XML Schema Automatic Matching Solution

Huynh Quyet Thang, Vo Sy Nam

Abstract**—**Schema matching plays a key role in many different applications, such as schema integration, data integration, data warehousing, data transformation, E-commerce, peer-to-peer data management, ontology matching and integration, semantic Web, semantic query processing, etc. Manual matching is expensive and error-prone, so it is therefore important to develop techniques to automate the schema matching process. In this paper, we present a solution for XML schema automated matching problem which produces semantic mappings between corresponding schema elements of given source and target schemas. This solution contributed in solving more comprehensively and efficiently XML schema automated matching problem. Our solution based on combining linguistic similarity, data type compatibility and structural similarity of XML schema elements. After describing our solution, we present experimental results that demonstrate the effectiveness of this approach.

Keywords**—**XML Schema, Schema Matching, Semantic Matching, Automatic XML Schema Matching.

I. Introduction

SCHEMA matching is a manipulation process on schemas that takes two heterogeneous schemas (possibly have auxiliary information) as input and produces as output a set of mappings that identify semantically relation between elements of the two schemas. Several applications relying on schema matching have arisen and have been widely studied in database and artificial intelligent domains, such as schema integration, data integration, data warehousing, data translation, peer-to-peer data management, ontology matching & integration, semantic Web, semantic query processing, etc. [1], [2], [3]. De factor there are great challenges on development solution for schema matching problem. First of all, semantic analysis of the schemas is been needed in this process. In other words, we have to deduce the schema's mining. However the schemas are designed by different creators with different minings and goals. Second, exploitation of information of schema matching meets with seriuos difficulties because we have to exploit a huge amount chaotic informations for example schema description document, schema data source, etc. From that analysis, we can find out that schema matching is difficult to implement manually. The development solution for automatic schema matching is very important and nessesary. Nowadays many approachs and solutions are proposed. Schema matching approaches generally use a combination of such methods to run the matching process. These approaches have been proposed many solutions that combine existing algorithms as well as new developed algorithms to achieve optimal matches.

Nowadays the XML became popular standard for effectively and appropriately data presentation and interchange through Web. The demand of using the XML is increasing and a huge amount of XML data is created. Together with this increasing, a huge amount of XML schemas also is created [20], [21], [22]. Therefore the problem of XML schema matching has been became important and received increasing interest. Up to now, many approaches have been produced to solve automatic XML schema matching problem. The approaches in Database management and Artificial intelligence have given effectively XML schema matching solution. However almost all of them are based on idea to extend the solution applied on exits model into the XML schemas [20], [21], [22]. Only some of works focused directly attention of DTD and XML schemas [12], [19]. Besides, very few solutions considered proposing a solution in order to solve automatic XML schema matching problem comprehensively and efficiently. In this paper, we contributed in proposing such a solution. In this paper we also presented the problem of clustering XML schemas, which is a first important task in data integration systems - typical application of schema matching solution.

In the rest of the paper, section II overviews current approaches for schema matching problem, describe existing matching algorithms as well as analyze strengths and weaknesses of these. Section III-IV describes the approach that we present; consist of data model represent XML schema and element similarity measurement (a combination of linguistic similarity measurement, data type compatibility and structural similarity measurement). Section V presents an evaluation study. Section VI describes briefly the problem of clustering the XML schema of XML data sources. Finally, section VII is the conclusion and future directions.

II. Typical Approaches for Schema Matching Problem

As we have described in section I, up to now there are many solutions to automatic schema matching and this approaches have focused on most of aspects in matching process, from data model, linguistic matching, and structural matching to structural matching. In this section we summarize this works and analyze strengths and weaknesses of these.

Manuscript received May 25, 2007. This work was supported in part by the Vietnam's Ministry of Science and Technology under Grant KHCB2.034.06.

Huynh Quyet Thang is the Head of Software Engineering Department, Hanoi University of Technology, Hanoi, Vietnam (phone: 844-8682595; fax: 844-8692906; e-mail: thanghq@it-hut.edu.vn).

Vo Sy Nam is with the Dept. of Network & System Engineering, Faculty of Information Technology, Hanoi University of Civil Engineering, Hanoi, Vietnam (phone: 844-6280158; namvs@uce.edu.vn).

Based on such parsing, in section III and section IV we propose a solution for XML automated schema matching more comprehensively and efficiently.

There are several approaches on development the schema matching solution. Several works have employed machine-learning techniques to perform matching, such as learner-based approach or neural network approach [2]. Schema matching tools that employ machine learning usually consist of a number of modules, called learners, and a specific module, the meta-learner to directs them. Neural network approaches employ advantage of neural networks to determine the similarity between data sources. Another approach also has been proposed for schema matching problem, object-oriented approach [3]. This approach exploits object-oriented characteristics to discover relationships between attributes in data sources. Besides, several works have used metadata approach, however, this approach does not solve the problem but just shifts the problem to mapping data sources and schemas to mapping ontology [3]. One typical approach to schema matching problem is that rule-based approach. The majority of current schema matching tools employs rules to match heterogeneous schemas [3], [4], [5]. Rule based solutions exploit schema information such as element names, data types, element constraints, structure hierarchy, etc.

Currently all of the methods of schema matching use schema information from the schema as the name of element, constraints of structures. Based on the studies and the surveys on the field of schema matching, we classify the most effective schema matching methods into three categories: linguistic matching, constraint-based matching and structural matching [1], [3]. In linguistic matching phase, existing algorithms generally combine several methods. A common solution is used to compute similarity between element names is that using strings matching [6]. To consider semantic relationships between element names, current linguistic matching solutions generally based on WordNet, a lexical database for English [7]. Authors in [8] have described and evaluated these algorithms completely. However, very few approaches show explicit how they exploit WordNet. A significant linguistic matching approach is proposed in [4] is that parsing element names proceeds in three steps: normalization, categorization and comparison. This approach produces a linguistic matching solution more comprehensively and efficiently. However, despite using any methods linguistic matching may produce high similarity scores even though the nodes do not semantically correspond to each other, thus we need techniques that can adjust such incorrectness.

Current approaches generally consider further schema constraints as a first step to adjust incorrect results that are obtained from linguistic matching phase. One of the most common solutions is consider data type compatibility. XML schema recommendation provides many different built-in data types and regular expressions, therefore, it is probably use such information in order to construct a data type compatibility table that support to linguistic matching phase [4]. In addition, we can use research as in [11] to extend data type compatibility measurement.

Structural matching is used to adjust incorrect matches from matching phases described in sections 3.2 and 3.3. However, up to now, very few studies on schema matching concerning XML's structure, since most of these are studies in database domains and thus consider essentially with relational schemas [1], [3]. Structural matching phase is generally consider structural similarity, in other words, similarity of contexts in which elements appear. There are three kinds of contexts for schema elements: the ancestor-context, the children-context and the leaf-context. Such notion of context is defined based on notion of path in schema graphs [12]. Authors in [12] also have proposed context similarity measurements, but they concerning only DTDs without XML schemas, in addition, they consider only child-context and leaf-context similarity. Cupid [4] and Similarity Flooding (SF) [5] systems have produced notion of context similarity, however none of them consider the three kinds of contexts: Cupid used only leaf-context similarity and SF used only child-context similarity. Authors in [15] have considered structural similarity based on ancestors and descendants relationships between schema elements, however they concerning only version change problems between XML documents, so this approach is not very significant in schema matching problem.

A next natural development of schema matching process is that creating the mapping between similar elements. Such mappings play important role in many applications, such as data integration, data warehousing, especially data transformation. Data transformation. As we known, XML schema features concerning sub typing, abstract types and substitution group mechanisms generally represent designer point of view, so we could use them as a set of meta-data to help the matching process to discover both direct and complex mappings [11]. Several studies (essentially in data integration field) described operations for creating virtual views over schemas. For example, authors in [11] have specified a set of operations for performing queries reformulations in data integration systems. Besides, studies in the area of tree matching concerned with the change detection problem for labeled trees [16]. They propose essentially three edit operations for matching trees: *delete*, *insert*, *relabel*. However, in the XML context, relabel one node into another semantically unrelated node causes an undesirable matching.

Generally, existing XML schema matching solutions still present several limitations, essentially for application domain reasons. For this reason, it is important to present a solution that solving more comprehensively and efficiently XML automatic schema matching problem. In the rest of the paper, we contributed in developing such a solution and hope that this solution is a step towards the optimal matching solution.

III. MODELING XML SCHEMA

Data model that represent XML schema is an important problem in XML schema matching. Data model is able to normalize schemas that are represented by different schema

languages, thus eliminating syntax differences between schemas. Several authors have suggested to modeling XML schemas on the basis of the Unified Model Language (UML) [13]. Another approach design XML schemas according to object-oriented models [14]. Another approach shifts XML schemas into trees [4]. In addition, there are some approaches as described in [1], [3]. The most significant approach is that represent schemas as labeled graphs [5]. Although such methods produce modeling foundations to design XML schemas but they do no show how the way properties and constraints are assigned to schema elements.

A. Schema graph

Based on ideas have produced in [4], [5], [19] in this approach, we produce a model for representing semantically XML schemas in term of directed labeled graphs with constraint sets that be defined over both nodes and edges, called schema graphs. Generally, such model allows to exploit all of features of XML schema, an aspect that be considered in very few existent schema matching solutions (Figure 1). The formal definition of the schema graph is given in [19]. Our interest is limited to method representation and method of installation of XML schemas applying for following calculation of schemas similarity.

B. Nodes and edges in the schema graph

As in [4], [5], [19] we classify schema graph nodes into two kinds: atomic nodes and complex nodes. Atomic nodes are the leaf nodes in the schema graph. Each of them has a simple content, which is atomic value (string, integer, date, etc.), list value or union value. Complex nodes are the internal nodes in the schema graph. Each of them has a complex content, which refers to some other nodes through directed labeled edges. In figure 1, nodes *Name* and *Address* are atomic nodes, while Nodes *University* and *Library* are complex nodes. We also distinguish three kinds of edges indicating containment, association and property relationships. A containment relationship, denoted *c*, is a composite relationship, in witch a composite node ("whole") consists of some component nodes ("parts"). A property relationship, denoted p, specifies the subsidiary attribute of a node. Last, an association relationship, denoted *a*, is a structural relationship, specifying that both nodes are conceptually at the same level. Such relationships are generally bidirectional; they essentially model key/keref and substitution group mechanisms. Three above relationships are represented in figure 1, such as relationships between Nodes *University* and *Name*, *University* and *Address*, *Journal-article* and *Journal* respectively.

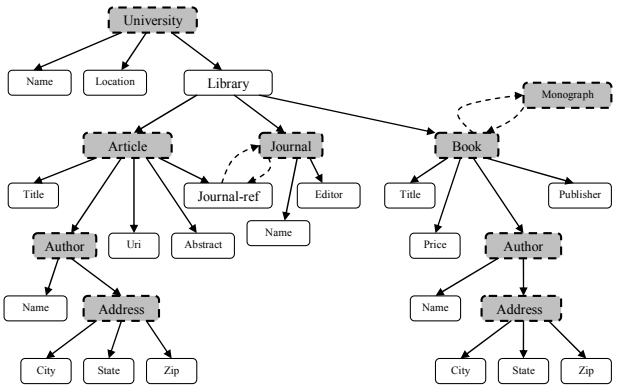

Fig. 1 A schema graph example [19]

C. Constrains in the schema graph

Constraints in the schema graph including constraints over an edge, a set of edges and a node. Typical constraints over an edge are cardinality constraints. Cardinality constraints over a containment edge specify the cardinality of child with respect to its parents. Cardinality constraints over a property edge indicate that attribute of a given node is optional or mandatory. The default cardinality specification is [1..1].

Constraints over a set of edges include: (1) ordered composition, is defined for a set of containment relationship and used for modeling XML schema "sequences" and "all" mechanisms, (2) exclusive disjunction, is applied to containment edges and used for modeling XML schema "choice" mechanism, and (3) referential constraint, is applied to association edges and used for modeling XML schema referential constraint. Such constraints are generally modeled through a join predicate.

The last ones are constraints over a node, including uniqueness and domain constraints. The uniqueness constraint requires each of appearances of a node to have unique content. Domain constraint essentially consider the content of atomic nodes, such constraints are very broad. For example, they can restrict the legal range of numerical values by giving the maximal/minimal values.

IV. Element Similarity Measure

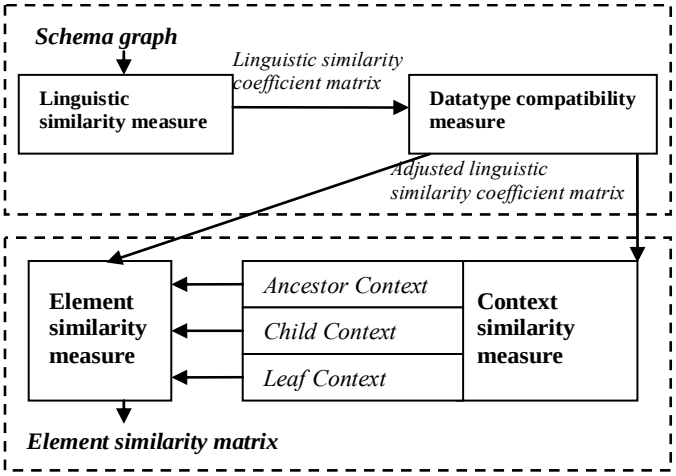

Fig. 2 Element similarity measure process

To computing element similarity, current schema matching approaches generally combine several matching methods. Based on such studies, especially the study in [12], [19], we present a process of element similarity measure as be illustrated in Figure 2. In the following sections, we further detail each phase of such process.

A. Linguistic similarity measure

In this solution, to measure linguistic similarity, we essentially combine two basic solutions that have described in section 2. For schema element names that include abbreviations, acronyms, punctuations, etc., we perform analyzing element names proceeds in three steps: (1) parsing names into tokens; (2) identifying abbreviations, and acronyms; and (3) discarding prepositions, articles, etc.). Detailed description could be found in [4]. In Cupid, linguistic similarity is based on the name similarity of elements, which is computed as a weighted mean of the per-token-type name similarity. This solution presents some limitation because of relying only on string matching methods. In our solution, to identify relation between words, instead of using above methods, we exploit WordNet and construct a domain-specific dictionary. Almost schema matching algorithms use WordNet for text processing. Budanisky proposed a survey of these algorithms, including Hirst and St-Onge algorithm, the Leacock and Chodorow algorithm, Jiang and Conrath algorithm, Resnik algorithm and Lin algorithm [2]. Based on the survey and performed experiments, Budanisky concluded that the Jiang and Conrath algorithm provides highest accuracy. We developed the solution based on Hirst and St-Onge algorithm [8], [9], [19]. We constructed domain-specific dictionaries for experimented schemas. Such method quite applies to other domain-specific schemas. For data type names, we can apply the same algorithm as above, but to simplify this problem, in this solution, we assume that nodes have the same names as their types.

B. Datatype compatibility measure

For datatype compatibility, we essentially use a datatype compatibility table that gives a similarity coefficient between two given XML schema built-in datatypes, such as the one used in [4] (Table I). After computing datatype compatibility coefficients, we can adjust linguistic similarity of atomic nodes that linguistic similarity between them exceeds a fixed threshold (Algorithm 1). Result of above process is an adjusted linguistic similarity matrix for elements in source and target schemas. To simplify the problem, in proposed solution we haven't consider datatype, defined by user. To solving the global problem of datatype defined by user we can use expanding research on datatype compatibility and hierarchical designed datatype [11], [19].

Algorithm 1: linguistic similarity and datatype compatibility measure

```
for (s ∈ N_S)
   for (t ∈ N_T)
     lsim(s,t) = linguistic_similarity(s,t);
for (s ∈ N_AS) //atomic nodes of source
   for (t ∈ N_AT) //atomic nodes of target
     if (lsim (s,t)> th) {
        dsim (s,t) = datatype_compatibility(s,t);
        LS(s,t) = ω_l *lsim(s,t)+ω_t *dsim(s,t);
   }
for (s ∈ N_S)
   for (t ∈ N_T)
     SimMatrix = SimMatrix ∪ (s,t,LS(s,t));
```

Parameters in above algorithm are given based on experimental results from [11]: $\omega_l = 0.5$, $\omega_t = 0.5$.

Fig. 2 Element similarity measure process

C. Structural similarity measure

In our approach, structural matching is performed relies on node context matching with supposition that two nodes are structurally similar if they have similar contexts. The context of a node is defined as the union of its ancestor-context, its child-context and its leaf-context. In the following we describe the basic steps to compare the contexts of two schema elements.

1) Path similarity measure

In order to compare two contexts, we essentially need to compare two paths. Authors in [12] have introduced the concepts of *Path Context Coefficient* to capture the degree of similarity in the paths of two elements. However, this solution has not high matching accuracy. For this reason, here we represent each path as a set of string elements, each element represent a node name, and then use the ideas of path similarity measure have been described in [10], [19]. In [19] is described the combination of query answer and tree pattern to achieve optimal path similarity. To applying the path similarity measure to the schema matching solution we following improvements:

- Relaxing the matching conditions by allow matching paths even if their source nodes do not match and their nodes appear in a different order. In addition, paths can also be matched even if there are additional nodes within the path, meaning that the child-parent edge constraint is relaxed into ancestor-child constraint. Such relaxations are inspired by ideas in query answering to approximate answering of queries (including path queries).
- Allowing two elements within each path to be matched even if they are not identical but their linguistic similarity exceeds a fixed threshold. That is, ordinary string comparison is now relaxed into string comparison that based on similarity threshold.

Based on criteria for matching the paths shown in [10], [19] we propose 4-step method for calculation of path similarity as follow:

- Using a classical dynamic programming algorithm in order to compute the Longest Common Subsequence (LCS) with relaxations that have described above, denoted lcsE, and then normalizing it to obtain a coefficient in [0,1], denoted

TABLE I
DATATYPE COMPATIBILITY COEFFICIENT TABLE

Type (s)	Type (t)	Compatibility coefficient (s, t)
string	string	1.0
string	date	0.2
decimal	float	0.8
float	float	1.0
float	integer	0.9
integer	short	0.8
…	…	…

LCS.

- Next, computing average positioning of the optimal matching, denoted AOP, and then using the LCS algorithm to compute the actual average positioning, denoted AP. Last, we compute coefficient indicating how far the actual positioning is from the optimal one, denoted POS.
- Next, capture the LCS alignment with minimum gaps by using another version of the LCS algorithm as have described in [18], from there compute this score that we note gaps, and then normalizing it to obtain a coefficient in [0,1], denoted GAPS.

Algorithm 2: path similarity measure

```
//Longest Common Subsequence
LCS(P1,P2) = |lcsE(P1, P2)|/|P1|;
//Average position
POS(P1,P2) = 1 - ((AP(P1,P2)-AOP(P1,P2))/(|P2|-
2*AOP(P1,P2)+1));
//LCS with minimum gaps
GAPS(P1,P2) = gaps/(gaps + LCS(P1,P2));
//Length differences
LD(P1,P2) = (|P2|- LCS(P1,P2))/|P2|;
//Path similarity measure
PS(P1,P2) = α*LCS(P1,P2)+β*POS(P1,P2)-
γ*GAPS(P1,P2)-δ*LD(P1,P2);
```

where:

- *$0 \le \alpha, \beta, \gamma, \delta \le 1$.*
- *$\alpha + \beta = 1$ so that **PS** $(P_1,P_2) = 1$ in case of a perfect match.*
- *γ and δ much be chosen small enough so that **PS** cannot take a negative value.*

Parameters in above algorithm are given based on experimental results from [10]: $\alpha = 0.75$; $\beta = 0.25$; $\gamma=0.25$; $\delta=0.2$

- Last, computing the length difference between a source path and LCS between such source path and target path, denoted LD, and then normalizing it to obtain a coefficient in [0,1], denoted LD.

Finally, path similarity measurement is obtained by combining all the above measures (Algorithm 2).

2) Context similarity measure

In the following we describe context similarity measure (including ancestor-context, child-context and leaf-context).

Ancestor-context similarity between two nodes is obtained by comparing their respective ancestor-contexts and weighted by the terminological similarity between them. Concretely, to measure ancestor-context similarity between two nodes, we can use follow formula:

AncestorContextSim(n_1, n_2)=***PS***$((root, n_1),(root, n_2))$*$***LS***(n_1, n_2);

where ***LS*** is gained from Algorithm 1 and ***PS*** is gained from Algorithm 2.

Child-context similarity between two nodes is obtained by (1) computing the terminological similarity between each pair of children in the two children sets of them; (2) selecting the matching pairs with maximum similarity values and (3) taking the average of best similarity values. Algorithm 3 describes detailed process of calculation of child-context similarity between two nodes.

Last, leaf-context similarity between two nodes is obtained by comparing their respective leaves sets. Here, the similarity between two leaves is obtained by combining context similarity (from current node to their leaf) and linguistic similarity between them. Concretely, to measure the leaf-context similarity between two nodes, we can use follow formula:

LeafSim(l_1, l_2) = ***PS***$((n_1, l_1),(n_2, l_2))$ * ***LS***(l_1, l_2);

where ***LS*** is gained from Algorithm 1 and ***PS*** is gained from Algorithm 2.

Algorithm 3 : context similarity measure

```
//Ancestor context similarity measure
AncestorContextSim(n1,n2) =
  PS((root,n1),(root,n2))*LS (n1,n2);
//Child context similarity measure
best_pairs = ∅;
while (SimMatrix ≠ ∅) {
   select (n1k,n2h,sim) where
 sim = max i∈[1,n], j∈[1,m]
{(n1i,n2j,lsim)∈SimMatrix};
  best_pairs = best_pairs ∪ (n1k,n2h,sim);
  SimMatrix = SimMatrix \ {(n1k,n2j,sim)|j=1,…,m}\
              {(n1i,n2h,sim)|i=1,…,n};
}
```

$$\text{ChildContextSim} = \frac{\sum_{(n_{1i},n_{2j},sim)\in best\_pairs} sim}{\max(m,n)};$$

```
//Leaf context similarity measure
for (l1i ∈ leaves(n1))
   for (l2j ∈ leaves(n2) {
      LeafSim(l1i, l2j) =
    PS((n1,l1i),(n2,l2j))*LS(l1i,l2j);
    SimMatrix = SimMatrix ∪
    (l1i,l2j,LeafSim(l1i,l2j));
    }
best_pairs = ∅;
while (SimMatrix ≠ ∅) {
   select (n1k,n2h,sim) where
   sim = max i∈[1,n], j∈[1,m]
  {(n1i,n2j,LeafSim)∈SimMatrix };
  best_pairs = best_pairs ∪ (n1k,n2h,sim);
  SimMatrix = SimMatrix \ {(n1k,n2j,sim)|j=1,…,m}\
              {(n1i,n2h,sim)|i=1,…,n};
}
```

$$\text{LeafContextSim} = \frac{\sum_{(l_{1i},l_{2j},LeafSim)\in best\_pairs} LeafSim}{\max(m,n)};$$

TABLE II
CHARACTERISTICS OF TESTED SCHEMAS [6]

Schemas	1	2	3	4	5
Max depth	4	4	4	6	5
# Nodes / paths	40/40	35/54	46/65	74/80	80/145
# Inner nodes / paths	7/7	9/12	8/11	11/12	23/29
# Leaf nodes / paths	33/33	26/42	38/54	63/68	57/116

The leaf context similarity between two nodes is obtained by (1) computing the leaf similarity between each pair of leaves in two leaves sets; (2) selecting the matching pairs with maximum similarity values and (3) taking the average of best similarity values.

Algorithm 3 describes detailed process of calculation of leaf-context similarity between two nodes.

D. Element similarity measurement

Element similarity is computed by combining all the above measures (linguistic similarity, datatype compatibility and context similarity). The similarity between two nodes is computed by weighted sum of their ancestor context similarity, their child-context similarity and their leaf context similarity. The formula for calculation of element similarity is as follow: ***sim***$(n_1, n_2) = \alpha$****AncestorContextSim***$(n_1, n_2) + \beta$****LeafContextSim***$(n_1, n_2) + \gamma$****ChildContextSim***(n_1, n_2);

where α, β, γ are coefficients specifying role of the similarities (linguistic similarity, datatype compatibility and context similarity).

Here $\alpha + \beta + \gamma = 1$ and $\alpha \geq 0, \beta \geq 0, \gamma \geq 0$.

Depending on the position of the nodes in schema, we can distinguish follow cases: (1) *both nodes are atomic nodes*: the similarity between two nodes is computed by the similarity of their respective ancestor context weighted by their terminological similarity; (2) *one of the two nodes is an atomic node*: the similarity between the two nodes is computed by weighted similarity of their ancestor and leaf contexts (Algorithm 4).

Algorithm 4 : element similarity measure

```
if (both n1 and n2 are atomic nodes)
    sim(n1,n2) = AncestorContextSim(n1,n2);
else if(n1 is a atomic node and n2 is not){
    LeafContextSim (n1,n2) = Σ_{l2i ∈ leaves(n2)} lsim(l2i,n1) / |leaves(n2)|;
    sim(n1,n2) = α*AncestorContextSim(n1,n2) +
                 β*LeafContextSim (n1,n2);
}
else {
    sim(n1,n2) = α*AncestorContextSim(n1,n2)+
                 β*LeafContextSim(n1,n2)+
                 γ*ChildContextSim(n1,n2);
}
//Creating element similarity coefficient
matrix
for (s ∈ NS)
   for (t ∈ NT)
      SimMatrix = SimMatrix ∪ (s,t,sim(s,t));
```

Parameters in above algorithm are given based on experimental results from [12]: α=0.33; β=0.33; γ=0.33.

V. EVALUATION

To implementing the solution we have installed above algorithms using Java and JWNL to exploit the WordNet and package XSOM to analyse XML schemas. To evaluate proposed solution, we used 5 XML schemas for purchase orders taken from www.biztalk.org (also provided in [6]). Table II summarizes the characteristics about such schemas. Look at this table; we can see that except for schema 1, in other schemas, the number of paths is different from the number of nodes, indicating the use of shared fragments in the schemas (i.e., there are association edges in the schema graph).

A. Matching quality measure

In this section, we consider the matching quality of proposed solution based on criteria have described in [5], [6], including *Precision*, *Recall*, *F-measure* and *Overall*. We have implemented all schemas from the source and calculated these criterions. Before dictionary correction the experiment results show that the criterion *Precision*, *Recall*, *F-measure* and *Overall* are 91%, 83%, 84% and 79%. After correction of the dictionary conformable to the domain of schemas, the experiment results show that *Precision*, *Recall*, *F-measure* and *Overall* are 95%, 86%, 88% and 82% (Table III).

B. Comparison with several systems

We have implemented again Cupid algorithm and test with above source schemas. As we known, SF is one of algorithms that have equivalent effect with Cupid. In this comparative study, we do not consider such algorithm because it can be compared with Cupid based experimental results that have described in [5]. In general, all criterion received our solution are better than Cupid algorithm's ones. Such comparative results are illustrated in Figure 3. As in [5], we can see SF consider essentially the similarity of child nodes, so, its matching quality even is lower than Cupid. Whereas, in our solution, we have considered all of ancestor-context, child-context and leaf-context.

VI. CLUSTERING XML SCHEMAS

As shown in the section I, an XML schema clustering is very important problem in data integration system. Solving XML schemas clustering requires combination of techniques and methods. In [12] is described the algorithm for clustering DTD based on calculation element similarity in source DTD

TABLE III
MATCHING QUALITY MEASURE

Measure	Average Result
Precision	95%
Recall	86%
F-measure	88%
Overall	82%

and target DTD. This idea can be applied in to XML schemas clustering.

We propose the sequence for clustering the SML schemas. The sequence includes two phases: (1) Calculation of the XML schemas similarity and (2) Clustering XML schemas.

In the first phase, the similarity between two schemas from the XML source is realized as the sum of all the best element similarity from the elements from two schemas. In the second phase, the clustering XML schemas is realized based on received in phase 1 the matrix of similarities and using algorithm for clustering XML schemas. We have installed proposed 4 algorithms for calculation of similarities and compared the experiment results with [12]. Based on evaluation in [12], we can conclude that clustering the XML schemas based on our solution gives better results.

VII. CONCLUSION AND FUTURE DIRECTIONS

In this paper we have described a solution for automatic XML schema matching problem. In this solution, we have combined several matching methods as well as using several ideas from studies in other fields in order to produce best matching results as well as possible. We implemented the solution and compared our implementation to two others. In future we will improve more linguistic phase as well as improve the performance of matching algorithms, especially linguistic algorithms. Currently, we used Hirst & St-Onge algorithm to exploit WordNet, however, such algorithm requires rather long computing time. Moreover, we plan to test the proposed solution using a broader and more complex data sources as well as compare it with some other solutions except Cupid and SF. One of the remarkable applications in schema matching is that automating translation, such as translating technical documents form English to Vietnamese. Current studies in automating translation generally use tree transformation methods, so their performance is limited. Schema matching could be a suitable solution to solve such problems. This is also one of the future directions that we are performing.

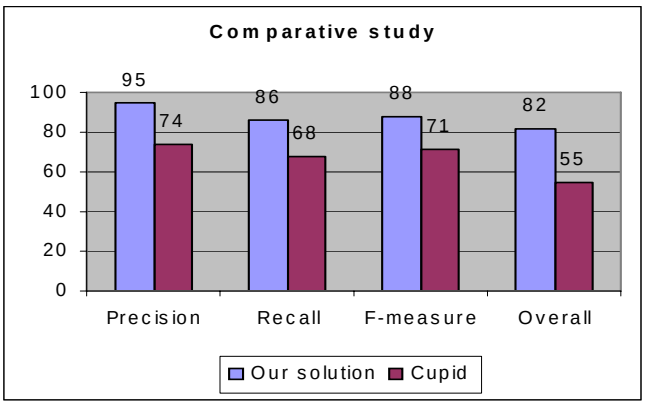

Fig. 3 Comparative study

REFERENCES

[1] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *In VLDB Journal*, pages 10: 334-350, 2001.
[2] A. H. Doan. *Learning to map between structured representations of data*. PhD thesis, University of Washington, 2002.
[3] L. Zamboulis. XML Schema Matching & XML Data Migration & Integration: A Step Towards The Semantic Web Vision. *Technical Report, 2003.*
[4] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with Cupid. *MSR Tech. Report* MSR-TR-2001-58, 2001, Available at: http://www.research.microsoft.com/pubs.
[5] S. Melnik, H. Garcia-Molina, E. Rahm. Similarity Flooding: A versatile Graph Matching Algorithm and its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering, 2002.* Available at: http://dbpubs.stanford.edu/pub/2001-25. (Extended Technical Report, 2001).
[6] H. H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. *In Proceedings of the Very Large Data Bases Conference* (VLDB), pages 610–621, 2001.
[7] A.G. Miller. WordNet: A lexical Database for English. *In ACM 38 (11),* pages 39-41, 1995.
[8] A. Budanitsky and G. Hirst. Semantic distance in WordNet. An experimental, application oriented evaluation of five measures, 2003.
[9] Lexical chains as representations of context for the detection and correction of malapropisms. In: Christiane Fellbaum (editor), WordNet: An electronic lexical database, Cambridge, MA: The MIT Press, 1998.
[10] D.Carmel, N. Efraty, G. M. Landau, Y. S. Maarek, and Y. Mass. An Extension of the vector space model for querying XML documents via XML fragments. Second Edition of the XML and IR Workshop, In SIGIR Forum, Volume 36 Number 2, Fall 2002.
[11] L.Xu. Source Discovery and Schema Mapping for Data Integration, PhD thesis, 2003.
[12] Mong Li Lee, Liang Huai Yang, Wynne Hsu, Xia Yang. XClust: Clustering XML Schemas for Effective Integration, in 11th ACM International Conference on Information and Knowledge Management (CIKM), McLean, Virginia, November 2002.
[13] N. Routledge, L. Bird and A. Goodchild. UML and XML Schema, ADC'2002, 2002.
[14] R. Xio, T. Dillon, E. Chang and L. Feng (2001). Modeling and Transformation of Object Oriented Conceptual Models into XML Schema. DEXA 2001, LNCS 2113, pages795-804, 2001.
[15] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML Documents. In ICDE, 2002.
[16] D. Shasha, J. Wang, K. Zhang, and F. Shih. Fast algorithms for the unit cost editing distance between trees. In Journal of Algorithms, pages 581-621, 1990.
[17] S. Amer-Yahia, S. Cho, D. Srivastava, "Tree Pattern Relaxation" EDBT'02, 2002.
[18] E.W. Myers. Incremental alignment algorithms and their applications. TR 86-22, Department of Computer Science, University of Arizona, 1986.
[19] A. Boukottaya, C. Vanoirbeek. Schema Matching for Transforming Structured Documents. In DocEng'05, 2-4, 2005.
[20] XML Schema Part 0: Primer, W3C Recommendation, 2004. Available at: *http://www.w3.org/TR/xmlschema-0/.*
[21] XML Schema Part 1: Structures, W3C Recommendation, 2004. Available at: *http://www.w3.org/TR/xmlschema-1/.*
[22] XML Schema Part 2: Datatypes, W3C Recommendation 2004. Available at: *http://www.w3.org/TR/xmlschema-2/.*