

Matching of XML Schemas and Relational Schemas

Sergio L. S. Mergen¹

Carlos A. Heuser¹

¹Instituto de Informatica
(UFRGS)

mergen@inf.ufrgs.br, heuser@inf.ufrgs.br

Abstract

XML is widely used for data exchange between relational databases. Generic exchange tools are based on mappings between the elements of both schemas. Despite the benefits of using generic tools, the manual definition of the mappings can become a time consuming and error-prone task. Given this scenario we propose algorithms for the automatic matching of XML schemas(DTDs) and relational schemas. Considering we restricted the schemas formats allowed for DTDs and relational schemas, the algorithms(matchers) we proposed take advantage of the particularities of each schema to provide a more precise matching than the other solutions available. We also assess the effectiveness of our matchers running experiments using real world schemas on the movies domain.

1. Introduction

As companies move to "paperless" data storage, in which data is store in digital databases, the need for exchanging data among databases has become more important. Therefore, techniques for the conducting the data exchange consistently are a major need. One of the main problems resides in communicating data between companies whose databases schemas are structurally and semantically different. One possible way to resolve these heterogeneities is converting data to a common data representation.

XML is widely used for that end because it provides a simple textual mechanism for data exchange, with powerful structure capabilities. With the adoption of XML as the intermediary data type and considering most data is still stored and maintained in relational databases, the problem of data exchange between heterogeneous databases can be translated to the problem of data exchange between a relational database and an XML representation.

Some tools carry out the data exchange by the definition of mappings between the elements of both schemas. For instance, tools like [1] and [12] provide the user with an interface to declare the mappings between elements of a XML schema and columns of a relational schema. The decision of using such tools frees the programmers from developing ad-hoc solutions whose use is limited to a constant set of schema (whenever a new schema is to participate of the data exchange, the code should be modified).

The manual definition of the mappings can become a time consuming and error-prone task in cases where the user must map a great number of large schemas. (Semi)automatic schema matching can be applied to this scenario by providing a fast and convenient way to find correspondences between XML schema elements and relational schema columns requiring none or minimum user interference.

Schema matching algorithms (matchers) are used to calculate similarities between the elements of two schemas. One common way to find the similarities between the elements is to compare the linguistic relation between them. This technique can be used as a basis for the development of more complex match approaches, such as the ones that used structure information. In this paper we identify some linguistic relations that can be applied exclusively for the matching of XML schemas elements and relational schemas columns. Based on such linguistic relations, we have implemented a linguistic matcher that improves the matching accuracy by finding correspondences that are not found by other solutions.

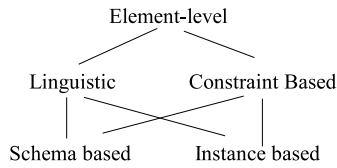


Figure 1: Element-Level Matcher classification (Adapted from [13])

Additionally, we also present an innovating structure matcher, specifically designed for handing atomic elements matching in a scenario where the schemas can be morphologically different, such as XML schemas and relational schemas. This structure matcher has the purpose of improving the results obtained by the linguistic matcher. To assess the effectiveness of our method we have also ran experiments that demonstrate in which cases our solution overcomes others.

This paper is organized as follows. Section 2 shows work related to our approach. Section 3 shows some characteristics of the DTD and the relational schema graphs. Section 4 describes the techniques proposed to achieve the matching of the schemas. In section 5 we present experimental results when applying the matcher to real-world schemas in the movie domain. Finally we summarize our work and present some plans for the future in section 6.

2. Related Work

In this section we present an overview of the matching approaches existing in the literature, and discuss their contribution to the specific problem of matching XML schemas and relational schemas.

We present the matching approaches based on the classification suggested by [13], in which the matchers are divided in element-level (see figure 1) and structure-level.

The element-level matchers can be further divided in linguistic matchers and constraint matchers. Linguistic matchers can be applied either to the schemas or to data instances. For the first case, the linguistic matcher looks for similarities in the name of the schema elements. In the second case, the matcher looks for the similarity in the data values, using IR techniques such as word frequencies [3].

Element-level constraint matchers can also be divided into schema level and instance level. An example of schema level constraint that can be exploited is the data types of the nodes (*Data Type Matchers*). The instances can also present constraints, such as a range of values for a specific data field.

The element-level matchers are suitable for the task of finding correspondences between XML schemas and relational schemas when the similarity lies in the properties of the atomic level of the schemas. However, element-level matchers alone are not helpful when the similarity is in the properties of the nodes above the atomic level. The limitations of these matchers urge the need for additional matching techniques that exploit the schemas structure.

Some structure matchers can be generally applied to graph schemas [10, 5, 11, 6]. The standard behavior of these matchers follows two steps: The similarity between two nodes is firstly pre-computed by a particular lexical matcher (linguistic matcher, data type matcher,...). In the next step this similarities are propagated in the tree. Two propagation directions are allowed. In bottom-up propagation the children nodes similarities affect the similarities of the parent nodes. In top-down propagation, the parent nodes similarities affect the children nodes.

This technique presents good results in discovering matches when the schemas are represented in the same way, but its effectiveness is questionable when the two schemas are morphologically different. For instance, DTDs XML schemas are typically trees, and relational schemas are better represented as graphs. Moreover, the way the information is structured differs from one model to the other, so the

propagation of scores may not present accurate results.

There are also methods that glean knowledge from past matching experiences to improve match accuracy [2, 5, 9]. Generally speaking, this kind of method calculates how trustfully a specific matching technique can be used to match two schemas.

3. Schema Representation

The XML schemas and the relational schemas are represented in our work as acyclic graphs. The matchers use the graphs in order to perform the comparisons between the nodes from both representations. The figure 2 shows the description of a relational schema and a DTD, along with the graphs generated for them.

Next we introduce some definitions that clarify the conversion of a relational schema into a graph.

Definition 1 (Relational Schema) Let $T = \{t_1, \dots, t_n\}$ be the set of tables from a relational schema RS . Each table t_i has a set of columns $tc_i = \{c_1, \dots, c_n\}$. Additionally, let $C = \{c_i\}$ be a set containing all the columns of the relational schema. Let the tuple $k = (t_i, t_j)$ be a foreign key relationship between two tables. Further, the set FK contains all the foreign key columns of RS .

Definition 2 (Relational Schema Graph) Let the triple $RG = \langle N, E, name \rangle$ be the graph generated for a RS , where $N = T \cup (C - FK)$ are the nodes of the graph, $E \subseteq T \times (T \cup (C - FK))$ are the edges, and $name : N \rightarrow \eta$ (where η is a set of names), is the naming function.

Every t_i turns into a table node. Likewise, every c_i that is not a foreign key turns into a column node.

The name of the nodes are defined by the *name* function. The edges connect tables with tables and tables with columns. The edges between tables and columns are described in definition 3.

Definition 3 (Containment Edges) Let $ed(t_j \rightarrow c_i)$ be a directed edge connecting a table node to a column node. Then, $\forall c_i \in tc_j \exists ed(t_j \rightarrow c_i)$.

The directed edges are called containment edges. As a matter of fact, we borrow the definition of containment edges from [10]. Containment edges are directional edges that indicate that a node is contained inside another node. One property of such edges is that they can only be contained by one parent node.

Definition 4 (Foreign Key Rule) Let $ed(t_i \leftrightarrow t_j)$ be a bidirectional edge connecting two table nodes. Then, $\forall k = (t_i, t_j) \exists ed(t_i \leftrightarrow t_j)$.

The definition 4 states how table to table edges are created. In [10] the edges connecting table are called aggregation edges. The difference between containment edges and aggregation edges is that the latter allow multiple parents. In our approach, the bidirectional edges are called interrelated edges. We rather think of the edges connecting table nodes as a mean of indicating the nodes have some kind of relationship between them, without defining which side of the relationship contains the other. The notion of interrelated edges is important for our matcher in the sense they allow greater navigability through the nodes due to its bidirectional nature.

Note that in definition 2 the foreign key columns are not represented as nodes, once they represent indirections to other tables, and these indirections are already represented by the interrelated edges. However, there could be cases where the foreign key would be helpful for inferring similarities. We intend to give foreign keys a better treatment in future work.

Definition 5 (Associative Tables Rule) Let $ed(t_i \leftrightarrow t_j)$ be a bidirectional edge connecting two table nodes. Further, let te_i be the interrelated edges of t_i . Then, $\forall (t_j \in te_i, t_k \in te_i) | tc_i \subseteq FK, (\exists ed(t_j \leftrightarrow t_k)) \wedge (N = N - t_i)$.

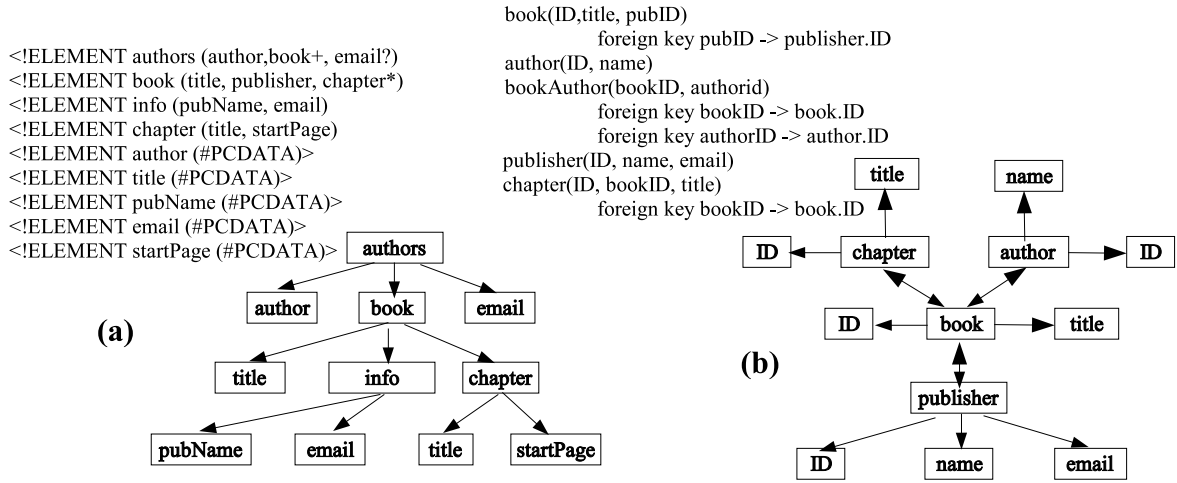


Figure 2: Graphs for DTD and Relational Schema

The definition 5 states that the associative tables are eliminated from the graph. Instead, the table nodes supposed to be linked with the associative tables are linked to each other (see the edge connecting table node **book** and table node **author**).

Next we introduce some definitions that clarify the conversion of a DTD into a graph.

Definition 6 (DTD Schema) Let e_i be a element from a DTD D . Let $ec_i = \{e_1, \dots, e_n\}$ be the set of children elements of e_i . Let $size(ec_i)$ be the number of elements in ec_i . Further, $EP = \{e_i \mid size(ec_i) > 0\}$, and $EC = \{ec_i\}$.

The definition above describes EP as a set containing all the elements that have children. EC is a set containing all the children presented in D .

Definition 7 (DTD Graph) Let the triple $DG = \langle N, E, naming \rangle$ be the graph generated for a D , where $N = EP \cup EC$ are the nodes of the graph, $E \subseteq EP \times EC$ are the edges, and $naming : N \rightarrow \eta$ (where η is a set of names) is the naming function.

The element that have children and the children themselves turn into element node. The name of the nodes are defined by the $naming$ function. The edges connect the elements of the DTD. Such edges are described in definition 8.

Definition 8 (DTD edges) Let $ed(e_i \rightarrow e_j)$ be a directional edge connecting two table nodes. Then, $\forall e_i \in ec_j \exists ed(e_i \rightarrow e_j)$.

The DTD graph directional edges are also named containment edges, given they express containment in the same way as the relational schema graph directed edges.

Note in the figure 2 that the elements **email** and **title** appears twice in the leaf nodes. The only difference of such nodes lies in the hierarchy above them. It is important to preserve the hierarchical context of this kind of elements in the graph representation, because the elements meaning can be highly related to the element's parent [15]. Besides, each of the repeated leaf nodes can be mapped to a different database column.

We are currently not working with DTD with ID / IDREF properties. DTDs without ID/IDREF can be viewed as trees, which simplifies the algorithm development. As a matter of fact, like [14], a single DTD can be viewed as a forest of trees, where each root node corresponds to an element with no parent.

Even though the graph properties are simple, additional features such as optionality can be easily added to the graph builder algorithm. For DTDs an optional node would be an element assigned with the "?" or the "*" keywords. Nullable columns in the relational schema would also be converted to optional nodes. A matcher could use this cue to infer the matches between the graphs [8].

4. The Matching Process

In this section we present an algorithm that computes the similarity of every pair of leaf nodes from the two graphs. First, we introduce a series of definitions.

We use the term mapping node to refer to the leaf nodes the algorithm is trying to match. The term mapping node helps contextualize the role of this kind of node in the matching process. We refer to a mapping node by a full path name hereby uniquely identifying the node across the graph. In DTDs graphs, a mapping node is labelled by its XPATH expression. In a relational schema, a mapping node(database column) is labelled as the concatenation of the table name and the column name.

Definition 9 (*Mapping Pairs*) Let $DMN = \{e_1, \dots, e_n\}$ be the set of mapping nodes of DG . Identically, $RMN = \{c_1, \dots, c_n\}$ represents the set of mapping nodes of RG . Furthermore, $MP = DMN \times RMN$ is the set of mapping pair possibilities.

Note that the size of MP is a cross product of DMN and RMN . The number of mapping pairs is equal to a sum of all the correct and incorrect mappings(in the user perception). The correct and incorrect mappings deduced by the matcher are also commonly called true positive and false positive mappings [4], respectively.

Only a small subset of MP corresponds to the true positives. Our task is thus to rank the MP set so the true positives get in the top of the list. Each mapping pair is associated with a score ranging from 0 to 1 that determines how similar the two members of the pair are. We use this score to rank the MP .

Having defined the convention we shall used further on, we begin explaining how linguistic matchers can be applied to our matching problem in example 1:

Example 1 Consider the schemas in figure 2. The DTD mapping node `title</authors/book/title>` and the relational schema mapping node `title<book.title>` are representation of the same data. Since both mapping nodes are written equally, a linguistic matcher would be an appropriate technique for matching the pair.

However, similar names do not necessary mean that two nodes have the same meaning, as demonstrated in the next example:

Example 2 The database node `title<book.title>` represents the title of the book, while the DTD node `title</authors/book/chapter/title>` represents the title of one of the book's chapter. The use of a linguistic matcher over the mapping pair formed by these two nodes would very likely result a high score, whereas their meaning is not the same.

Therefore, the traditional linguistic comparison of the leaf nodes names is a limited solution, that can possibly lead the matching process astray by providing wrong assumptions about which mapping pairs are true positives. It is known that precision can be increased by using structure information [7]. Thus, to minimize the occurrence of false positives, we deem that the similarity of a mapping pair depends not only of the similarity of the mapping pair itself, but also on the similarities of the structure above the members of the mapping pair as well.

The algorithm we propose takes into account the structure information, dividing the processing in two phases. The first phase computes the linguistic similarity of two mapping nodes using some structure information. The second phase enhances the similarity calculated in the first phase by looking for resemblances in the structure of each mapping node. Each phase will be detailed in the next subsections.

4.1. Containment Edge Phase

In order to establish a method for comparing the names of two mapping nodes, we first attempted to estimate the linguistic relation between the DTD atomic elements and the relational schema columns.

In other words, we tried to determinate a DTD designer behavior, when choosing the elements names that correspond to columns of a relational database. There can be many linguistic relations between the mapping nodes. In this paper we identify three of them. The algorithm can be enriched with the addition of new forms of linguistic relations.

The most evident relation is that an element name is the same as its respective column name in the relational schema. Indeed, such situation occurs for the element **title** `</authors/book/title>` and the column **title**`<book.title>`, described in figure 2. Both element and column represent the same data and have the exact same name.

In some cases, an element name is highly similar to a table name. This situation can happen when modelling a DTD element that corresponds to the most significant column of a table. In such cases it is a common practice that the name of the element assumes the name of the table. We can see this kind of occurrence when modelling database column **name**`<author.name>` as the element **author**`</authors/author>`.

Lastly, the element's name could be a concatenation of the column name with the table name. The modelling of element **pubName**`</authors/book/info/pubName>` for database column **name**`<publisher.name>` is a case where the table's name is partially used to form the element name.

Based on this linguistic relations, we create three string pairs(p_1, p_2, p_3). The first member of the pair is always the atomic element name(e_i). The second member varies according to the outlined description below:

- $p_1(e_i, c_j)$ - where c_j is the column name.
- $p_2(e_i, c_j.parent)$ - where $c_j.parent$ is the table name.
- $p_3(e_i, c_j + c_j.parent)$ - where $c_j + c_j.parent$ is the concatenation of the table name and the column name

As we can see, the linguistic relations identified are highly related to the relational schema containment edges(column node to table node connections). In fact, we named this phase after such edges, due to its importance for the algorithm processing.

The algorithm perform its task by comparing one member of the pair with the other. Two linguistic matchers are used in the comparisons: *ngrams* and *substring* matcher. For the first two pairs, the *ngrams* matcher suffices. On the other hand, it fails to handle concatenated information.

We propose the *substring* linguistic matcher that presents better results with concatenated information. The *substring* matcher is useful for situation involving composite words. The algorithm assumes a high similarity if the greatest difference between two composite words is the order in which the words appears, such as the words *bookName* and *nameOfBook*.

Each of the pairs is compared using both *ngrams* and *substring*. The maximum value of all comparisons is used as the final linguistic similarity of the mapping pair, as shown in the function given next :

$$sim1 = \max (ngrams(p1), ngrams(p2), ngrams(p2), substring(p1), substring(p2), substring(p3))$$

We could also use additional linguistic matchers, or any other kind of lexical matchers to leverage the leaf node matching process furthermore. The impact of such extensions is one of our subjects of study for the future.

4.2. Interrelated Edges Phase

The second phase of the algorithm has the purpose of reinforcing previously calculated similarities. The reason why we decided to incorporate this mechanism in the matching is related to the limitations

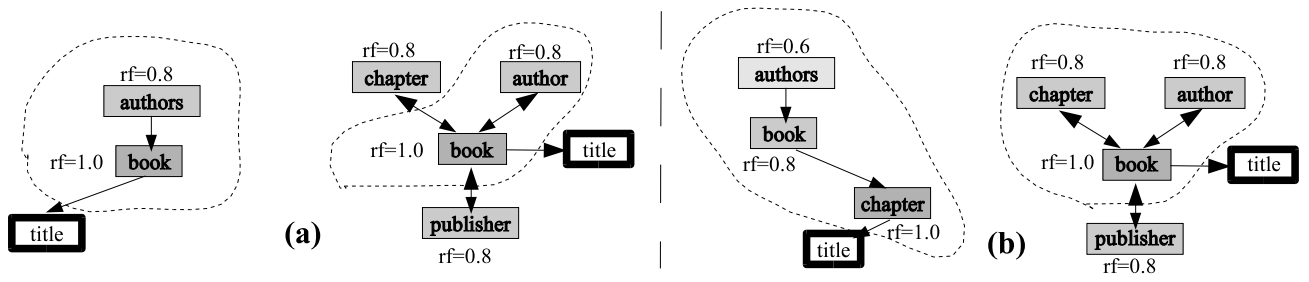


Figure 3: Mapping Pairs Example

of the technique described in subsection 4.1. Since this technique exploits poorly the structure of the schemas, some mapping pairs scores could be miscalculated, specially when the true semantic of a mapping node lies in the structure above it.

One situation where the structure could help in the matching process was already shown in example 1 and 2: The goal is to provide the best matching for database mapping node **title** $\langle book.title \rangle$. Based solely on the linguistic similarity of the mapping pairs, two matching candidates emerge with score equals to 1: **title** $\langle authors/book/title \rangle$ (fig. 3-a) and **title** $\langle authors/book/chapter/title \rangle$ (fig. 3-b). However, only the former represents the true positive.

It is than necessary to untie the scores of the two mapping pairs by providing a new similarity score for both of them. The new similarity value can be calculated by observing the related nodes of both members of the mapping pair in question. The related nodes (non-atomic nodes directly or indirectly connected to a mapping node) present information that are somehow related to the meaning of the mapping node itself. Each mapping node has a set of related nodes, as defined below:

Definition 10 (Related Nodes) Let $(er_{i1}, \dots, er_{in})$ be the set of the related nodes of e_i . Likewise, let $(cr_{i1}, \dots, cr_{in})$ be the set of the related nodes of c_i .

The occurrence of similarities in the related nodes increases the belief that the mapping nodes are indeed similar. Nevertheless, not every related node is actually relevant to the similarity computation. Parent nodes that are closer to the mapping node should have a greater relevance in the matching process. The notion of distance is directly proportional to the the number of edges separating the nodes.

In figure 3-a and b we demonstrate the related nodes for the two mapping pairs identified in Example 1 and 2. A gray scale is used to denote which nodes are more relevant. Darker gray means higher relevance.

Every related node has a relevance factor rf . The relevance factor ranges from 0 to 1, where 1 means the greatest possible relevance. The closest related node to the mapping node has the strongest $rf(1.0)$. The remaining related nodes relevance suffers a linear regression in the order of rfd as their distance to the mapping node increases. In figure 3 we aleatory define the rfd valuing 0.2. We can see in figure 3(b) how the relevance factor is low when the related node is far from the mapping node(see related node **authors** $\langle authors \rangle$). Note that to collect the majority of the related nodes of a relational schema mapping node one must navigate through interrelated edges(table to table connections). From this remark follows the algorithms name (*Interrelated Edges Algorithm*).

Related nodes with $rf = 0$ can be taken of the sets because they will not affect in any way the similarity calculus. Therefore, for larger values of rfd , the sets become smaller. Only a small piece of the graphs actually participate in the similarity calculus. On the other hand, for smaller values of rfd , the lists get bigger, which means that a wider spectrum of the graph nodes participate in the similarity calculus.

In order to compute the similarity of two mapping nodes(e_i, c_j) we will compute the similarity of all pairs of nodes that are related to e_i and c_j . This similarity value is computed taking into account the comparison of the nodes names and the distance from each node to its mapping element, as given by the relevance factor (rf).

The similarity between two mapping nodes $e_i \in E$ and $c_j \in (C - FK)$ is computed as follows:

Step 1: For each node er_i related to e_i and for each node cr_j related to c_j two similarity values are computed.

- $ling(er_i, cr_j)$ is the greater score of either the *ngrams* or the *substring* matcher. At this point only the names of the related nodes are compared.
- $relevance(er_i, cr_j)$ is the arithmetic average of the relevance factor of er_i and cr_j .

This two values are combined giving the score (local score) for the (er_i, cr_j) pair.

$$localScore = \frac{ling(er_i, cr_j) + relevance(er_i, cr_j)}{2} \quad (1)$$

Step 2: In this step the local scores of all related pairs to (e_i, c_j) that were computed in step 1 are combined into the resulting similarity value. This value is computed as follows:

$$sim2 = \frac{\sum_1^n localScore_i}{n} \quad (2)$$

Equation 2 is the average of the sum of all local scores whose value exceeds a given threshold. Local scores under a pre-defined threshold are discarded. Otherwise they could lower excessively the score of true positive mappings. Only local scores that contribute to the improvement of a mapping pair are considered useful. In figure 3 we enlance the related nodes actually used to calculate local scores. Each related node from one side forms a *rpair* with a related node from the other side. Notice that a local score depends not only of the linguistic similarity of the *rpair*, but also on the relevance of each member of the pair, which means that even a high linguistic similarity could produce a low local score, if the relevance of the related nodes is low.

4.3. Similarities Combination

The last step is to combine the similarity value computed in the Containment Edge Phase with the similarity value computed in the Interrelated Edge Phase. We use a weighted average to create a final score, as shown in equation 3:

$$finalScore = (sim1 * \alpha) + (sim2 * \beta) \quad (3)$$

where $\alpha + \beta = 1.0$

We choose to work with $\alpha = 0.75$ and $\beta = 0.25$. The differences in the weights can be comprehensively understood when analyzing the purpose of the algorithm phases. The Containment Edge algorithm represents a straightforward way to compute the similarity of leaf nodes, basically because it deals directly with the leaf nodes themselves. On the other hand, the Interrelated Edges algorithm is much more structure reliable, and its similarity value should be used with the simple purpose of highlighting the similarity of true positives in detriment of the false positives. This contrasting can be achieved by either reducing the similarity of wrong mapping pairs or by improving the similarity of the right mapping pairs.

For instance, it could happen that the reinforcement reduces the scores of all mapping pairs, including the right ones. However, in such cases, the false positives will be more penalized than the true positives, that in turn will remain in the top as the most similar mapping nodes.

DTD mapping nodes	R mapping nodes	score	
/movie/certificates/cert/country	movie.country	1.0	■
/movie/country	movie.country	1.0	□
/movie/cast/role/name	actor_name	1.0	■
/movie/cast/role/name	director.name	1.0	■
/movie/title	movie.title	1.0	□
/movie/language	movie.orig_language	0.864	□
/movie/year	movie.prod_year	0.761	□

(a)

DTD mapping nodes	R mapping nodes	score	
/movie/cast/role/actor	Actor_cod	1.0	⊗
/movie/cast/role/actor	Actor_name	1.0	□
/movie/director	director.cod	1.0	⊗
/movie/director	director.name	1.0	□
/movie/certificates/cert/country	movie.country	1.0	■
/movie/country	movie.country	1.0	□
/movie/cast/role/name	actor_name	1.0	■
/movie/cast/role/name	director.name	1.0	■
/movie/title	movie.title	1.0	□
/movie/language	movie.orig_language	0.864	□
/movie/year	movie.prod_year	0.761	□

(b)

DTD mapping nodes	R mapping nodes	score	
/movie/title	movie.title	0.886	□
/movie/country	movie.country	0.886	□
/movie/director	director.cod	0.834	⊗
/movie/director	director.name	0.834	□
/movie/language	movie.orig_language	0.785	□
/movie/cast/role/actor	Actor_cod	0.764	⊗
/movie/cast/role/actor	Actor_name	0.764	□
/movie/cast/role/name	Actor_name	0.764	■
/movie/cast/role/name	director.name	0.764	⊗
/movie/year	movie.prod_year	0.708	□

(c)

□	true positive mapping
■	false positive mapping
⊗	false positive mapping that can be eliminated with a data Type Matcher

Figure 5: Experimental Results

schema is a derivation of more than one element from another schema, forming the so called indirect mappings [16]. One of the consequences of this complex relation between the schemas is the need for 1:n mappings, where one element of DG maps to more than one element of RG . We intend to provide support for indirect mappings in future work.

In respect to the 1:1 mapping, we also eliminate mapping pairs if a member of the mapping pair is already part of another mapping pair whose score is greater. This guarantees only the top scores prevail. In the case of ties, none of the tied mapping pairs are eliminated.

The prevailing scores indicate the algorithm assumptions of the correct mapping pairs. Nevertheless, not every mapping pair presented in the lists are true positives (the true positives are listed in figure 4). To demonstrate the accuracy of each mapping result we use some symbols whose meaning is explained in figure 5.

One of such symbols(⊗) indicates the usage of a Data Type Matcher. The Data Type matcher can help disambiguate matchings by eliminating matching where the matched elements have different data Types.

The baseline (a) resulted in 4 true positives and 3 false positives. Considering the real number of true positives is up to 6, we have a precision of 57% and a recall of 66%. The precision could be improved if it was possible to untie the scores of the first two mapping pairs rows, involving the DTD mapping nodes */movie/certificates/cert/country* and */movie/country*. However, the scores remain the same even after using the data type matcher, since the mapping nodes have the same data type.

Running the Containment Edge matcher (b) we see that four new mapping pairs appear in the top of the list (two false positives and two true positives). The added true positives bring recall to 100%.

The identification of these true positives was possible because our technique compared the elements

names with tables names. However, our strategy presents a weakness in the sense it cannot identify which column of the table represents the correct match. Thus, all column of the matched table are returned as plausible candidates. It results in the added false negatives mentioned earlier, thereby lowering precision to 54%. By using the Data Type matcher it is possible though to increase precision to 66% (the mapping nodes of the \boxtimes lines have different data types, and are therefore eliminated).

The experiment (c) shows the results when the reinforcement is used to untie the scores encountered in the previous two lists. One particularly interesting remark is the elimination of the wrong mapping for **country** \langle *movie.country* \rangle . It was possible because the distance of DTD node **movie** \langle *movie* \rangle to the mapping node **country** \langle *movie/country* \rangle is lesser than its distance to the mapping node **country** \langle *movie/certificates/cert/country* \rangle .

Notice that the scores get lower for every mapping pair, if compared to experiment (b). Still, recall remains 100%). Additionally, precision is slightly increase to 60%. Finally, the precision can be improved to 85% by using the Data Type Matcher. The Data Type Matcher eliminates three wrong mapping pairs. Two of them are directly eliminated because of the use of different data types. The mapping pair involving relational schema node **name** \langle *director.name* \rangle was indirectly discarded because of our 1:1 filtering mechanism (another mapping pair has a better score for that node).

6. Conclusion and Future Work

The matching of DTDs and relational schemas is a challenging task, in that it involves schemas of different models whereas the most popular matchers known in the literature are dedicated to the matching of schemas that belong to the same model only.

Our commitment in this paper is to provide a solution for this sort of mappings. With that in mind, we constructed an algorithm specifically for the matching of DTDs and relational schemas. We explore the particular kinds of linguistic relations that exist in both schemas to find similarities where other generic solutions fails.

Even though many schemas can be successfully matched by our matcher, it is not yet a complete solution, once it does not handled every kind of arbitrariness found in the schemas. One example of limitation arises when the elements names from one schema are completely different from the elements names of the other schema. In such cases, it is necessary to leverage matching by adding different types of atomic-level matchers, and possibly trying to analyze instance data.

Yet, new structure matchers can enhance even further the pre-calculated similarities. One possible research path on this matter is to use the vicinity(sibling nodes) of the mapping nodes to infer matching instead of using just the related nodes of the mapping nodes, as we have proposed.

Another area of research is the study of how the tuning of the algorithm variables affect the matching. Another correlated work is to change the way the nodes relevance in the *Interrelated Edge Algorithm* are computed. Currently the relevance of the related nodes drops with a constant ratio, defined in the variable *rfd*, as their distance to the mapping node increases. A form of alternative calculus would be the usage of a quadratic function to determine the ratio of the relevance decrease.

Besides providing new linguistic relations between elements of the DTD and columns of the relational schema, we believe that the method behind the *Interrelated Edge Phase* is a generic contribution for the matching of any sort of schema format. After all, the essence of the approach is to compares two lists of nodes, where each list member is associated with a relevance factor. A list of nodes along with relevance factor for each list members can be easily created independently of the schema format.

Yet incomplete, we believe our efforts in this work bring some light in the matching of XML schemas and relational schemas. Taken this work as a basis, we can exploit different aspects of the schemas, such

as the existence of indirect mappings.

References

- [1] http://www.hitsw.com/products_services/xmlplatform.html.
- [2] Alon Halevy Anhai Doan, Pedro Domingos. *Machine Learning*, chapter Learning to Match the Schemas of Data Sources: A Multistrategy Approach, pages 279 – 301.
- [3] R. Baezza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, Menlo Park, California, 1999.
- [4] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. *Comparison of schema matching evaluations*. 2002.
- [5] Hong-Hai Do and Erhard Rahm. Coma: A system for flexible combination of schema matching approaches. In *Proceedings of the 28th Conf. on VLDB*, 2002.
- [6] Prasanna Ganesan, Hector Garcia-Molina, and Jennifer Widom. *ACM Transactions on Information Systems (TOIS)*, chapter Exploiting hierarchical domain structure to compute similarity, pages 64 – 93.
- [7] N. Guarino, C. Masolo, and G. Vetere. Ontoseek: Contentbased access to the web. In *IEEE Intelligent Systems*, volume 14.
- [8] J.A. Larson, S.B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. In *IEEE Transactions on Software Engineering*, pages 15(4):449–463. IEEE Computer Society Press, April 1989.
- [9] Jayant Madhavan, Philip A. Bernstein, Kuang Chen, Alon Halevy, , and Pradeep Shenoy. Corpus-based schema matching. In *Workshop on Information Integration on the Web at the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.
- [10] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of VLDB*, 2001.
- [11] S. Mehiik, H. Garcia-Molinaa, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of ICDE*, 2002.
- [12] L. Popa, M. A. Hernandez, Y. Velegrakis, R. J. Miller, F. Naumann, and H. Ho. Mapping xml and relational schemas with clio, demo. In *In ICDE*, 2002.
- [13] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. In *VLDB Journal 10*, 2001.
- [14] C. Reynaud, J. P. Sirot, and D. Vodislav. Semantic integration of xml heterogeneous data sources. In *Proceedings of the 2001 International Database Engineering & Applications Symposium*, July 2001.
- [15] G. Wang, J. Goguen, Y. Nam, and K. Lin. Interactive schema matching with semantic functions. In *Semantic Integration Workshop*, October 2003.
- [16] Li Xu and David W. Embley. Discovering direct and indirect matches for schema elements. In *Eighth International Conference on Database Systems for Advanced Applications (DASFAA)*, Kyoto, Japan, March, 26-28 2003.