sequentially through the MKB during the filling period P_R, and then proceed to exact searches from the moment the MKB is sufficiently full.

3.5 Efficiency and effectiveness

The efficiency and the effectiveness of an algorithm or a strategy of schema matching are two important factors, particularly in the presence of large volumes. Effectiveness is the ability to correctly identify the mappings. Efficiency, for its part, is the ability to use the least possible resources. Indeed, the processing time may take several hours or even days [32, 30].

3.5.1 Efficiency

As in [14], in order to estimate the efficiency of our strategy, we take the measurement tools of the Information Retrieval (IR) area, in particular the so-called F-Measure, which is a combination of the two called Recall and Precision. In Figure 3:

- Part A, which is the "false negative", represents the XPaths of the XML file that are not included in the MKB.

- Part B, the "true positive", is that of the XPaths contained in the XML file existing in the MKB. The mapping is established in this case automatically.

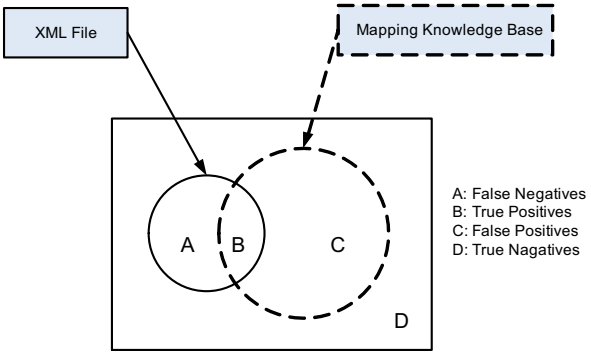

Figure 3: Efficiency measures

- Part C, the "false positive", is the set of the XPaths, although they exist in the MKB, they are not useful for the processing of the XML file, since they are not contained in it.

- Part D, the "true negatives", is of no interest, since it is the part of the XPaths that are included neither in the XML file to load nor in the MKB.

Precision reflects the degree of homogeneity of the XML files. It is the part of real mappings among all those found in the MKB. The set of all the found mappings is the set of all the XPaths contained in the MKB. Precision is calculated as following:

$$\text{Precision} = |B|/|B|+|C|$$

Recall, on the other hand, is the part of the real mappings among all those sought. The set of all mappings being sought is the set of all the XPaths contained in the XML file. Recall is calculated as following:

$$\text{Recall} = |B|/|A|+|B|$$

Both Precision and Recall, will give only a relative idea of the quality [14]. A third measure, which is the harmonic mean of the two, is used: F-measure. The latter is given by the following formula:

$$\text{F-Measure} = 2*(\text{Precision}*\text{Recall})/(\text{Precision}+\text{Recall})$$

Although we have no information about the volume of the XML files that we have to load into the target database, i.e. about the number of XPaths they contain, and we cannot influence this volume, we have in our simulation (Figure 2) considered the hypothesis that the volume is constant (equal to 100 XPaths). We have further assumed that the percentage of the XPaths found in the MKB is equal to 1. These assumptions are reflected in Figure 4 by a constant Precision. But we see in the same figure that Recall and F-Measure "accompany" the evolution of the volume of the MKB. Both measures increase up to a certain value and then stabilize. This means, concerning these two measures, that they reach their maximum in the point where the volume of the MKB has stabilized. It is precisely this point that we determine in 3.4, that is the point at which the user can switch to an exact search.

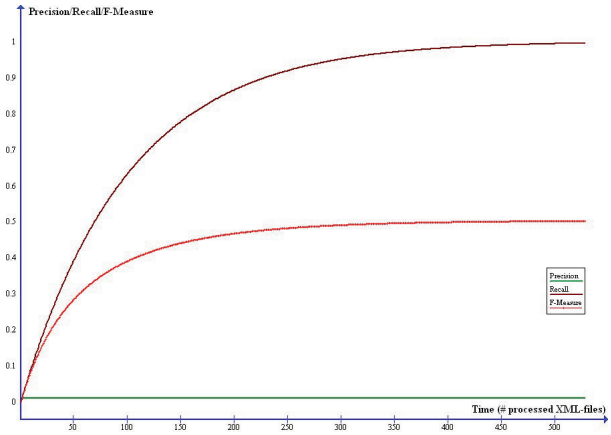

Figure 4: Precision, Recall and F-Measure

3.5.2 Effectiveness

To test the effectiveness of our concept, we generated XML files with increasing volumes. The created XML files belong to the same domain, which is the e-Commerce domain (Orders, Shipping, etc.). The testing took place with XML files, whose XPaths number varies from 250 to 450. The volume of the files which contain instances also varies from 0.3 to 12MB. The MKB contains 450 XPaths. The tests were done on a Windows XP machine with 1.98 GB of RAM and 2x86 processors 3GHz and a repository created in a database Oracle Database 10g Enterprise Edition, that was installed on the same machine. Looking at the graph of the test results (Figure 5), we note that the exact search is the most effective. This result is expected since the access to the MKB is not sequential in this case but occurs through an index. The evaluation of the similarity index between each of the XPaths contained in the XML file and each of those contained in the MKB does not occur in the case of an exact search. The creation of the MKB as a table of the repository, containing three columns `mkb_path`, `table_name` and `column_name`, and thus representing a mapping `mkb_path` → `(table_name, column_name)`, is followed by the creation of an in-

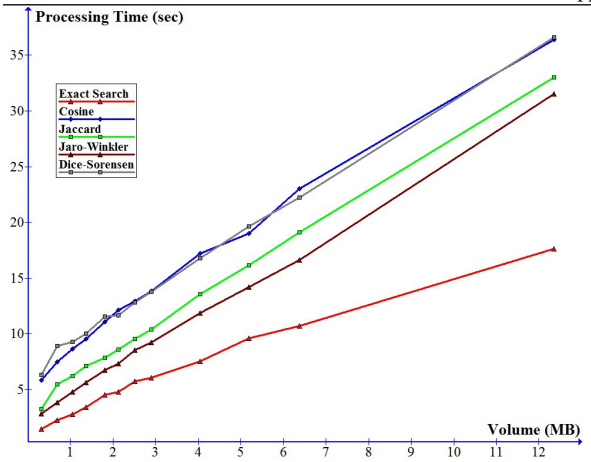

Figure 5: Efficacity test results

dex on the `table_name` and `mkb_path` columns. Access as in the following example:

```
SELECT column_name
FROM mkb
WHERE table_name = "orders" AND mkb_path =
"/Orders/Customers/Address/City";
```

necessarily use the index on the `table_name` and `mkb_path` columns. The primary role of an index is to accelerate access by reducing disk I/O [28]; this is valid for any DBMS. The good performance of the exact search is justified by the use of an index. We also note that the difference between the exact search and the similarity index calculation, regardless of the method of calculation, widens increasingly with time. Therefore, the more the volume of the MKB is big, the more the exact search that uses an index is effective.

3.5.3 Applying the principle of locality

We saw earlier that the period of filling the MKB varies according to the degree of homogeneity of XML files to be loaded into the target database. This period may take some time if the files belong to several completely different areas, but the efficiency and effectiveness can also be improved during this period. Both operating systems [24] as DBMS [27] use strategies to manage caches or buffers. These strategies are generally based on the principle of spatial or temporal locality. They use in order to optimize the time of response, instructions or data located in the memory areas near to those occupied by recently accessed instructions or data (spatial locality) or reuse data or instructions used in the recent past (temporal locality).

We use in order to optimize the access time to the MKB during the filling period, that is to say before we switch to an exact search, strategies that are similar to LRU and LFU caching strategies [24]. Least Recently Used (LRU), the first of these two strategies, assumes that the most recently used data and instructions are those likely to be used again in the near future. The second strategy, Least Frequently Used (LFU), prioritizes the most often referenced data and instructions, considering that they are likely to be used again in the future.

The application of one or the other of these strategies implies an extension of the table structure of the MKB. LRU requires adding a `last_used` column that contains a time-stamp indicating the date at which the last access to each XPaths took place.

LFU requires an additional `number_uses` column containing the total number of access of each XPaths. The `last_used` and `number_uses` columns are updated at each new reading from the MKB or at the moment of inserting in the MKB. The application of LRU and/or LFU also requires a descending sort of the MKB according to the `last_used` and `number_uses` columns. The access to the MKB during its sequential filling to find a "recently added" or "often used" identical XPath to an XPath extracted from the XML file is faster if the MKB is sorted according to the one or the two columns:

- `last_used` for LRU
- `number_uses` for LFU
- `last_used, number_uses` for LRU and LFU
- `number_uses, last_used` for LFU and LRU

This results in the creation of indexes. For instance, concerning the Oracle DBMS, a so called hint tells the DBMS that an index must be used even if it is a sequential access.

Although index maintenance presents CPU and I/O resource demand in write-operations, LRU/LFU can speed up the finding/matching process, precisely because read-operations are over time more and more while write-operations are less (Figure 2).

3.5.4 Switch to an exact search

Figure 6 shows the MKB volume's evolution (number of XPaths contained in the MKB) according to the number of files processed. The number of XPaths contained

in the MKB is considered by the user as constant if it didn't change for a while more than a value δ (since a number of files have been processed). We seek, according to δ, the point of the x-axis μ, from which the user can opt for an exact search. This is given by:

$$\mu = \min(x_i), (\max(y_i) - y_i) \leq \delta$$

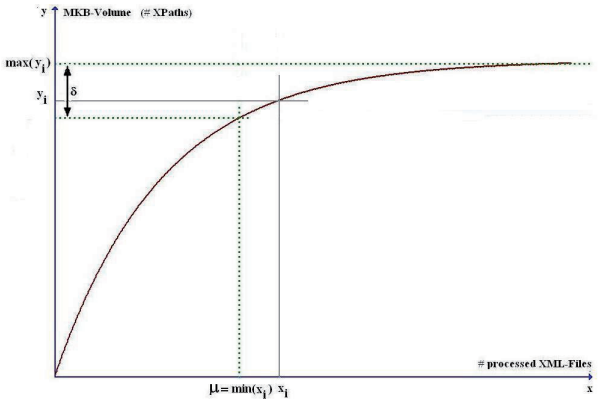

Figure 6: Switch to an exact search

This means that after μ processed files, the user can opt for an exact search and switch to automatic loading of XML files in the database. The developed prototype within the context of our research has a module of statistics that enables to determine the switching point according to a δ value entered by the user. The produced statistics allow defining from when the volume of MKB can be considered stable.

3.5.5 From (semi-)automatic to fully automated processing

As we can see in Figure 7, the processing of an XML file with the prototype we developed starts after extracting data from web, databases or documents and storing them in an XML file. It takes place according to the following steps:
(1) The first module, called Matcher, starts by reading the XML file.

(2a) it performs the extraction of the XPaths from the XML file and stores them in a table which we call Mapping Table. An XPath is the way to go along the tree to reach each of the instances contained in the file. This should not be confused with the language XPath. Example: `/Article/JournalIssue/pubDate/Year`.
The Mapping Table contains two columns. It represents the relationship between the XPaths contained in the XML file and the column names of a transient table. Example: `/Article/JournalIssue/pubDate/Year` → `Year`.
Transient tables are called that way because they are used as a "transit" for the records that shall be loaded into the target relational database. Their structures are static and defined by the user, who is in this case a database administrator or as named in the literature related to the field of data warehousing [5], the ETL- or ELT-team. Transient tables are called in this same literature by "staging area".

(2b) the second column is filled if the Matcher has found in the MKB a similar or an identical XPath. The MKB is a table that has a similar structure of the Mapping Table. The MKB contains the entire history of the mappings inputted or validated by the user in the previous processing. An identical XPath as in the example given above would be `/Article/JournalIssue/pubDate/Year` while a similar XPath could be, for example, `/Article/Journal/Date/Year`.

(2c) as well the instances as the XPaths contained in the XML file are stored for technical reasons in a temporary table. This allows indexing and sorting operations. Unlike the transient tables, the temporary table is a table in the repository. It is an integral part of the prototype.

(3a) the mapping, that is to say the correspondence between the XPaths and column names of the transition table is inputted or validated by the user. A graphical interface is available for this purpose.

(3b) the mapping, inputted or validated by the user, is stored in the MKB.

(4a) a second module, called Loader, reads the data contained in the temporary table.

(4b) it loads them in the transient tables.

(5a) a third module, named Rules Engine, accesses Rules. The Rules, which are SQL commands, are stored as text in a table of the repository.

(5b) the Rules Engine reads the records contained in the transient table.

(5c) based on the Rules, the Rules Engine dispatches

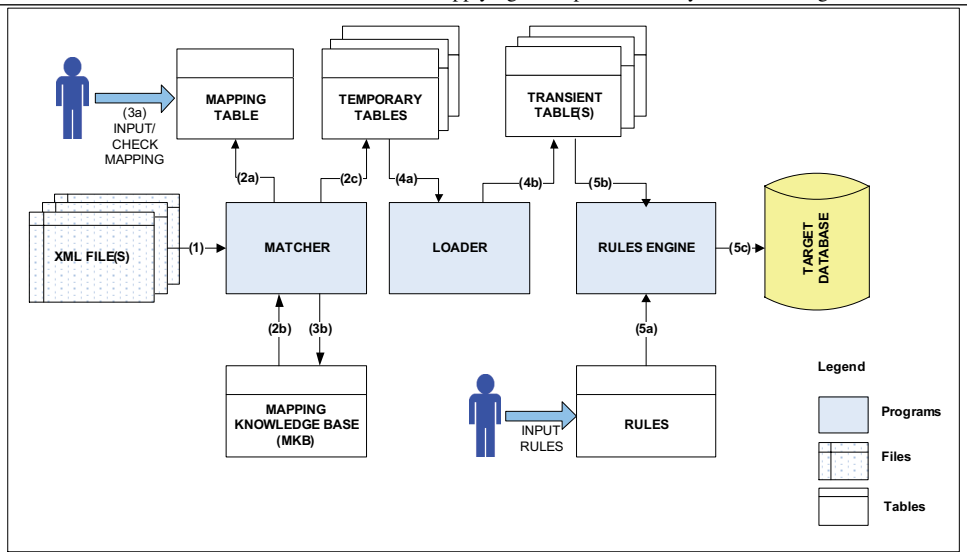

Figure 7: Process sequences

the records contained in the transient tables over the different tables of the target relational database. The prototype is configurable (Figure 8). Amongst others the following parameters can be specified:

- The similarity index to be used: Jaccard, Cosine, Jaro-Winkler, Dice-Sørensen or exact search
- The threshold value for the similarity index, produced by the automatic schema matching
- The LFU/LRU or LRU/LFU strategy that may be applied to access to the MKB
- The potential thesaurus to use to look for synonyms

Among these parameters, the threshold value of the similarity index is these involved in the automation of the entire process of loading XML files into the target database. Indeed, an XML file can be loaded fully automatically if the automatic matching, "inspired" from the mappings contained in the MKB, resulted in a mapping of all columns of the transition table without exception, and if the similarity index calculated for each column was always less than or equal to the threshold value. The user must complete and/or validate the mapping established automatically if the condition is not met.

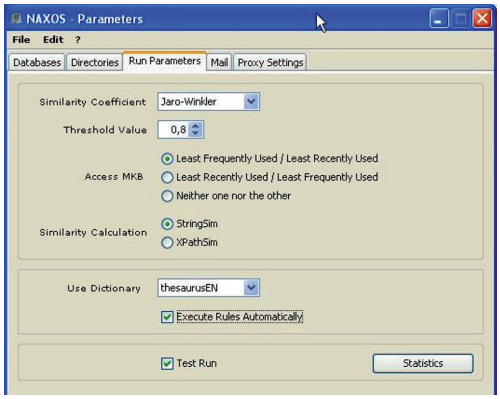

Figure 8: Customizing the prototype

4 Conclusion

The proposed architecture for a (semi-)automatic loading of XML data into a relational database is based on the use of mappings established during previous processing. The table, which we called Mapping Knowledge Base, containing these mappings becomes over time increasingly voluminous, which slows the processing. The effectiveness of the architecture increases with the volume of the MKB. Unfortunately efficiency declines.

We propose in order to accelerate the processing, the use of an exact search starting from the moment the MKB's volume stops rising. This volume is, however, reached only after a certain period, which varies according to the heterogeneity of the XML files to be loaded, that is to say depending on the number of the different fields to which belong the XML files. We recommend using to access the MKB during these period the

LRU/LFU strategies.

References

[1] Alexe, B., Chiticariu, L., Miller, R. J., and Tan, W.-C. Muse: Mapping Understanding and deSign by Example. In *IEEE 24th International Conference on Data Engineering*, pages 10–19, Washington, DC, USA, April 2008. IEEE Computer Society.

[2] Altova. Altova. http://www.altova.com. Online; accessed 29-October-2014.

[3] Aumüller, D., Do, H. H., Massmann, S., and Rahm, E. Schema and ontology matching with coma++. In *SIGMOD Conference*, pages 906–908, 2005.

[4] Batini, C., Lenzerini, M., and Navathe, S. B. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[5] Bauer, A. H. and Günzel, H. H. *Data-Warehouse-Systeme*. dpunkt, Heidelberg, 2009.

[6] Bellahsène, Z., Bonifati, A., and Rahm, E., editors. *Schema Matching and Mapping*. Springer, 2011.

[7] Bellahsène, Z. and Duchateau, F. Tuning for schema matching. In Bellahsène, Z., Bonifati, A., and Rahm, E., editors, *Schema Matching and Mapping*, pages 293–316. Springer, 2011.

[8] Bernstein, P. A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I. Data management for peer-to-peer computing : A vision. In *WebDB*, pages 89–94, 2002.

[9] Bernstein, P. A. and Melnik, S. Model management 2.0: manipulating richer mappings. In Chan, C. Y., Ooi, B. C., and Zhou, A., editors, *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Beijing, China, 2007. ACM.

[10] Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., and Fienberg, S. Adaptive name matching in information integration. *Intelligent Systems*, 18(5):16–23, 2003.

[11] Carey, M. J. Data delivery in a service-oriented world: the bea aqualogic data services platform. In Chaudhuri, S., Hristidis, V., and Polyzotis, N., editors, *SIGMOD Conference*, pages 695–705. ACM, 2006.

[12] Cohen, W. W., Ravikumar, P., and Fienberg, S. E. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration*, pages 73–78, August 2003.

[13] da Silva, R., Stasiu, R. K., Orengo, V. M., and Heuser, C. A. Measuring quality of similarity functions in approximate data matching. *J. Informetrics*, 1(1):35–46, 2007.

[14] Do, H. H., Melnik, S., and Rahm, E. Comparison of schema matching evaluations. In *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, pages 221–237. Springer-Verlag, 2003.

[15] Do, H. H. and Rahm, E. Coma - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[16] Fagin, R., Haas, L. M., Hernández, M. A., Miller, R. J., Popa, L., and Velegrakis, Y. Clio: Schema mapping creation and data exchange. In Borgida, A., Chaudhri, V. K., Giorgini, P., and Yu, E. S. K., editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 198–236. Springer, 2009.

[17] Giunchiglia, F., Shvaiko, P., and Yatskevich, M. S-match: an algorithm and an implementation of semantic matching. In Bussler, C., Davies, J., Fensel, D., and Studer, R., editors, *The Semantic Web: Research and Applications*, volume 3053 of *Lecture Notes in Computer Science*, pages 61–75. Springer, Berlin / Heidelberg, 2004.

[18] Halevy, A. Y., Ives, Z. G., Suciu, D., and Tatarinov, I. Schema mediation in peer data management systems. In Dayal, U., Ramamritham, K., and Vijayaraman, T. M., editors, *ICDE*, pages 505–516. IEEE Computer Society, 2003.

[19] IBM. IBM InfoSphere Data Architect. http://www-03.ibm.com/software/products/en/ibminfodataarch. Online; accessed 29-October-2014.

[20] Kahloula, B. and Bouamrane, K. Using a mapping knowledge base in a system for (semi-)automatic

loading of xml data into relational databases. *Proceedings World Congress on Computer and Information Technology (WCCIT), 2013*, pages 1–7, June 2013.

[21] Lenzerini, M. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGMOD-SIGART Symposium on Principles of Databaseystems (PODS)*, 2002.

[22] Lerner, B. S. A model for compound type changes encountered in schema evolution. *ACM Trans. Database Syst.*, 25(1):83–127, 2000.

[23] Madhavan, J., Bernstein, P. A., and Rahm, E. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[24] Maffeis, S. Cache management algorithms for flexible filesystems. *ACM SIGMETRICS Performance Evaluation Review*, 21:1–3, 1993.

[25] Microsoft. Microsoft Using BizTalk Mapper. http://msdn.microsoft.com/en-us/library/aa547076.aspx. Online; accessed 29-October-2014.

[26] Miller, R. J., Haas, L. M., and Hernández, M. A. Schema mapping as query discovery. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 77–88, 2000.

[27] Oracle. Oracle Introducing Oracle Database Cache. http://docs.oracle.com/cd/A97336_01/cache.102/a88706/ic_intro.htm#1001429. Online; accessed 29-October-2014.

[28] Oracle. Oracle Schema Objects. http://docs.oracle.com/cd/B10500_01/server.920/a96524/c11schem.htm. Online; accessed 29-October-2014.

[29] Popa, L., Velegrakis, Y., Miller, R. J., Hernández, M. A., and Fagin, R. Translating web data. In *VLDB*, pages 598–609. Morgan Kaufmann, 2002.

[30] Rahm, E. Towards Large-Scale Schema and Ontology Matching. In Bellahsene, Z., Bonifati, A., and Rahm, E., editors, *Schema Matching and Mapping*, Data-Centric Systems and Applications, chapter 1, pages 3–28. Springer, Berlin, Heidelberg, 2011.

[31] Rahm, E. and Bernstein, P. A. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[32] Shvaiko, P., Euzenat, J., Giunchiglia, F., Stuckenschmidt, H., Noy, N. F., and Rosenthal, A., editors. *Proceedings of the 4th International Workshop on Ontology Matching (OM-2009) collocated with the 8th International Semantic Web Conference (ISWC-2009) Chantilly, USA, October 25, 2009*, volume 551 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.