

Matching XML DTD To Relational Database Views

M. Pagotto and A. Celentano

Dipartimento di Informatica, Università Ca' Foscari di Venezia
{mpagotto,auce}@dsi.unive.it

Abstract

In this paper we approach the problem of matching a set of XML Document Type Definitions against a view schema drawn from a relational database, aiming at evaluating the similarity between the view schema and the DTDs for (semi)automatic translation of relational tables (i.e., views, queries) data into XML documents. The paper discusses a matching algorithm and evaluates the similarity criteria defined.

The approach is intended to be used in an interactive system supporting a user in populating documents belonging to a set of predefined schemas with data extracted from a database. The application we foresee are mostly related to the areas of prototyping and reuse, and are based on a scenario in which several libraries of DTDs describe common documents or data structures defined by standards related to specific application domains, and databases defined for various purposes (e.g., legacy repositories, application programs, etc.) store data that need to be processed according to the schemas defined. In such a scenario a system for evaluating the degree of correspondence between database data and document schemas can help to develop prototypes at low cost with great flexibility.

1 Introduction

XML [7, 8] as a language for data exchange between several data sources is the baseline of several proposed models and ongoing projects. Its simple yet flexible structure, its extensibility and the possibility of superimposing on it a typing mechanism make XML suitable not only as a language for Web-based data exchange, but also as an intermediate language for applications involving data exchange at various level (see for example [1, 2, 4, 6]).

In this paper we approach the problem of matching a set of XML Document Type Definitions against a view schema drawn from a relational database, aiming at evaluating the similarity between the view schema and a DTD for (semi)automatic translation of relational tables (views, queries) into XML documents.

The approach is intended to be used in an interactive system supporting a user in “translating” relational queries/views into XML documents, e.g. for populating documents belonging to a set of predefined schemas with data extracted from a database. We note that, differently from other matching problems known in the literature, it is of primary concern to preserve the “meaning” of the match in terms of the application domain, against the completeness of the match itself. In practice we expect that, unless relational views and DTDs come from a coordinated design, we can obtain only partial matches between the two structures. Some data belonging to the view will

not be considered by the XML document schema, and the XML schema could require (or accept as an option) data which are not part of the relational view.

The application we foresee are mostly related to the areas of prototyping and reuse, and are based on a scenario in which several libraries of DTDs describe common documents or data structures defined by standard related to specific application domains. Databases defined for various purposes (e.g., legacy repositories, application programs, etc.) store data that need to be processed according to the schemas defined. In such a scenario a system for evaluating the degree of correspondence between database data and document schemas can help to develop prototypes at low cost with great flexibility. Further details can be found in [5].

1.1 The problem

We define in a more precise way how we approach the problem. Given a relational database D with schema S_D and a query Q whose result is the relation R_Q with schema S_Q , R_Q can be mapped to an XML document D_Q with a corresponding Document Type Definition DTD_Q . Given a set (a library) of XML DTDs DTD_i , we define a similarity measure between each DTD_i and DTD_Q aiming at finding the DTD that best models the schema of the relation defined by the query, based on structural coverage of DTD_Q elements by the elements of the DTD_i . We define then a mapping between the data of relation R_Q and the elements of an XML document satisfying definition DTD_i , in order to construct a document mirroring as close as possible the query result.

We base our approach on the tree-structured model of XML documents and DTDs described in [3]. An XML document is modeled as a labeled ordered tree, called a *loto* (standing for *labeled ordered tree object*). Nodes correspond to XML elements and their labels provide the type names of the elements. The children of a node are totally ordered. In figure 1 is illustrated an example of an XML document and the corresponding *loto*.

A DTD is modeled as a *loto type definition (ltd)*, that associates to each type name a language on the alphabet of type names. Figure 2 shows the DTD and the corresponding *ltd* for the XML document of Figure 1.

2 Mapping a relational view to a *ltd*

In the scope of this paper a view over a relational database is defined by a query yielding a relation. Let us consider a relational database D and a SQL query Q . The execution of Q yields a relation R_Q . We represent the schema of R_Q as a *ltd* in order to map it onto a DTD for a class of XML documents.

Given a SQL query, we consider only the *select-from* clauses for the goals of our discussion (since other clauses do not change the schema), for example:

```
SELECT  $t.a_1, \dots, t.a_n, r.b_1, \dots, r.b_m$ 
FROM  $T\ t, R\ r$ 
```

where $a_1, \dots, a_n$ are attributes of relation T and $b_1, \dots, b_m$ are attributes of relation R . The result of the query is a relation with the following schema:

$a_1, \dots, a_n, b_1, \dots, b_m$.

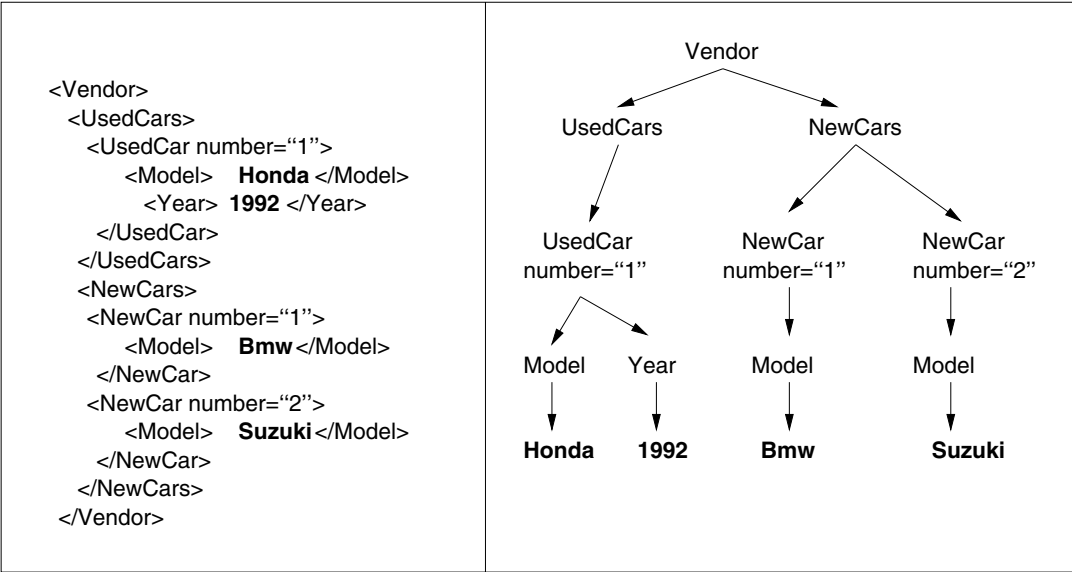


Fig. 1. An XML document and the corresponding *loto*.

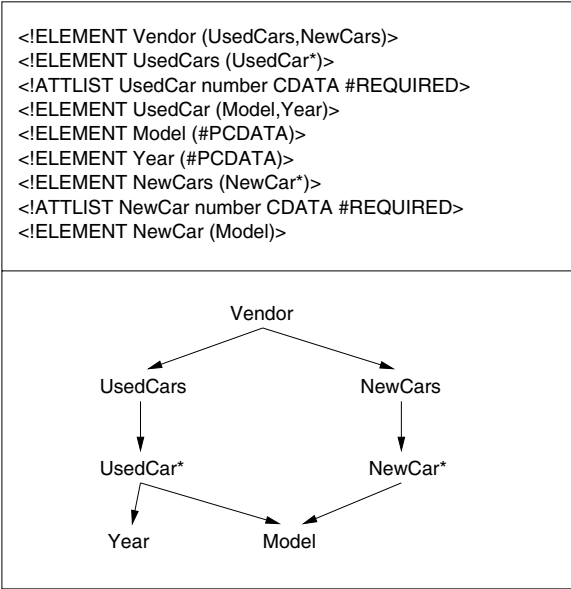


Fig. 2. A DTD and the corresponding *ltd* for the document in Figure 1.

We represent this schema as a *ltd*, hence mapping it to a DTD. As Figure 3 shows, *viewName* is the name of the schema, *viewElement* is the list of the labels assigned to the relations defined in the FROM clause and used in the SELECT clause. Each element of *viewElement* is a DTD tag which collects the view attributes coming from one of the relations specified in the query. In this way the DTD retains in its structure some information about the relations from which the view is built. Each component of *viewElement* models the tuple type. From the *ltd* a DTD source can be built by visiting it in descending order and by keeping track of intermediate nodes. By instantiating the *ltd* with the actual query results, a specific *loto* is built, whose visit in pre-order gives the XML source satisfying the DTD.

The tag *description* in Figure 3 plays a reference role for describing the *ltd* in terms of its application meaning. As stated in the introduction, we assume that the approach here described is applied to the selection of suitable DTD from a library for XML document compilation. Hence some information about the DTD themselves must be taken into account in order to manage such a library. The tag *description*, without entering into details, is a placeholder for such information.

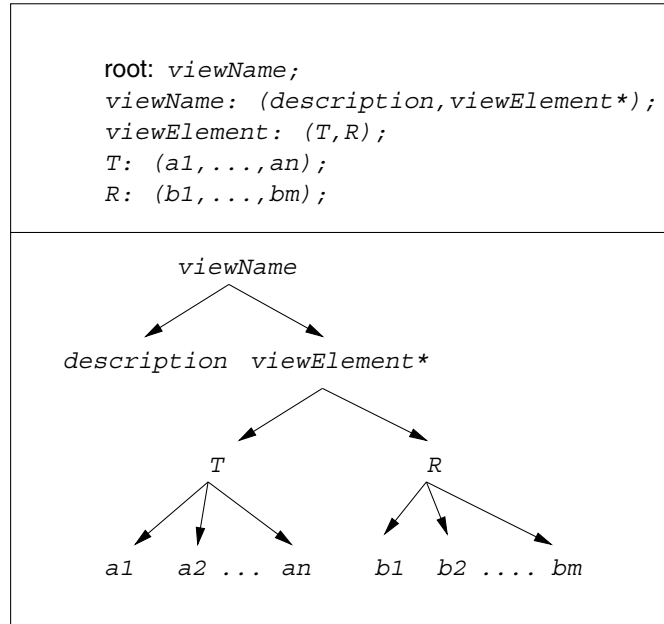


Fig. 3. An example of *loto* type definition modeling a relational view.

3 Matching relational view schemas to DTDs

The problem of matching a relational view schema to a set of DTDs in order to find the most similar is therefore translated to the problem of building *ltd* from relational views and DTDs, and comparing them. Given a *ltd* λ_0 and a set of *ltd* $\Lambda = \{\lambda_1, \dots, \lambda_n\}$, compare λ_0 to each of λ_i by computing a similarity measure $\sigma_{0,i}$. The similarity is defined not only by a numeric value, but also by the list of the corresponding nodes in the two *ltd*.

We must consider that two *ltd* are similar to the degree that they represent equivalent information, both from the structural and conceptual viewpoints. From the structural viewpoint,

a correspondence must be established between nodes and subtrees of both *ltd*, covering as much as possible of the two structures. From the conceptual viewpoint, types and labels in the nodes must correspond (at some extent). For types we rely on compatibility, while for labels we rely on synonymous definitions among names.

In a *ltd* of a DTD the nodes model tags and data are included in the tags modeled by leaf nodes. From a structural viewpoint, the tag modeled by node *t* includes the tags modeled by nodes of the subtree of *t*.

In the *ltd* modeling a (DTD for a) relational table the leaves model tags corresponding to attributes of the view resulting from the query execution. The leaves that model attributes belonging to the same table are children of a same node that models the tag corresponding to the whole table, which is of type *tuple*.

This means that in the XML document having such a DTD as a document schema and containing data from the table, there are zero or more objects corresponding to the tag, each needing an identifier, that we'll call *number*, distinguishing the several tuple instances according to some (in principle arbitrary) order.

4 DTD similarity evaluation

For every node of a *ltd*, we define an object *infoNodeForMatching* with the following components:

label = the label of the node, which is also the label of the tag modeled by the node;

type = the “structural” type of the node, i.e., *tuple* if the node models a tag corresponding to a database relation, or the empty string if the node models another type of tag.

For every set of leaves children of the same node we define an object *objectForMatching* (*ofm*) with the following components:

subFrontier = an array with the information (*infoNodeForMatching*) about leaf nodes, children of the same father node; array elements are ordered as leaves are;

ancestors = an array with the information (*infoNodeForMatching*) about ancestor nodes of the set of leaves considered; the first element of the array refers to the father node, the second element refers to the grandfather node, and so on. The last element refers to the root node of *ltd*.

For a *ltd* we define an object *setOfObjectsForMatching* (*sofm*) with the following component:

collection = an array of *ofm* where the first element is the leftmost *ofm* in the *ltd* and the last element of vector is the rightmost one.

We want to evaluate the degree of similarity between two DTD represented by their *ltd*, *ltd*₁ for *DTD*₁ and *ltd*₂ for *DTD*₂. Let *sofm*₁ and *sofm*₂ be the *setOfObjectsForMatching* of the two *ltds*.

We begin with computing the degree of similarity between *ofm*_{*i*} element in *sofm*₁ and *ofm*_{*j*} element in *sofm*₂. Let $\alpha_{i,j}$ be the result of the computation.

$\omega_{i,j} = 0;$
for (*r* = 1; *r* <= | *ofm*_{*i*}.*subFrontier* |; *r*++) {

```

 $\vartheta_r = 0;$ 
for ( $k = 1; k \leq |ofm_j.subFrontier|; k++$ ) {
  if ( $ofm_j.subFrontier[k].label$  synonymous of
     $ofm_i.subFrontier[r].label$ ) {
    if ( $k == r$ )
       $\beta_{r,k} = \zeta_{=} \times \xi;$ 
    else
       $\beta_{r,k} = \frac{\zeta_{\neq} \times \xi}{|k-r|};$ 
    if ( $ofm_j.subFrontier[k].type == ofm_i.subFrontier[r].type$ )
       $\beta_{r,k} = \beta_{r,k} \times \varepsilon_{=};$ 
    else
       $\beta_{r,k} = \beta_{r,k} \times \varepsilon_{\neq};$ 
  }
  else  $\beta_{r,k} = 0;$ 
  if ( $\beta_{r,k} \geq \vartheta_r$ )  $\vartheta_r = \beta_{r,k};$ 
}
 $\omega_{i,j} = \omega_{i,j} + \vartheta_r;$ 
}

```

```

 $\rho_{i,j} = 0;$ 
if ( $\omega_{i,j} \neq 0$ ) {
  for ( $r = 1; r \leq |ofm_i.ancestors|; r++$ ) {
     $\tau_r = 0;$ 
    for ( $k = 1; k \leq |ofm_j.ancestors|; k++$ ) {
      if ( $ofm_j.ancestors[k].label$  synonymous of
         $ofm_i.ancestors[r].label$ ) {
        if ( $k == r$ )
           $\mu_{r,k} = \eta_{=} \times \xi;$ 
        else
           $\mu_{r,k} = \frac{\eta_{\neq} \times \xi}{|k-r|};$ 
        if ( $ofm_j.ancestors[k].type == ofm_i.ancestors[r].type$ )
           $\mu_{r,k} = \mu_{r,k} \times \varepsilon_{=};$ 
        else
           $\mu_{r,k} = \mu_{r,k} \times \varepsilon_{\neq};$ 
      }
      else  $\mu_{r,k} = 0;$ 
      if ( $\mu_{r,k} \geq \tau_r$ )  $\tau_r = \mu_{r,k};$ 
    }
     $\rho_{i,j} = \rho_{i,j} + \tau_r;$ 
  }
}

```

$\alpha_{i,j} = \omega_{i,j} + \rho_{i,j};$

where:

- $\omega_{i,j}$ is an esteem of the degree of similarity between the subfrontier of ofm_i and the subfrontier of ofm_j ;
- $\beta_{r,k}$ is an esteem of the degree of similarity between the r -th node of subfrontier of ofm_i and the k -th node of subfrontier of ofm_j ;
- ϑ_r is an esteem of the degree of similarity between the r -th node of subfrontier of ofm_i and the node of subfrontier of ofm_j that is most similar;
- ζ_+ is a real coefficient that represent the weight of the hypothesis that two synonymous nodes are in same position in the two subfrontier considered;
- ζ_- is a real coefficient that represent the weight of the hypothesis that two synonymous nodes are *not* in same position in the two subfrontier considered;
- $\rho_{i,j}$ is an esteem of the degree of similarity between the ancestors of ofm_i and the ancestors of ofm_j ;
- ξ is a real coefficient that shows the degree of synonymy between the labels of the nodes considered;
- $|k-r|$ shows the distance between the r -th node of the subfrontier of ofm_i and the k -th node of the subfrontier of ofm_j during the subfrontier analysis; whereas shows the distance between the r -th node of the subfrontier of ofm_i and the k -th node of the subfrontier of ofm_j during the ancestors analysis;
- $\mu_{r,k}$ is an esteem of the degree of similarity between the r -th node of ancestors of ofm_i and the k -th node of ancestors of ofm_j ;
- τ_r is an esteem of the degree of similarity between the r -th node of ancestors of ofm_i and the node of ancestors of ofm_j that is most similar;
- η_+ is a real coefficient that represents the weight of the hypothesis that two synonymous nodes are in same position in the two ancestors considered;
- η_- is a real coefficient that represent the weight of the hypothesis that two synonymous nodes are *not* in the same position in the two ancestors considered;
- ε_+ is a real coefficient that represent the weight of the hypothesis that two synonymous nodes are of the same type;
- ε_- is a real coefficient that represent the weight of the hypothesis that two synonymous nodes are *not* of the same type;
- $\alpha_{i,j}$ is an esteem of the degree of similarity between the ofm_i of $sofm_1$ and the ofm_j of $sofm_2$.

In practice, the idea is to consider each node r of the subfrontier of the ofm_i in $sofm_1$ of the ltd_1 , and to find the node of the subfrontier of the ofm_j in $sofm_2$ of the ltd_2 that is most similar, i.e., the node that maximizes ϑ_r . Then ϑ_r is added to $\omega_{i,j}$.

If $\omega_{i,j}$ is not zero, there is some similarity between the subfrontiers considered, therefore it is possible to compute $\rho_{i,j}$, the degree of similarity between the ancestors.

Then, for each node r of the ancestors of the ofm_i in $sofm_1$ of ltd_1 , find the node of the ancestors of ofm_j in $sofm_2$ of ltd_2 that is most similar, i.e., the node that maximizes τ_r . Then τ_r is added to $\rho_{i,j}$. The complete similarity coefficient $\alpha_{i,j}$ is the sum between $\omega_{i,j}$ and $\rho_{i,j}$.

The computational complexity of $\alpha_{i,j}$ is

$$O((| \text{opm}_i.\text{subFrontier} | \times | \text{opm}_j.\text{subFrontier} |) + (| \text{opm}_i.\text{ancestors} | \times | \text{opm}_j.\text{ancestors} |))$$

The coefficients ξ , ζ , $\zeta_{\neq}$, η , $\eta_{\neq}$, ε , $\varepsilon_{\neq}$ must be defined suitably according to the application domain.

As a last step we consider the computation of the degree of similarity between sofm_1 of ltd_1 and sofm_2 of ltd_2 . Here, the idea is to consider each collection of the ofm_r of the sofm_1 of ltd_1 , and to find the collection of ofm of the iofm_2 of ltd_2 that is most similar.

Let $\sigma_{1,2}$ be the result of calculation that represent the degree of similarity between ltd_1 and ltd_2 .

```

 $\sigma_{1,2} = 0;$ 
for (r = 1; r <= |  $\text{sofm}_1.\text{collection}$  |; r++) {
   $\lambda_r = 0;$ 
  for (k = 1; k <= |  $\text{sofm}_2.\text{collection}$  |; k++) {
    compute  $\alpha_{r,k};$ 
    if ( $\alpha_{r,k} > \lambda_r$ ) {
       $\lambda_r = \alpha_{r,k};$ 
    }
  }
}
 $\sigma_{1,2} = \sigma_{1,2} + \lambda_r;$ 
}

```

where:

- λ_r is an esteem of the degree similarity between the *collection* of the ofm_r of sofm_1 and the *collection* of the ofm of sofm_2 that it's most similar;
- $\sigma_{1,2}$ is the esteem of the similarity between the ltd_1 and ltd_2 , whereas the similarity between the dtd_1 and dtd_2 .

The computational complexity of $\sigma_{k,q}$ is:

$$O(| \text{sofm}_k.\text{collection} | \times | \text{sofm}_q.\text{collection} | \times (\text{cost of computation of } \alpha_{i,j}))$$

By comparing the *ltd* of a relational view with the members of a library of *ltd* of Document Type Definitions, DTDs can be ranked according to their structural similarity with the relational view.

5 An example

We consider the *ltd* γ , a set Γ of *ltd*, $\Gamma = \{x, y, w\}$ and a synonymy table X (Table 1) with the assumption that name identity has synonymy coefficient 1. We compute the degree of similarity between γ and each *ltd* in Γ , using the algorithm presented in previous section, using the values in Table 2 for the coefficients ζ , $\zeta_{\neq}$, η , $\eta_{\neq}$, ε , $\varepsilon_{\neq}$.

For each *ltd* we describe the entities which we'll consider in DTD matching. Figures 4–7 show the *ltds* which contain the following objects:

ltd γ :

$$\text{sofm}_\gamma.\text{collection} = \{ \text{ofm}_{\gamma,1}, \text{ofm}_{\gamma,2} \}$$

<i>Label</i>	<i>Synonymous</i>	<i>coeff. of synonymy (ξ)</i>
<i>Condominium</i>	<i>Building</i>	0.65
<i>Condominiums</i>	<i>Buildings</i>	0.9
<i>Name</i>	<i>CondoName</i>	1.0
<i>FullAddress</i>	<i>Address</i>	1.0
<i>NofApartments</i>	<i>NofAps</i>	1.0
<i>NofApartments</i>	<i>NumberOfApartments</i>	1.0
<i>AccountNo</i>	<i>AccNo</i>	1.0

Table 1. A synonymy table.

<i>coeff.</i>	<i>value</i>
$\zeta_{=}$	2.0
$\zeta_{\neq}$	1.5
$\eta_{=}$	1.5
$\eta_{\neq}$	1.0
$\varepsilon_{=}$	1.5
$\varepsilon_{\neq}$	0.5

Table 2. The values for coefficients $\zeta_{=}$, $\zeta_{\neq}$, $\eta_{=}$, $\eta_{\neq}$, $\varepsilon_{=}$ and $\varepsilon_{\neq}$.

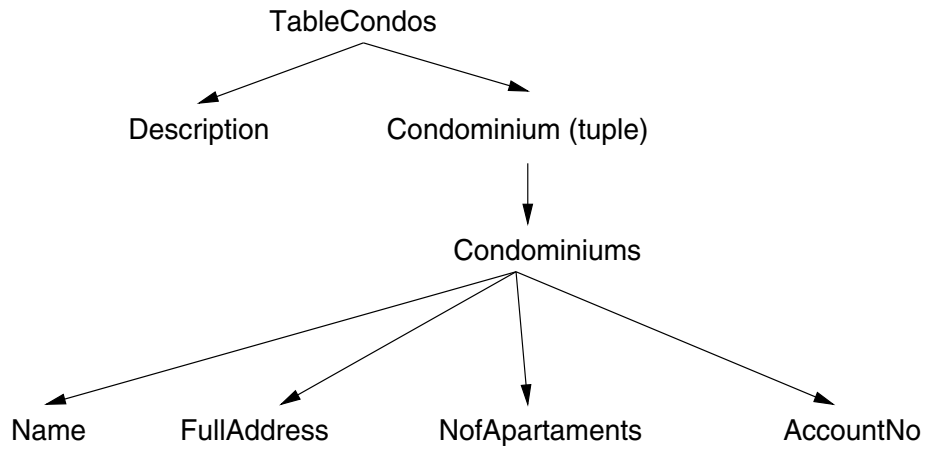


Fig. 4. The *ltd* γ .

$ofm_{\gamma,1}.subFrontier = \{ (Description,) \}$
 $ofm_{\gamma,1}.ancestors = \{ (TableCondos,) \}$
 $ofm_{\gamma,2}.subFrontier = \{ (Name,), (FullAddress,),$
 $(NofApartments,), (AccountNo,) \}$
 $ofm_{\gamma,2}.ancestors = \{ (Condominiums,), (Condominium,tuple),$
 $(TableCondos,) \}$

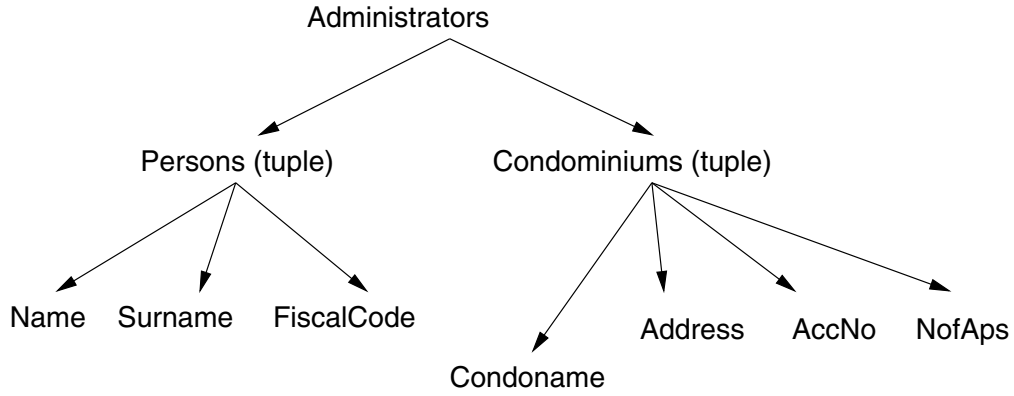


Fig. 5. The *ltd x*.

ltd x:

$sofm_x.collection = \{ ofm_{x,1}, ofm_{x,2} \}$
 $ofm_{x,1}.subFrontier = \{ (Name,), (Surname,),$
 $(FiscalCode,) \}$
 $ofm_{x,1}.ancestors = \{ (Persons,tuple),$
 $(Administrators,) \}$
 $ofm_{x,2}.subFrontier = \{ (CondoName,), (Address,),$
 $(AccNo,), (NofAps,) \}$
 $ofm_{x,2}.ancestors = \{ (Condominiums,Tupla),$
 $(Administrators,) \}$

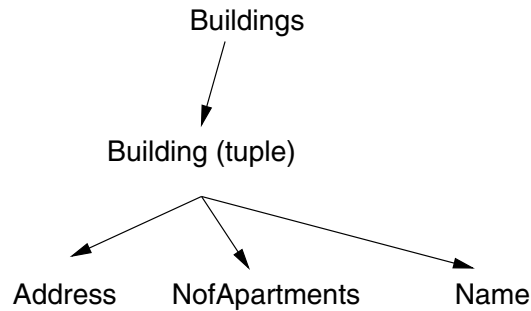


Fig. 6. The *ltd y*.

ltd y:

$sofm_y.collection = \{ opm_{y,1} \}$
 $ofm_{y,1}.subFrontier = \{ (Address,), (NumberOfApartments,), (Name,) \}$
 $ofm_{y,1}.ancestors = \{ (Building,tuple), (Buildings,) \}$

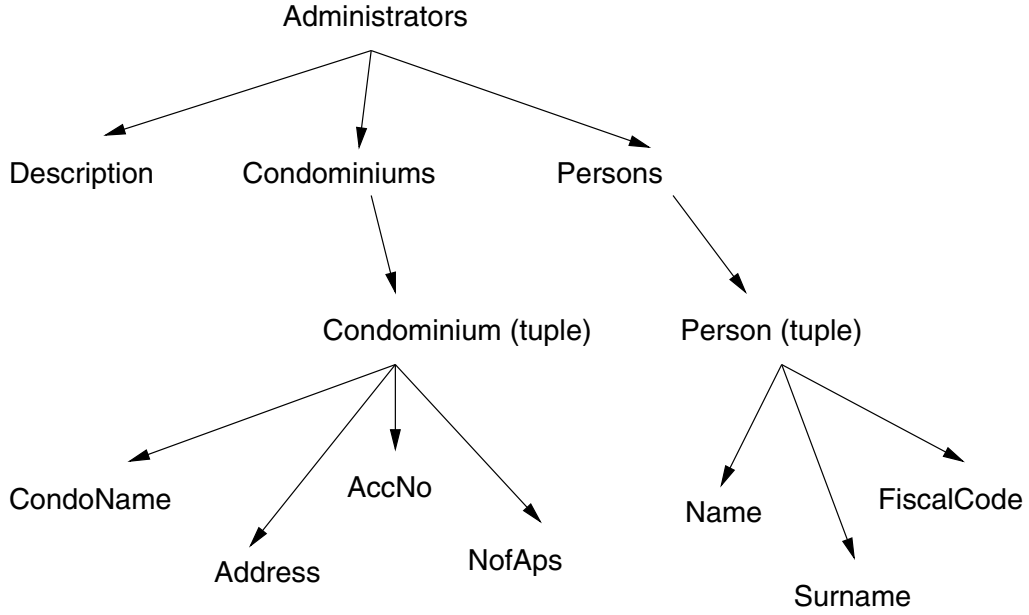


Fig. 7. The *ltd w*.

ltd w:

$sofm_w.collection = \{ ofm_{w,1}, ofm_{w,2}, ofm_{w,3} \}$
 $ofm_{w,1}.subFrontier = \{ (Description,) \}$
 $ofm_{w,1}.ancestors = \{ (Administrators,) \}$
 $ofm_{w,2}.subFrontier = \{ (CondoName,), (Address,), (AccNo,), (NofAps,) \}$
 $ofm_{w,2}.ancestors = \{ (Condominium,tuple), (Condominiums,), (Administrators,) \}$
 $ofm_{w,3}.subFrontier = \{ (Name,), (Surname,), (FiscalCode,) \}$
 $ofm_{w,3}.ancestors = \{ (Persona,tuple), (Persons,), (Administrators,) \}$

By computing the similarity between the *ltd* γ and the other *ltds* we obtain the values listed in Table 3. As an example we detail the computation of $\sigma_{\gamma,x}$, the similarity between the *ltds* γ and x

$$\omega_{1,1} = 0.0$$

$$\rho_{1,1} = 0.0$$

$$\alpha_{1,1} = \omega_{1,1} + \rho_{1,1} = 0.0$$

$$\begin{aligned}
\omega_{1,2} &= 0.0 \\
\rho_{1,2} &= 0.0 \\
\alpha_{1,2} &= \omega_{1,2} + \rho_{1,2} = 0.0 \\
\omega_{2,1} &= \zeta_{=} \times (\text{synon. between Name and Name}) \times \varepsilon_{=} \\
&= (2.0 \times 1.0 \times 1.5) = 3.0 \\
\rho_{2,1} &= 0.0 \\
\alpha_{2,1} &= \omega_{2,1} + \rho_{2,1} = 3.0 \\
\omega_{2,2} &= \zeta_{=} \times (\text{synon. between Name and CondoName}) \times \varepsilon_{=} + \\
&\quad \zeta_{=} \times (\text{synon. between FullAddress and Address}) \times \varepsilon_{=} + \\
&\quad \frac{\zeta_{\neq} \times (\text{synon. between NofApartments and NofAps})}{|\text{pos NofApartments} - \text{pos NofAps}|} \times \varepsilon_{=} + \\
&\quad \frac{\zeta_{\neq} \times (\text{synon. between AccountNo and AccNo})}{|\text{pos AccountNo} - \text{pos AccNo}|} \times \varepsilon_{=} \\
&= (2.0 \times 1.0 \times 1.5) + (2.0 \times 1.0 \times 1.5) + \left(\frac{1.5 \times 1.0}{|3-4|} \times 1.5\right) + \left(\frac{1.5 \times 1.0}{|4-3|} \times 1.5\right) \\
&= 10.5 \\
\rho_{2,2} &= \eta_{=} \times (\text{synon. between Condominiums and Condominiums}) \times \varepsilon_{\neq} \\
&= (1.5 \times 1.0 \times 0.5) = 0.75 \\
\alpha_{2,2} &= \omega_{2,2} + \rho_{2,2} = 11.25 \\
\sigma_{\gamma,p} &= \max(\alpha_{1,1}, \alpha_{1,2}) + \\
&\quad \max(\alpha_{2,1}, \alpha_{2,2}) \\
&= 11.25
\end{aligned}$$

<i>ltd</i>	σ_{γ}
<i>w</i>	16.50
<i>x</i>	11.25
<i>y</i>	7.95

Table 3. The similarity evaluation results.

6 Conclusion

In order to test and evaluate the matching algorithm we have built a prototype in Java able to manage the whole process of XML documents production from relation database queries. Figure 8 shows one of the windows of the prototype, showing the results of the match. The prototype allows a user to:

- define and execute SQL queries and build the *ltd* which represents the schema of the result;
- build a library of XML DTDs, both generated by the prototype itself or defined elsewhere, and edit the DTDs;
- build and maintain a thesaurus storing synonymous names;
- edit the similarity coefficients $\zeta_{=}$, $\zeta_{\neq}$, $\eta_{=}$, $\eta_{\neq}$, $\varepsilon_{=}$ and $\varepsilon_{\neq}$;

- compute similarity between the *ltd* corresponding the SQL query in input and the *ltd* selected from a library;
- select one specific DTD and build an XML document with the data contained in the SQL query result.

Early experiments show that the ranking proposed by the matching algorithm is plausible as long as DTDs do not differ seriously from the query schema. In more detail, we note the following issues:

- Since a view on a relational database contains a collection of data, the concept of *tuple* is important for modeling the resulting XML document. Each element of the tuple must be identified by an attribute value, even if this is not strictly required by XML documents. However, if the XML representation is viewed as an intermediate representation rather than a final document, it is plausible to be able distinguish the tuple elements in an unambiguous way.
- The use of a thesaurus should go beyond the simple schema devised in this algorithm. We have implicitly assumed that column names in a relational table are self-explaining, therefore meaningful in terms of the application domain, which could not be true (mainly for existing legacy databases). A more sophisticated use of meta-information for describing the real meaning of the data components has to be investigated.
- The structural similarity between the *ltds* is based on the reciprocal position of nodes and leaves both among the ancestors and among the sibling nodes. We have tried to model the exigency that a rich structure, i.e., organized along different aggregation levels, should be preserved in XML translation, and conversely a simple structure should not be artificially enriched.

Further work could be done in several directions, but mainly in improving the user interaction at all stages of the process. Among the issues we shall approach as mainly meaningful the integration of several query results into one or more XML documents, with a twofold goal: to be able to transform data related to more complex application domains, and provide an effective way to automatically build (XML) documents from data repositories. For the latter issue, in several application fields programs have been designed for automatic generation of documents, even with sophisticated capabilities. We mention, as a meaningful example, juridical documents like notary acts, certificates, contracts, modeled along formalized and verifiable patterns. The use of XML could improve their usability non only as a final product but also as an information transportation model.

References

1. Bert Bos. *XML representation of a relational database*. W3C, <http://www.w3c.org/XML/RDB.html>, 1997.
2. Alin Deutsch, Mary Fernandez, and Dan Suciu. *Storing Semistructured Data in Relations*. <http://www.research.att.com/~suciu/workshop99-announcement.html>, 1999.
3. Bertram Ludäscher, Yannis Papakonstantinou, Pavel Velikhov, and Victor Vianu. *View Definition and DTD Inference for XML*. <http://www.sdsc.edu/~ludaesch/Paper/icdt-ws99.html>, 1999.
4. Project MIX. *The MIX (Mediation of Information using XML) Home Page*. <http://www.db.ucsd.edu/Projects/MIX/>.

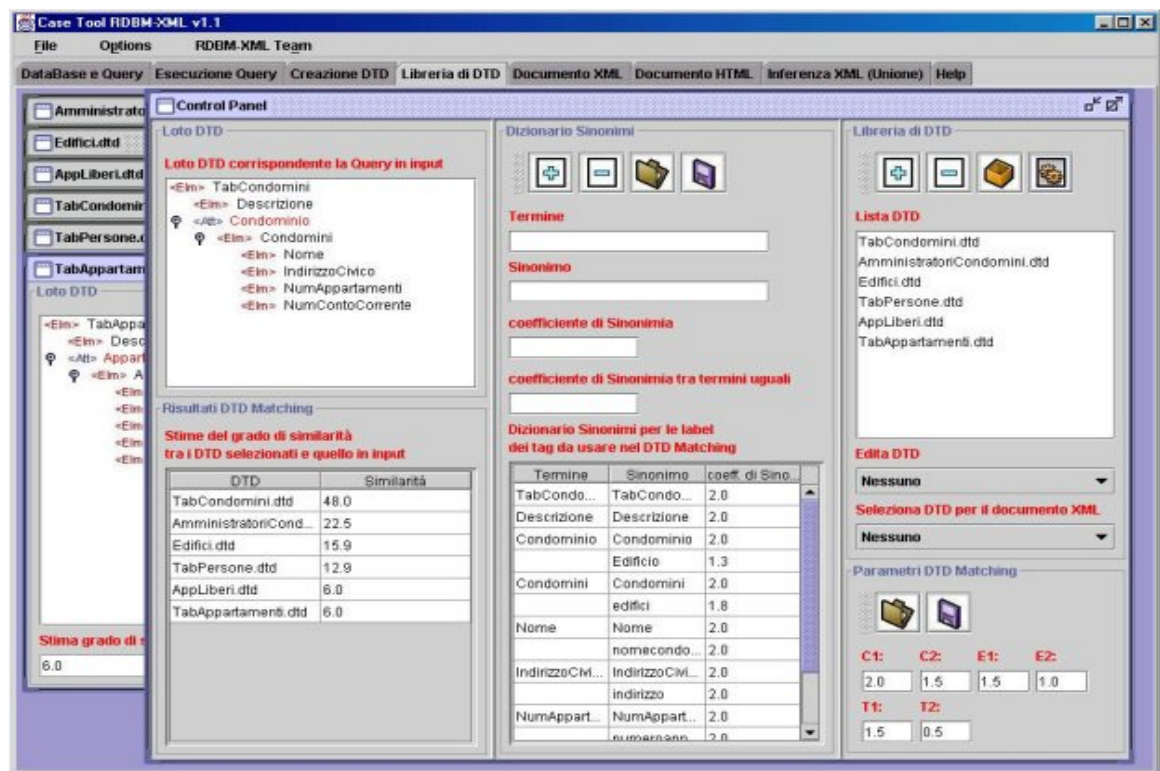


Fig. 8. A prototype screen showing matching results.

5. M. Pagotto. Un sistema per il matching tra viste relazionali e documenti XML. Degree dissertation, Universit Ca' Foscari di Venezia, March 24, 2000 (in preparation).
6. W3C. *QL'98 - The Query Languages Workshop*. <http://www.w3.org/TandS/QL/QL98/>.
7. W3C. *The World Wide Web Consortium*. <http://www.w3.org/>.
8. W3C. *Extensible Markup Language (XML) 1.0*, 10 February 1998. <http://www.w3.org/TR/1998/REC-xml-19980210.html>.