EXSMAL: EDI/XML semi-automatic Schema Matching ALgorithm

Uddam CHUKMOL[*], Rami RIFAIEH[+], Nabila Aicha BENHARKAT[+]
[*]*Département Informatique - Institut de Technologie du Cambodge, Phnom Penh, Cambodia*
[+]*LIRIS-CNRS, National Institute of Applied Science of Lyon, Lyon, France*
E-mail: uddam.chukmol@itc.edu.kh, rami.rifaieh@insa-lyon.fr, nabila.benharkat@insa-lyon.fr

Abstract

We describe a schema matching algorithm EXSMAL that automates the semantic correspondence discovery between the EDI (Electronic Data Interchange) messages of various standards (EDIFACT, SWIFT...) by using XML Schema as the pivot format. This algorithm takes two schemata of EDI messages as the input, computes the basic similarity between each pair of elements by comparing their textual description and data type. Then, it computes the structural similarity value basing on the structural neighbors of each element (ancestor, sibling, immediate children and leaf elements) with an aggregation function. The basic similarity and structural similarity values are used in the pair wise element similarity computing which is the final similarity value between two elements.

1. Introduction

EDI is characterized by the possibility of sending/treating messages between information systems without any human intervention. With growing business, many companies have to treat different type of messages and standards. Therefore, a large number of translations are needed in order to enable the communication between an enterprise and its suppliers and clients [6]. Although the use of XML has simplified the task of data exchange, the problem of data heterogeneity remains largely unresolved. For the same kind of data, independent developers often design XML syntaxes (i.e. messages) that have very little in common in terms of employed vocabulary and presentation. In order to simply manage this representation incompatibility, we suggest automating the similarity findings. We explore in this paper the development of an EDI/XML semi-automatic Schema Matching Algorithm. The algorithm uses XML Schema, as the pivot format, to represent the schemas of EDI messages.

2. Related Work

We are only interested in similarity matching that helps to identify the semantic correspondence between elements of the input schema or the branching diagram of the messages. In the literature, we can find three types of matching algorithm: instance based matching, representation based or schema matching and usage based or ontology matching. In all these approaches, we are only interested by representation based matching since EDI branching diagrams, i.e. usage guide, are very likely to schemas. In the schema matching, some prototypes have been developed such as [1], [3], and [4]. Nonetheless, they are not suitable to the matching of EDI messages [5], [7] and [2]. Indeed, EDI messages do not have significant field names (e.g.: NAD represents the Address in EDIFACT and 32A represents the amount of the transfer with SWIFT). Though, an element of an EDI message is defined with: textual description (a short text describing the element's role in the message), data type, constraints (condition depending on the instance value of the element and can influence the value restriction of another element in the message), status (an information indicating if the element's existence in the message is mandatory, optional…), cardinality (the possible occurrence number of an element within another element in a message). Another important fact concerns the meaning variation of an element due to its location in the message (structural influence).

Therefore, we have to identify a new similarity algorithm, which takes into consideration the specific characteristics of EDI message's branching diagram

Proceedings of the Seventh IEEE International Conference on E-Commerce Technology (CEC'05)
1530-1354/05 $20.00 © 2005 **IEEE**

expressed with XML Schema. Our choice for the XML schema is motivated by its potential to define the structure and semantics of an EDI message.

3. Our Approach

In this section we describe EXSMAL (EDI/XML semi-automatic Schema Matching ALgorithm) proposed as a solution for the EDI message's schema matching. The criteria for matching will include data-type, structure, and elements descriptions. Other information related to an element (constraints, status, cardinality) will be taken into account for the future extension of this work. The algorithm is briefly described in **Figure.1.**

Input: S, T: two XML Schemata
Output: set of triplets <S_i, T_j, V_{sim}>
With S_i: an element of S
T_j: an element of T
V_{sim}: the similarity value between S_i and T_j
Matching(S, T) {
Convert S and T to tree
For each pair of elements <S_i, T_j>, compute {
Basic similarity value.
Structural similarity value.
Pair-wise element similarity value. }
Filter: eliminate the element pairs having their V_{sim} below an acceptation threshold value. }

Figure.1: Short description of EXSMAL

3.1. Basic Similarity

This similarity is the weighted sum of the textual description and data type similarity. We calculate the basic similarity between a pair of elements, each of which comes from the input schema. In effect, we deal with a subset of element criteria; an element has a strong basic similar value with another if their textual description and data type are strongly similar.

We can compute the basic similarity of two elements *s* and *t* by using the following formula:

$$basicSim(s,t) = descSim(s,t)*coeff\_desc + coeff\_type*dataTypeSim(s,t)$$

where $coeff\_desc + coeff\_type = 1$
$0 \leq coeff\_desc \leq 1$ and $0 \leq coeff\_type \leq 1$.

3.1.1. Textual Description Similarity. We choose to use the textual description associated with each element instead of element name. In effect, element names are not useful for comparing EDI message elements because they are neither significant nor readable. This similarity indicates how much two elements are similar according to their textual description. We use the information retrieval technique to solve this problem. From each description to compare, we extract a terms vector containing every term with their associated term frequency in the description. We, then, compute the cosine of the two terms vectors to evaluate a part of the pair wise description similarity. This option is not sufficient to determine the textual description similarity because it takes into account only the term frequency in both descriptions. Therefore, we add another computing to this description comparison by supposing that all the textual description associated with every element of the target schema forms a corpus, which will be indexed. With every single description extracted from a source element, a query which considers the terms order in the description is formulated to query the above index in order to get a set of scores indicating how much it is relevant to the descriptions in the corpus. The score and the description affinity resulted from the vectors cosine computing will be finally used to calculate the description affinity between two given elements.

3.1.2. Data Type Similarity. We used a static matrix defining the XML schema primitive data type affinity. The values given as the data type affinity between two elements is obtained from the empirical study on those data type format and value boundary. These similarity values help to obtain the basic affinity degree of two comparing elements' types.

3.2. Structural Similarity

The structural similarity is computed by using two modules: the structural neighbors computing and the aggregation function ***agg***. This computing is based on the fact that two elements are structurally similar if theirs structural neighbors are similar.

3.2.1. Structural Neighbors. The structural neighbors of an element *e* is a quadruplet *<ancestor(e), sibling(e), immediateChild(e), leaf(e)>* in which:

- Item[1](e)=ancestor(e): the set of parent elements from the root until the direct parent of *e*
- Item[2](e)=sibling(e): the set of sibling elements that share the same direct parent element as *e*
- Item[3](e)=immediateChild(e): the set of direct descendants of *e*
- Item[4](e)=leaf(e): the set of leaf elements of the sub-tree rooted at *e*.

The choice of the structural neighbors of an element is related to many structural observations that we can summarize as follows:

- Ancestor elements influence their descendants meaning, however, they do not define the entire structural semantic of a given element.
- Moreover, two elements can perfectly share the same ancestral structure but differ by the influence from their siblings.
- To reinforce the exact semantic of an element, we choose to ponder the immediate children because they define the basic structure of the parent element and the choice of the last level descendant will help us to go through the finest-grained content or intentional detail of an element.

8.1.1. Structural Similarity Value Computing. Let *s* and *t*, two elements to match and *C(s)*, *C(t)* the structural neighbors of *s* and *t* respectively.

- C(s)=<ancestor(s), sibling(s), immediateChild(s), leaf(s)> the structural neighbors of s
- C(t)=<ancestor(t), sibling(t), immediateChild(t), leaf(t)> the structural neighbors of t

Let:

- ancSim(s,t): ancestor item similarity (between ancestor(s) and ancestor(t))
- sibSim(s,t): sibling item similarity (between sibling(s) and sibling(t))
- immCSim(s,t): immediate child item similarity (between immediateChild(s) and immediateChild(t))
- leafSim(s,t): leaf item similarity (between leaf(s) and leaf(t))

The structural similarity value of two elements *s* and *t* depends on the similarity value resulting from the comparison of each pair of structural neighbors items (*ancSim(s, t), sibSim(s, t), immCSim(s, t)* and *leafSim(s, t)*). Therefore, the structural similarity value is computed in function of the ancestor item, sibling item, immediate child item and leaf item's similarity.

The similarity value of each structural neighbors items' pair is computed by using the function *agg(M, thr)* which take a matrix *M* and a threshold value *thr* ∈ [0, 100] as input. It returns the aggregated value of the input matrix *M* (see **Figure.2**).

```
Let M be a Matrix.
Input thr the threshold value defined by the user.
For Item[x](E1i) ∈  Item[x](e1) {
For Item[x](E2j) ∈ Item[x](e2) {
    M[Item[x](E1i)][ Item[x](E2j)] =
    sim_base(Item[x](E1i), Item[x](E2j)); }   }
sim_Item[x](e1, e2) = agg(M, thr);
```

Figure.2: Structural neighbors item's pair similarity

The function *agg* uses the arithmetic mean (*avg*) and the standard deviation (*sd*) measures of the descriptive probability to compute the variation coefficient (*vc*) of all the values in *M*. Thus, *M* forms a population that contains only the basic similarity values. We use the standard deviation of the arithmetic mean as dispersion measure because it is sharply more exact than others dispersion measures (inter-quartile range, variance, etc). We compute the arithmetic mean *avg* and standard deviation *sd* of *M* respectively with:

$$avg = \frac{\sum_{i=1}^{|ancestor(s)|}\left(\sum_{j=1}^{|ancestor(t)|} M[s_i][t_j]\right)}{|\,ancestor(s)\,| \times |\,ancestor(t)\,|}$$

and

$$sd = \sqrt{\frac{\sum_{i=1}^{|ancestor(s)|}\left(\sum_{j=1}^{|ancestor(t)|}\left(M[s_i][t_j] - avg\right)^2\right)}{|\,ancestor(s)\,| \times |\,ancestor(t)\,|}}$$

We compute the variation coefficient *vc* of *M* by:

$$vc = \frac{sd}{avg} \times 100$$

By comparing the calculated variation coefficient with the *thr* value given by a user, *agg* decides if the arithmetic mean of *M* will be the aggregated value of *M* or not. Wishing that we get the small dispersion of all the values in *M* around its arithmetic mean, the main target of this *agg* function is to get a descriptive value from a set of values. With the value *thr* given by the user we can adjust the aggregated value of the matrix *M* by eliminating some low values interfering in the arithmetic mean computing.

If the user gives $thr \geq vc$, then *agg* returns *avg* as the aggregated value of *M*. If the user gives $thr < vc$, we will eliminate all the values from *M* below: $avg\left(1 - \left(\frac{thr}{100}\right)\right)$ interfering in the arithmetic mean computing. We obtain a subset of values in *M* and apply again the aggregation function. We apply this computing to all the structural neighbors' items similarity (*ancSim(s,t)*, *sibSim(s, t), immCSim(s, t) and leafSim(s, t)*).

Therefore, the structural similarity value between two elements *s* and *t*, *structSim(s, t)*, is computed with the following formula:

$$structSim(s,t) = ancSim(s,t)*coeff\_anc + sibSim(s,t)*coeff\_sib + immCSim(s,t)*coeff\_immC + leafSim(s,t)*coeff\_leaf$$

Where $0 \leq coeff\_anc \leq 1$, $0 \leq coeff\_sib \leq 1$, $0 \leq coeff\_immC \leq 1$, $0 \leq coeff\_leaf \leq 1$,

And

coeff_anc+coeff_sib+coeff_immC+coeff_leaf=1

However, to make our structural processing flexible and complete, we observe the structural neighbors of each pair of elements before deciding which value to use (e.g. matching two neighbors without sibling elements will have to make the *coeff_sib* value available for other coefficients values). Thus, we equally dispatch the value of *coeff_sib* over other three coefficients. Finally the value of the remained three coefficients will be the sum of its initial value with a part of value from the *coeff_sib*.

To sum up, depending on the *thr* value, we will have the different aggregated value of the same matrix. The rest of the structural neighbor's item similarity (*sibSim(s, t), immCSim(s, t) and leafSim(s, t)*) will be calculated the same way as *ancSim(s, t)* with help from the function *agg*.

3.3. Pair-Wise Element Similarity

The pair-wise element similarity value is computed as the weighted sum of the basic similarity value and the structural similarity value. It's proposed as the final similarity value for a pair of elements in our approach.

Let *s* and *t*, two elements to match. The pair wise element similarity of *s* and *t* is computed by the following formula:

$$similarity(s,t) = basicSim(s, t) * coeff\_base + structSim(s, t) * coeff\_struct$$

Where $0 \le coeff\_base \le 1$,

$0 \le coeff\_struct \le 1$,

And $coeff\_base + coeff\_struct = 1$

3.4. Filtering

This is the last step in our algorithm consisting of eliminating all the pairs of elements with the pair wise element similarity value below the value thr_{accept} given by the user ($0 \le thr_{accept} \le 1$).

As we are using many coefficients in our algorithm, we suggest a method to calculate the best value of each coefficient. We provide the possibility for the user the run the performance batch which helps them to determine the good set of coefficients to use a process of matching.

9. Conclusion

This algorithm can be classified among the schema based approaches that combines the structural similarity and the textual description similarity. It can differentiate from other approaches with the following particularities:

- It treats the textual description of the elements, which is richer than other approaches treating only the elements names. In effect, this choice was directed by the particularity of EDI branching diagram. We used some known techniques in Information Retrieval techniques to find the similarity of two elements' descriptions.
- It fully treats the structure of an element by covering the structural neighbors' items: ancestors, siblings, immediate children, and leaves.

We developed a prototype implementing EXMAL and some more tools helping the users to find out the best set of coefficients to use. Our prototype can be improved by allowing user's intervention after the matching process in order to define the mapping expression between the matched elements (i.e. applying the next step after schema matching).

As future works, we consider using all the elements of EDI's branching diagram (e.g. constraint, status, cardinality, etc.). We envisage enlarging our performance test with a larger number of real-world EDI message schemata.

10. References

[1] Hong-Hai Do and E. Rahm. "COMA - A system for flexible combination of Schema Matching approaches". In Proceeding of the 28th VLDB Conference, August, 2002, Hong Kong, China. pp. 610-621.

[2] Hong-Hai Do, S. Melnik and E. Rahm. "Comparison of Schema Matching Evaluations". In Proceedings of the GI Workshop "Web and Database", October, 2002, Erfurt. pp. 221-237.

[3] J. Madhavan, P. A. Bernstein and E. Rahm. "Generic Schema Matching with Cupid". In Proceedings of the 27th VLDB Conference, 2001, Rome, Italy. pp. 49-58.

[4] S. Melnik, H. Garcia-Molina and 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 (ICDE), 2002, San Jose, California, USA.

[5] E. Rahm and P.A. Bernstein. "On Matching Schema Automatically". Technical Report 1/2001, Department of Computer Science, University of Leipzig, Germany,

[6] R. Rifaieh and N. A. Benharkat. "An Analysis of EDI Message Translation and Message Integration Problem". In Proceedings of the CSITeA-03, June, 2003, Rio De Janeiro, Brazil, 8 pages.

[7] M. Yatskevitch. "Preliminary Evaluation of Schema Matching Systems". Technical Report, DIT-03-028, November, 2003, Department of Information and Communication Technology, University of Trento, Italy.