A Tool for Semi-Automated Semantic Schema Mapping: Design and Implementation

Dimitris Manakanatas, Dimitris Plexousakis

Institute of Computer Science, FO.R.T.H.
P.O. Box 1385, GR 71110, Heraklion, Greece
{manakan, dp}@ics.forth.gr

Abstract. Recently, schema mapping has found considerable interest in both research and practice. Determining matching components of database or XML schemas is needed in many applications, e.g. for e-business and data integration. In this paper a complete generic solution of the schema mapping problem is presented. A hybrid semantic schema mapping algorithm which semi-automatically finds mappings between two data representation schemas is introduced. The algorithm finds mappings based on the hierarchical organization of the elements of a term dictionary (WordNet) and on the reuse of already identified matchings. There is also a graphical user interface that allows the user to parameterize the algorithm in an easy and fast way. Special attention was paid to the collaboration of the algorithm with a matching management tool. This collaboration, as proved by the evaluation of the algorithm, resulted in the creation of a generic system for detecting and managing mappings between schemas of various types and sizes.

1 Introduction

In most schema integration systems schema matching is a fundamental problem in many applications, such as integration of web-oriented data, e-commerce, schema evolution and migration, application evolution, data warehousing, database design, web site creation and management, and component-based development. A matching process uses two schemas as input and produces a schema mapping between pairs of elements of the input schemas which are semantically related [2], [3], [4], [6], [9], [10], [11].

Most of schema matching is typically performed manually, possibly supported by a graphical user interface. Obviously, manually specifying schema matches is a time-consuming, error-prone, and therefore expensive process. Moreover, there is a linear relation between the level of effort and the number of matches to be performed, a growing problem due to the rapidly increasing number of web data sources and e-businesses to integrate. A faster and less labor-intensive integration approach is needed. This requires automated support for schema matching and this is the aim of this work. Furthermore, most approaches in the domain, due to their complexity and lack of an intuitive interface, appeal to experts making, thus, schema matching an expensive process. Taking also into consideration the call for more and more accurate

matching production, we consider the task of supporting the matching process of substantial importance.

The semantic algorithm which is introduced in this paper and the semi-automated matching tool which was implemented based on this algorithm created a generic schema matching tool for many kinds of schemas (relational, XML, OWL). This tool can be easily operated even by a non-expert user because of its practical and intuitive interface. Moreover, the new and improved techniques for matching reuse offered by the algorithm minimize the total time spent on manual matching increasing, at the same time, the amount and quality of the output results.

Another important component of this work is the WordNet Lexical Database [15] that helps finding matchings which could not be identified by previous approaches. The proposed tool can find matchings with different cardinalities (1:1, 1:n and n:1). Genericity and expandability are the major advantages of this work.

The paper is organized as follows. Section 2 presents previous work and the basic characteristics of known matchers, including also a taxonomy of them. Section 3 introduces our semantic algorithm and describes its operation. The evaluation results based on quality measures and comparison with other approaches are given in Section 4. Section 5 presents the main components of the system, such as the interface, the interaction between the algorithm and COMA++ [2] and the WordNet Lexical Database. Section 5 also includes an analysis of use in the area of Purchases-Orders. Future work is discussed in Section 6 and conclusions are drawn in Section 7.

2 Related Work

In this section we present a classification of the major approaches to schema matching [5] and describe the most popular ones. Fig. 1 shows part of this classification scheme together with some representative approaches.

2.1 Classification of Schema Matching Approaches

An implementation of a schema matching process may use multiple match algorithms or matchers. This allows the selection of the matchers depending on the application domain and schema types. Given that the use of multiple matchers may be required there are two sub problems. First, there is the design of individual matchers, each of which computes a mapping based on a single matching criterion. Second, there is the combination of individual matchers, either by using multiple matching criteria (e.g., name and type equality) within a hybrid matcher or by combining multiple match results produced by different match algorithms. For individual matchers, the following largely-orthogonal criteria are considered for classification:

- *Instance vs. schema:* Matching approaches can consider instance data (i.e., data content) or only schema-level information.

- *Element vs. structure matching:* Match can be performed for individual schema elements, such as attributes, or for combinations of elements, such as complex schema structures.
- *Language vs. constraint:* A matcher can use a linguistic approach (e.g., based on names and textual descriptions of schema elements) or a constraint-based approach (e.g., based on keys and relationships).
- *Matching cardinality***:** Each element of the resulting mapping may match one or more elements of one schema to one or more elements of the other, yielding four cases: 1:1, 1:n, n:1, n:m. In addition, there may be different match cardinalities at the instance level.
- *Auxiliary information*: Most matchers not only rely on the input schemas S1 and S2 but also on auxiliary information, such as dictionaries, global schemas, previous matching decisions, and user input.

Note that this classification does not distinguish between different types of schemas (relational, XML, ontologies, etc.) and their internal representation, because the algorithms depend mostly on the kind of information they exploit, not on its representation.

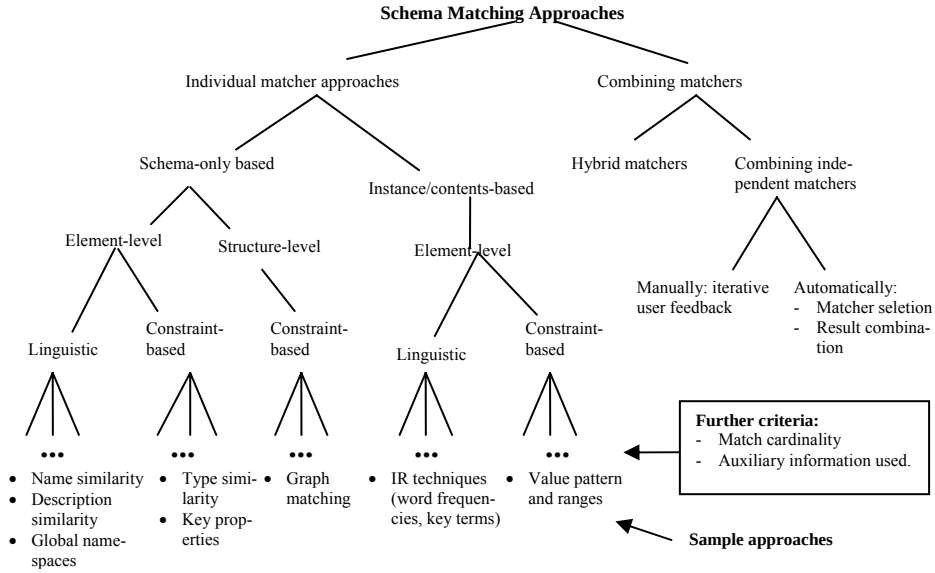

Fig. 1. Classification of schema matching approaches

2.2 Prevalent Approaches from the Literature

In this section we present the most important generic implementations in the domain whose characteristics are displayed in Table 1. It is observed that the determination of a similarity value in the interval [0,1] for every possible matching between two schemas is common in all applications.

	Clio [8]	*Cupid [10]*	*SF [11]*	*Protoplasm [3]*	*COMA++ [2,4]*
Schema types	XML, Relational	XML, Relational	Relational, RDF, XML	Relational, XML, ODMG	Relational, RDF, OWL, XSD, XDR
Metadata representation	Schema trees	Schema trees	Propagation graph	Directed graph (SMM graph)	Directed acyclic graph
Match granularity	Element/ Structure level: Entities/ relationships/ attributes	Structure level: Entities/ relationships/ attributes	Element/ structure level: Entities/ relationships/ attributes	Element/ structure level: Entities/ relationships/ attributes	Element/ structure level: Entities/ relationships/ attributes
Match cardinality	1:1 and n:1 and n:m	1:1 and n:1	1:1	1:1 and n:1	1:1 and n:1
Comments	No ontology manipulation	A very good structure algorithm	Hybrid structure algorithm	No ontology manipulation	The most complete tool

Table 1. Main Characteristics of proposed schema matching approaches

Clio consists of a set of Schema Readers, which read a schema and translate it to an internal representation, a Correspondence Engine (CE), which is used to identify matching parts of the schemas or databases, and a Mapping Generator, which generates view definitions to map data in the source schema into data in the target schema. The correspondence engine uses n:m element level matchings that have been gained by knowledge or that have been given by the user via a graphical interface.

On the contrary, Cupid is a hybrid schema matcher, combining a name matcher and a structure-based matcher. This tool finds the element matchings of a schema, using the similarity of their names and types at the leaf level.

SF uses no external dictionary, but offers several filters for the best matchings selection from the result of the structure-based matcher. After the end of the structural matching, the user can choose which matchings he wants to keep by using certain filters provided by the system.

Protoplasm offers a new architecture or schema matching, which includes two new internal representations, which are called Schema Matching Model Graph (SMM Graph), an interface which handles the mapping algorithms and, finally, a smart technique of combinational execution of them. It enables the addition of a new algorithm when the user wants to and it is able to handle relational databases, XML schemas and ODMG.

Finally, COMA++ offers a large library of different matchers and supports several ways of combining the results from different matchers. These matchers support finding structure-based matchings as well as element-level matchings.

3 A Hybrid Semantic Schema Mapping Algorithm

Our efforts are focused on the design and implementation of a Hybrid Semantic Schema Mapping Algorithm. The algorithm is presented in this section from three points of view: its input, operation and output. Broadly speaking, the structure of the algorithm can be seen in Fig. 2, but it will be also analytically described in the sequel.

```
Input: 2 Schemas, S1, S2, (XDR, XSD, OWL, RDF, Relational Databases)
                          in Directed Acyclic Graph format
 Output: Mapping Matrix with matchings between two schemas
     Operation:
(1): While (∃ next S1.node )
(2): {
(3):      get next S1.node
(4):      If (Each similarity value X of current S1.node with an
(5):      S2.node  ∉ [minimum threshold, maximum threshold]) then
(6):      {
(7):           While (∃ next S2.node )
(8):           {
(9):                get next S2.node
(10):               tokenize S1.node based on delimiters ' ', '.', '_', '-'
(11):               tokenize S2.node based on delimiters ' ', '.', '_', '-'
(12):               While (∃ S1.tokens)
(13):               {
(14):                    get next S1.node and set it to i
(15):                    While (∃ S2.tokens)
(16):                    {
(17):                         get next S2.token and set it to j
(18):                         Max Sim (i, j) = 0
(19):                         If ( i == j ) then
(20):                         Max Sim (i, j) = 1
(21):                          else
(22):                              For each ( Lexicon based matching i with j )
(23):                                   If (Lexicon based Similarity (i, j) > Max Sim (i, j)) then
(24):                                        Max Sim (i, j) = Lexicon based Similarity (i, j)
(25):
(26):                         If (Max Sim (i, j) > Threshold) then
(27):                              store Max Sim (i, j) in Similarity_value_Matrix
(28):                    }
(29):               }
(30):               Calculate the sum of maximum similarity values for each row
(31):                                   of the Similarity_value_Matrix and set it as Sum1
(32):               Calculate the sum of maximum similarity values for each column
(33):                                   of the Similarity_value_Matrix and set it as Sum2
(34):               MaxSim (S1.node, S2.node) = (Sum1 + Sum2) /
(35):                                        (amount (S1.tokens) + amount (S2.tokens))
(36):                Store [S1.node, S2.node, MaxSim (S1.node, S2.node)]
(37):                                                  in Mapping Matrix
(38):           }
(39):      }
(40): }
(41): Return Mapping Matrix
```

Fig. 2. Hybrid Semantic Mapping Algorithm

3.1 Input

The algorithm's inputs are two schemas that can be relational databases, XML schemas (XDR, XSD), ontologies (OWL, RDF), or, finally, a combination of these kinds. A schema consists of a set of elements, such as relational tables and columns or XML elements and attributes. We represent schemas by rooted directed acyclic graphs [4]. Schema elements are represented by graph nodes connected by directed links of different types, e.g. for containment and referential relationships. Schemas are imported from external sources, e.g. relational databases or XML files, into the internal format on which match algorithm operates. The algorithm will provide the proper results only if the input schemas are in a directed acyclic graph format. Thus, before its execution, the schemas must be correctly encoded.

3.2 Operation

The algorithm after getting its input scans one-by-one the nodes of the first schema's graph (source) and compares them with all the nodes of the second schema's graph (target). For each node of the source graph, the existence of a matching with a node of the target graph is examined. If such a matching exists and satisfies the similarity thresholds declared by the user then this node is overlooked and the algorithm continues with the next node of the source graph.

If there is no matching at all, or there is a matching but it does not satisfy the given thresholds, then each node of both graphs are tokenized based on the delimiters ' ', '.', '_', '-'. For each token of the source node, the existence of a matching with a token of the target node is examined using a suitable dictionary. The maximum similarity value for each token combination is kept as long as it is greater than the threshold (default value equals 0.5). In this way, a matrix with similarity values between tokens of the source node and tokens of the target node is generated; with dimensions N×M where N, M are the amounts of source and target node's tokens respectively. For each such matrix, the sums of the maximum similarity values of each row (Sum1) and of each column (Sum2) are calculated. Sum1 is the sum of the maximum similarities of the source node towards target node while Sum2 is the reciprocal. Maximum similarity value of the source node and the target node is determined as the mean value of these two sums (lines 34-35 Fig. 2):

$$MaxSim(source\_element, target\_element) = \frac{Sum1 + Sum2}{N + M}$$

These values are calculated for all the nodes of the source graph and then the mapping matrix for the two schemas is returned. Thereafter, regarding the strategy selected, the appropriate matchings are kept in the database.

If, for example, we want to match node "last_name" of schema S1 with node "first_name" of schema S2, the tokenization, described in lines 10 and 11 of Fig. 2, results in "last" and "name" for the S1 node and "first" and "name" for the S2 node. We set token "last" to i and token "first" to j. While "last" ≠ "first", the maximum similarity based on the dictionary is 0.96. It is greater than the threshold, hence it is stored in the similarity value matrix (lines 26-27 of Fig. 2). Then "name" is set to j

and, as above, maximum similarity based on the dictionary is 0.6597222 and it is stored in the similarity value matrix. Now "name" is set to i and "first" to j. For them maximum similarity is 0.77700615 and it is stored in the similarity value matrix. Finally, "name" is set to j and because "name" = "name" maximum similarity is 1. From all the above, the similarity value matrix using the dictionary appears to be:

$$\begin{bmatrix} 0.96 & 0.659722 \\ 0.77700615 & 1 \end{bmatrix}$$

Therefore, Sum1 = 0.96 + 1 = 1.96 and Sum2 = 0.96 + 1 = 1.96. Hence, maximum similarity for these two nodes is (1.96 + 1.96)/4 = 0.98 and it is stored in the mapping matrix.

3.3 Output

The algorithm's output is a matrix containing the matchings between the two input schemas. This matrix contains tuples each of which consists of a source node, a target node and the similarity value. In the previous example, the algorithm returns the tuple (last_name, first_name, 0.98). The matrix is then processed and, depending on the selected filter (strategies), the proper matchings are kept and stored in the database. Selection filters are analyzed in-depth in Section 5.

4 Evaluation on real world schemas

4.1 Matching Quality Measures

To provide a basis for evaluating the quality of an algorithm, the match task has to be performed manually first. The obtained real match result can be used as the "gold standard" to assess the quality of the result semi-automatically determined by the algorithm. Comparing the semi-automatically derived matches with the real matches results in the sets shown in Fig. 3 [16]. False negatives, A, are matches needed but not semi-automatically identified, while false positives are matches falsely detected by the semi-automatic match operation. True negatives, D, are false matches, which have also been correctly discarded by the automatic match operation. Intuitively, both false negatives and false positives reduce the match quality.

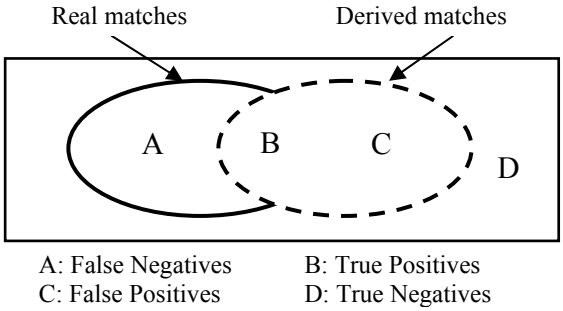

Fig. 3. Comparing real matches and semi-automatically derived matches

Based on the cardinalities of these sets, the following quality measures [4], [5], [16], which we use in our evaluation can be computed:

1. $Precision = \frac{|B|}{|B|+|C|}$
2. $Recall = \frac{|B|}{|A|+|B|}$
3. $FMeasure = \frac{2*|B|}{(|A|+|B|)+(|B|+|C|)} = 2 * \frac{Precision * Recall}{Precision + Recall}$
4. $Overall = 1 - \frac{|A|+|C|}{|A|+|B|} = \frac{|B|-|C|}{|A|+|B|} = Recall * (1 - \frac{1}{Precision})$

One can notice that FMeasure represents the harmonic mean of Precision and Recall. The measure Overall [4], [11] was developed specifically in the schema matching context. The main underlying idea of Overall is to quantify the post-match effort needed for adding missed matches and removing false ones.

4.2 Execution Time measure

This algorithm was not intended to be applied to systems in which execution time is the most important factor, but to find more and better quality matchings. So, quite interesting is both how the execution time fluctuates in relation to the quality of the results and the decrease of this time when previous mapping results are reused.

The measurements showed that the manual matching time can decrease while the quality of results rises. When results from previous execution of the algorithm or other algorithms offered by COMA++ are used, the time can be further reduced.

4.3 Comparison Results

For a correct and reliable evaluation of the algorithm, the schemas must be chosen so as to satisfy every possible case. For this reason the selected schemas vary in size

(from 10 up to 824 elements) and kind. The set of schemas and their characteristics are illustrated in Table 2 [1], [7].

So far, system evaluation was not conducted for different kinds of schemas, i.e. comparison between a relational schema and ontology [16], or was performed only for specific combinations, i.e., comparison between XML and ontology [1]. Regarding the evaluation in this paper, we compared both homogeneous (i.e., relational vs. relational) and heterogeneous (i.e., relational vs. ontology) schemas and we report the results. These results are also compared with those arising from the known approaches.

Schema	*Nodes*	*Attributes-Links*	*Schema*	*Nodes*	*Attributes-Links*
Relational *PO2*	4	20	Relational *PO1*	8	37
Ontology *Order1.owl*	33	36	Ontology *Order2.owl*	32	29
XML *CIDX*	7	27	XML *Excel*	12	36
Relational *bibliography*	2	8	Ontology *Bibliographic*	75	749
Relational *PO2*	4	20	XML *Noris_xdr*	11	54
XML *mondial_xsd*	27	93	Ontology *Mondial_owl*	214	93

Table 2. Characteristics of the evaluated schemas

In Fig. 4, the mean values of four quality measures that were used are shown. Three out of four measures (Precision, Recall, FMeasure) exceed 0.85 (best value is 1) while Overall, which is the most pessimistic measure, is round about 0.75. These results are obviously what we were hoping for. It is difficult to obtain such values when we compare heterogeneous schemas.

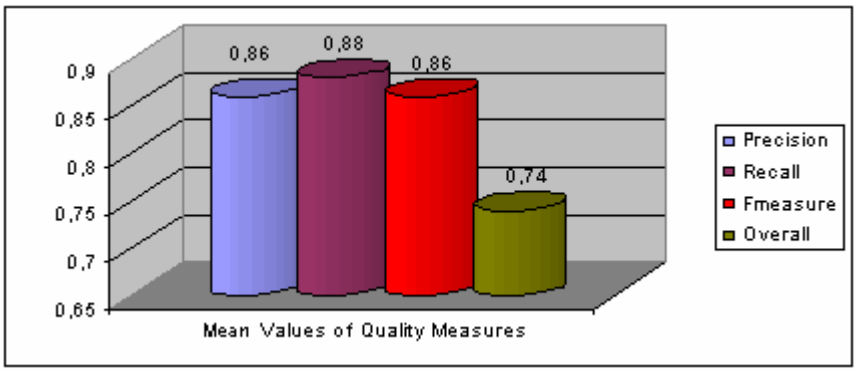

Fig. 4. Mean Values of Quality Measures

As emerged from the evaluation, the algorithm is as good as other approaches and in some cases is even better. The algorithm was compared with other 6 known ap-

proaches [1], [2], [4], [5], [10], [11], [16] and only COMA++ and an approach of XML and ontology matching were able to provide better results in some points. The results are shown in Table 3 and illustrated in Fig. 5. It is obvious that *Overall* has a great significance because for all approaches, except our algorithm and COMA++, it has small values. Although the XML-OWL approach has the best *Recall*, it is not a generic solution as our algorithm.

	Algorithm	***Coma/ Coma++***	***Cupid***	***SF***	***SemInt [9]***	***LSD [6]***	***XML-OWL [1]***
Schema Types	XML, Relational, OWL, a combination of the above	XML	XML	XML, Relational	Relational	XML	XML, OWL
Matching Cardinality	1:1, 1:n, n:1, n:m, m:n	1:1, n:m	1:1, n:1	1:1, m:n	m:n, n:m	1:1, n:1	1:1, n:1
Mean Presicion	0.86	0.93	0.83	0.84	0.78	0.8	0.6
Mean Recall	0.88	0.89	0.48	0.5	0.86	0.8	0.9
F-Measure	0.86	0.90	0.42	0.65	0.81	0.8	0.72
Mean Overall	0.74	0.82	~0.2	~0.5	0.48	0.6	0.3

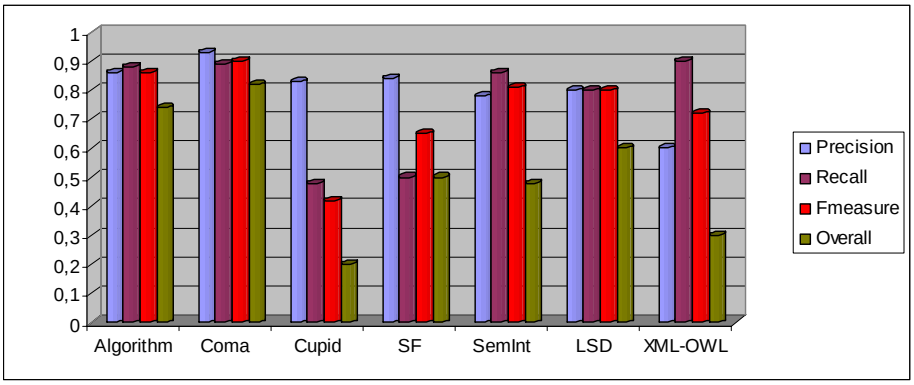

Table 3. Results of the comparison between approaches' evaluation

Fig. 5. Graph of comparison results

5 Tool Presentation

The implemented tool consists of three major segments. The first one is the user interface, the second one is the mapping management tool and the collaboration between the algorithm and the tool, and, last but not least, a platform where the WordNet's parameters are set.

5.1 Interface

User interface can be further divided into four partitions.The first partition enables the user to choose two schemas which would be the input for the algorithm. In this part there are two drop-down lists, one for the selection of source schema and one for the selection of target schema. Both lists include all schemas that are stored in our database as well as in the database the mapping management tool, COMA++, uses.

After the input schemas' selection, the user has to define which elements of the source schema will be compared with the elements of the target schema. This definition is achieved by selection of two thresholds in the second compartment of the interface. Lower similarity threshold cannot be less than zero while upper similarity threshold cannot be greater than 1. If the user does not fill the threshold fields the default values are 0 and 1 respectively. In such a case all elements for which a matching with similarity value between 0 and 1 has been found, either by the algorithm or by COMA++, are ignored.

After that it is time for the user to choose which matchings derived from the algorithm he/she wants to keep in the databases (the algorithm's and the mapping management tool's). In the third part of the interface, the user can select one out of four strategies from a drop-down list. The strategies are the "Maximum Similarity Matching", which gives for each element the matching with the greatest similarity value, the "Matching Above Similarity Threshold", which gives all the matchings with similarity value that exceeds a given threshold, "Distance from Maximum Similarity", which returns all the matchings for an element whose similarity value have a certain distance from the maximum similarity for that element, and finally "All Matchings", which returns all matchings produced.

When all selections have been made, the algorithm may be executed at partition 4 of the interface. In this partition, the user may also choose to deploy the graphic mapping management tool.

5.2 Interaction between Algorithm and Coma++

To provide the user with a complete estimation of the algorithm's results a tool for mapping management, COMA++, was chosen. This tool permits the execution of the majority of the known algorithms developed in the schema mapping domain. Furthermore, it enables the management (creation, deletion, modification, integration, aggregation etc.) of the mappings which resulted from the presented algorithm as well as those implemented in the tool.

Another point worth noting is the collaboration between our algorithm and the mapping tool we use. The algorithm cooperates with COMA++ and uses many of the data in the last's base, such as previous matching results, throughout its execution. Precisely, both system databases must be updated for the correct and efficient execution of the algorithm. Every time the interface is used, an algorithm for the synchronization of the two system databases runs in the background. This algorithm checks which schemas are absent from its database but appear in the tool's database. These schemas are inserted in the algorithm's database because only the schemas which are in both databases appear in the interface.

5.3 WordNet Lexical Database

In this section, we will elaborate on the use of a lexicon for the comparison of the schemas and the definition of their mappings. Our system uses the English dictionary WordNet [12], [13], [14], [15], which is widely used in the information retrieval domain.

WordNet consists of files with terms and it is programmed in such a way that it can convert these files into a database. It also includes search routines and a suitable user interface, which displays information from the dictionary's database. Term files organize nouns, adjectives, verbs, adverbs and pronouns in groups of synonyms. Appropriate code converts these files to a database, which keeps coded relationships between the groups of synonyms.

Information in WordNet is organized in logical groups called "synsets". Every "synset" consists of a list of synonymous words or collocations and pointers which describe the relations between this "synset" and the other ones. A word can exist in more than one "synset", and in more than one part of speech depending on its sense. The words in a "synset" are grouped such that they are interchangeable in some context.

Pointers represent two kinds of relations: lexical and semantic. Lexical relations hold between semantically related word forms. Semantic relations hold between word meanings. These relations consists of (but they are not limited to) hypernyms (..is kind of), hyponyms (is kind of..), antonymy, entailment, meronyms (parts of ...), holonyms (..is part of). Nouns and verbs are organized into hierarchies based on the hypernymy and hyponymy relation between "synsets". The use of these hierarchies is the major point in the similarity values' calculation of the element names of the two schemas. Fig. 6 depicts an example of a hierarchical structure of WordNet's nouns.

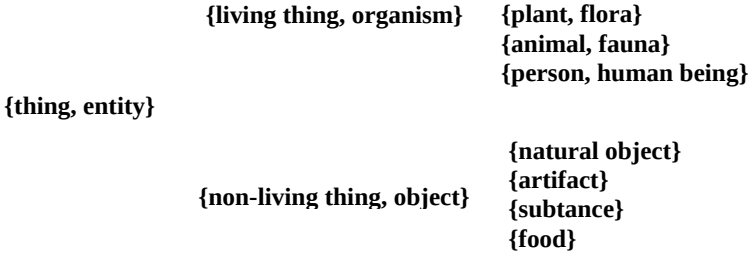

Fig. 6. Example of a hierarchical structure of WordNet's nouns and verbs

The similarity value of two elements is calculated based on the depth that the elements appear in the different Lexicon's hierarchies and is defined by the type of Fig. 7. If, for example, we want to find the similarity value of the elements "animal" and "person", based on the hierarchy shown in Fig. 6, it will be:

(depth=1, depth1=2, depth2=2) distance= ((2-1) / 2 + (2-1) / 2) / 2 => distance= 0,25
sim=1 – 0,25 => sim= 0,75

// Selection based on the number of the ISA links up to the detection point.
// *depth* is the common parental depth, *depth1* is the depth of the first lemma
// and *depth2* is the depth of the second lemma from the top of the hierarchy

$$distance = ((depth1-depth) / depth1 + (depth2-depth) /depth2) /2$$
$$sim = 1 - distance^2$$

Fig. 7. Calculation of the similarity value with the use of WordNet

5.4 Scenario

Assume that a user is interested in finding the matchings between two relational databases (PO1, PO2) that come from the purchase domain. The user has to execute COMA++, via the described interface, to store the two databases in the proper form in the COMA++ database (Fig. 8).

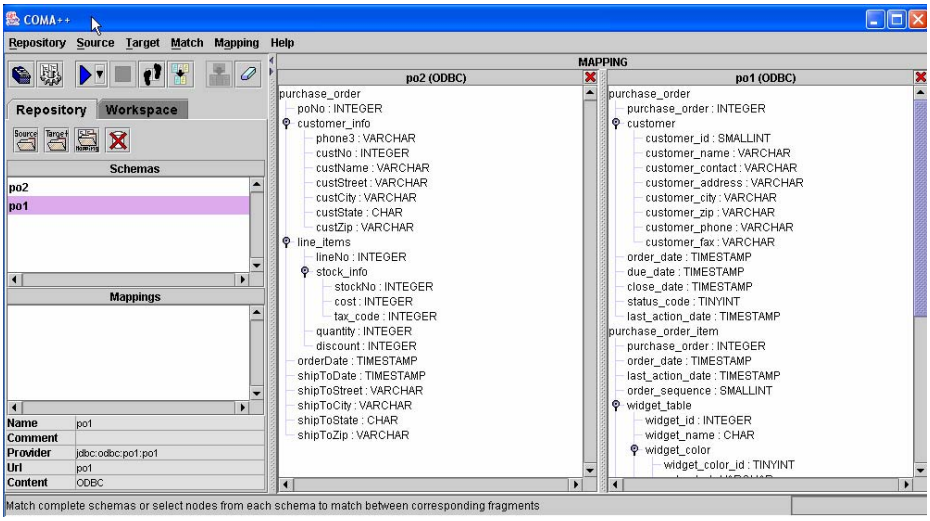

Fig. 8. Storing the databases PO1 and PO2 in COMA++ database

Then, the user has to run again the interface, in order to ensure that the schemas are automatically saved in the algorithm's database via the synchronization routine, as already described. Afterwards, COMA++ is called by the interface and since an algorithm has been executed, the results are saved for reuse by our algorithm (Fig. 9).

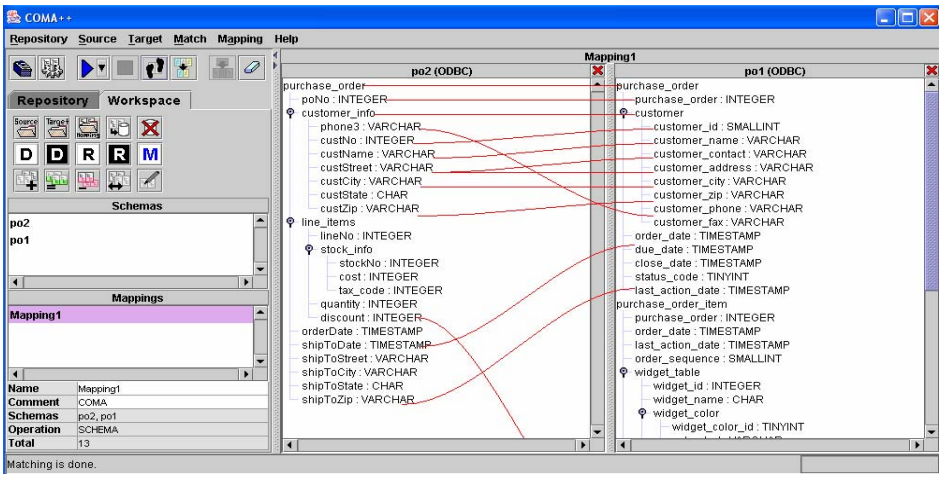

Fig. 9. Results of the comparison between PO1 and PO2 using COMA++

Finally, we select to ignore the matchings with similarity value below 0.92, by proper parameterization of the interface and then the algorithm is executed. After the end of the execution, COMA++ is called for the results' visualization (Fig. 10).

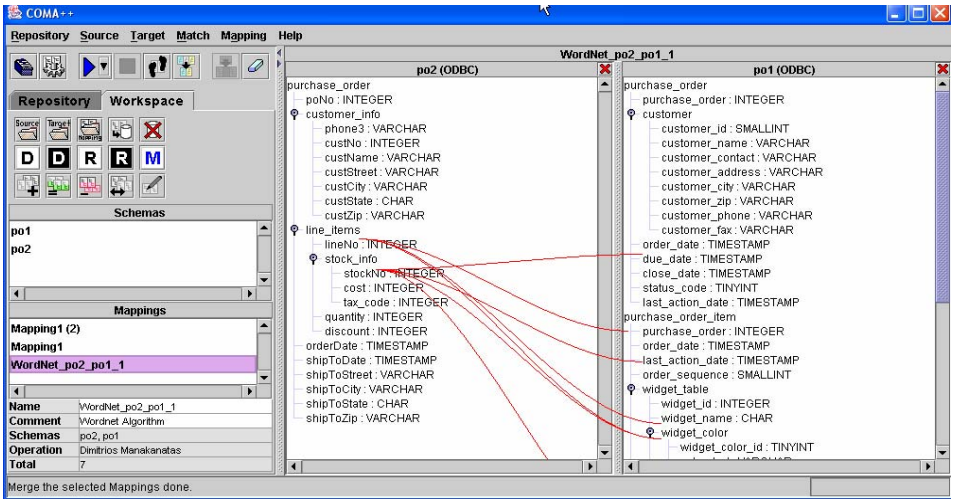

Fig. 10. Algorithm's results

These results can be aggregated with those produced by COMA++ and get the final mapping shown in Fig. 11.

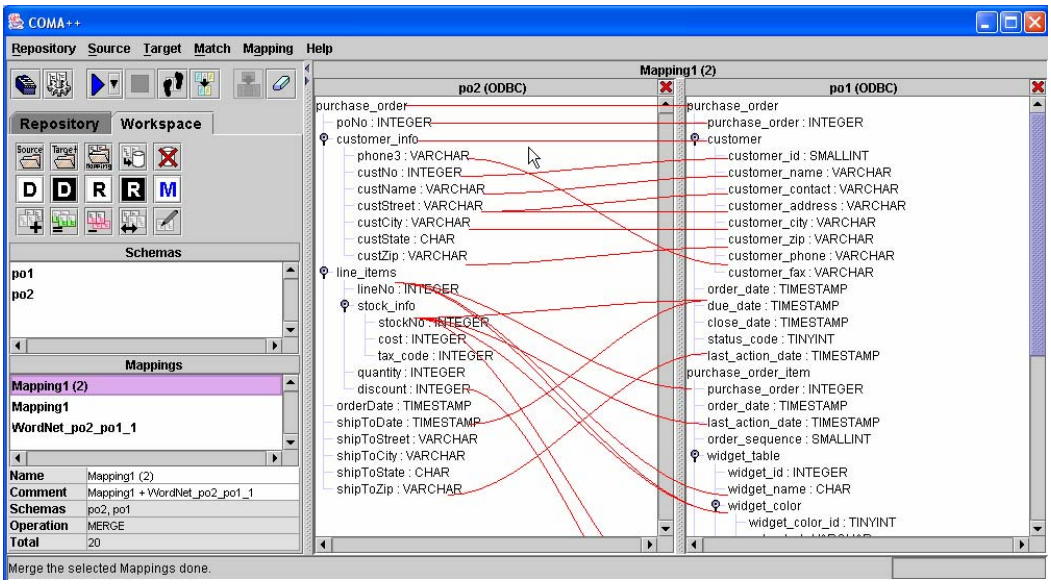

Fig. 11. Final results

6 Discussion and Directions for Future Work

6.1 Semantic Mapping Algorithm

While the algorithm offers better quality in the results, in most cases, than other approaches, it may be modified or extended in order to get an even better performance. Techniques for structural mapping detection could be taken into consideration for the eradication of undesirable results that may occur. Moreover, the algorithm is not able to take into account the disjointness information which could eventuate in more reliable results if exploited in terms of the Lexicon.

If used at the instance level, the presented work could give important results in the case of homonyms' ambiguity. In order to overcome the problems appearing when terms with same spelling but different meaning are to be matched, we could consider the data described by the schema and decide whether a mapping is valid or not.

The tool could also show improvement if properly updated so as to handle the new version of WordNet, 2.1. The algorithm may suffer from slow response time but offers a lot at the semantic level. Matchings that could not be identified so far, like (mother, father, 1.0), (animal, lion, 0.87), (child, family, 0.93), can be now designated with characteristic similarity values. It could provide even better results if there was the right "synset" hierarchy for every application domain in the Lexicon.

6.2 Schema Management

Schema management is an important part in the implemented system. The schemas are modified and stored in the form of a directed acyclic graph in both system databases. After that, different algorithms, depending on the schema's kind, are used for the resulted matchings' storage in the system databases. These algorithms play a fundamental role in the correct storage of matchings and, therefore, in their correct display and reuse.

For ontologies and relational schemas the algorithm for modification and store of the schemas as well as the algorithm for creation and storage of the paths of the graph created no problems. On the contrary, when trying to create and store the element paths from the root node of the graph to an XML schema, the implemented algorithms were not able to find the paths for all nodes. That is why, in this case, the paths are created by COMA++, which, when matching the same schema as source and target, creates automatically the element paths for the schema graph.

In the future, a reliable and efficient algorithm for paths creation can be implemented. It would be very gratifying if this algorithm supports also other XML schema languages, except XSD and XDR.

7 Conclusions

In this paper a generic solution for schema mapping was presented. This work tried to cover as many of the gaps in the domain as possible by providing better quality in the results and matchings that cannot be identified by other approaches. This goal, as shown by the evaluation, was achieved to a large extent. To summarize, an algorithm and, by extension, a tool have been created which can be used in different application domains, such as data warehouses, bioinformatics and query processing systems, providing the user with reliable results.

Acknowledgments

The authors would like to thank Hong-Hai Do and Erhard Rahm for the provision of COMA++. This work has been supported in part by the project PROGNOCHIP, funded by the General Secretariat of Research and Technology, Greece.

References

1. An Y., Borgida A. and Mylopoulos J., Constructing Complex Semantic Mapping Between XML Data and Ontologies, ISWC, 6-10, 2005
2. Aumueller D., Do H. H., Massmann S., Rahm E., Schema and Ontology Matching with COMA++, Proc. ACM SIGMOD international conference on Management of data, 906-908, 2005

3. Bernstein P.A., Melnik S., Petropoulos M., Quix C., Industrial- Strength Schema Matching, SIGMOD Record, 33(4), 38-43, 2004
4. Do H. H., Rahm E., COMA – A System for Flexible Combination of Schema Matching Approach, Proc. VLDB, 610-621, 2002
5. Do H. H., Rahm E., Melnik S., Comparison of Schema Matching Evaluations. Proc. GI – Workshop "Web and Databases", Erfurt, Oct.2002
6. Doan A.H., Domingos P., Levy A., Learning Source Descriptions for Data Integration. Proc. WebDB, 81-92, 2000
7. Giunchiglia F., Shvaiko P., Yatskevich M. An Archive of matching examples C2CProject Matching Examples, http://www.dit.unitn.it/~p2p/Experimentaldesign.html
8. IBM Corp., Clio Web Page, http://www.almaden.ibm.com/cs/clio/demo.html
9. Li, W, Clifton C., Semantic Integration in Heterogeneous Databases Using Neural Networks. Proc. VLDB, 1-12, 1994
10. Madhavan J., Bernstein P.A., Rahm E., Generic Schema Matching with Cupid, Proc. In VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, pages 49--58, San Francisco, CA, USA, 2001
11. Melnik S., Garcia-Molina H., Rahm E. Similarity Flooding: A Versatile Graph Matching Algorithm, ICDE, 117-128, 2002
12. Miller G. A., Beckwith R., Fellbaum C., Gross D., Miller K., Introduction to WordNet: An On-Line Lexical Database, International Journal of Lexicography, 3(4), 235–244, 1990, ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps
13. Miller G.A., Nouns in Wordnet: A Lexical Inheritance System, International Journal of Lexicography, 3(4), 245–264, 1990, ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps
14. Miller G.A., WordNet: A lexical database for English. Communications of the ACM, 38 (11), 39-41, 1995
15. WordNet a Lexical Database for the English Language, http://wordnet.princeton.edu/
16. Yatskevich M., Preliminary Evaluation of Schema Matching Systems, Technical Report DIT-03-028, May 2003.