Exploiting related digital library corpora with query rewriting (Extended Abstract) *

Federica Mandreoli and Riccardo Martoglia

Università di Modena e Reggio Emilia,
Dip. di Ingegneria dell'Informazione, Modena, Italy
{mandreoli.federica, martoglia.riccardo}@unimo.it

Abstract. In this paper, we present the preliminary results of the ongoing research activity we are carrying out in the context of approximate XML query answering when the schemas of the XML documents are available. The method we propose involves a preliminary *schema matching* process, which automatically identifies the *semantic* and *structural* similarities between the schema elements to be used in the subsequent operation of *query rewriting*, in which a query written on a source schema is automatically rewritten in order to be compatible with the other useful XML documents. The proposed approach has been implemented in a web service, named XML S^{3}MART, which is part of the open architecture proposed in the ongoing Italian CNR co-funded ECD Project.

1 Introduction

In recent years, the constant integration and enhancements in computational resources and telecommunications, along with the considerable drop in digitizing costs, have fostered development of systems which are able to electronically store, access and diffuse via the Web a large number of digital documents and multimedia data. In such a sea of electronic information, the user can easily get lost in his struggle to find the information he requires. For these reasons, the concept of *Digital Library* (*DL*) has become a pivotal one: Exactly as a physical library, a DL contains a collection of documents that are at the users' disposal, however it goes much further. In fact, along with the documents themselves, a good DL offers an entire ensemble of systems and services designed to help users to easily find and access the data they are looking for. DLs are now widely available all over the web, but they are still far from perfect in delivering such enhancements to the user.

This is the challenging scenario of the ongoing Italian CNR co-funded Project "Technologies and Services for Enhanced Content Delivery" (ECD Project): It is aimed at producing new and advanced technologies in order to enable the development of the next generation of Digital Libraries, offering enhanced contents and services such as thematic catalogues, media collections (audio, video, WAP) and, most importantly, advanced search engines with a cutting-edge search effectiveness. In the next generation DL, data (textual documents or even metadata on multimedia items) are expressed in XML, one of the most open and powerful inter-communication standards available today, and are associated to XML Schemas; queries submitted to the DL search engine are written in XQuery [1], a language expressive enough to allow users to perform structural inquiries, going beyond the "flat" bag of words approaches of common plain

* The present work is partially supported by the "Technologies and Services for Enhanced Content Delivery" Fondo Speciale Innovazione 2000 Project.

text search engines. Such high flexibility could also mean more complexity: Hardly a user knows the exact structure of all the documents contained in the DL. Further, XML documents about the same subject and describing the same reality, for instance compact disks in a music store, but coming from different sources, could use largely different structures and element names, even though they could be useful in order to satisfy the user's information need. Given those premises, the need for a method which allows to perform queries on all the useful documents of the DL, also on the ones which do not comply with the structural part of the query itself, becomes apparent.

Recently, several works took into account the problem of answering approximate structural queries against XML documents. Much research has been done on the instance level, trying to reduce the approximate structural query evaluation problem to well-known unordered tree inclusion (e.g. [3, 13]) or tree edit distance [6] problems directly on the data trees. However, the process of unordered tree matching is difficult and extremely time consuming; for instance, the edit distance on unordered trees was found in [14] NP hard. On the other hand, a large number of approaches prefer to address the problem of structural heterogeneity by first trying to solve the differences between the schemas on which data are based (e.g. [5, 8, 9]). However, most of the work on XML schema matching has been motivated by the problem of schema integration and the fundamental aspect of query rewriting remains a particularly problematic and difficult to be solved aspect [12]. Conversely, most of the works on query rewriting do not actually benefit from the great promises of the schema matching methods, and, while presenting complex theoretical studies [11], do not propose practical and efficient ways of performing the rewriting operation itself.

In this paper, we present the preliminary results of the ongoing research activity we are carrying out in the context of the ECD Project in order to develop a service, named XML S^{3}MART (XML Semantic Structural Schema Matcher for Automatic query RewriTing), which will enable effective and efficient structural approximate searches among large numbers of "related" XML documents. In particular, we propose an approach for *approximate query answering* which relies on the available schema information and does not require explicit navigation of the XML data instances. A *schema matching* process extracts the *semantic* and *structural* similarities between the schema elements which are then exploited to perform the *rewriting* of the submitted queries, making them compatible with the available documents' structure. The paper is organized as follows: In Sec. 2 we discuss the role of the XML S^{3}MART service in the architecture proposed in the ECD project. Then, in Sec. 3 and 4 we analyse more deeply the matching and rewriting features. Finally, Section 5 concludes the paper.

2 Overview of the XML S^{3}MART Service

From a technological point of view, the development of the next generation of DLs must rely on solutions allowing easy extension of the offered functionalities and promoting information exchange between different systems and/or entire different libraries. To this end, the ECD project proposes an architecture which is *open*, that is partitioned into a series of autonomous modules, *web services* [2], which cooperate in order to make the DL data and its services available and accessible on the web. The use of XML together with web services gives the architecture an high level of inter-operability. It relies on many services for the access and management of DLs such as automatic summary and link generators, clients that will allow users to take personal notes in the documents of their interest, wrapper modules for the XML conversion of non-XML data, and so on.

One of the fundamental issues on which the ECD project focuses is the need of an efficient and effective access to multiple digital library corpora. Figure 1 shows how the

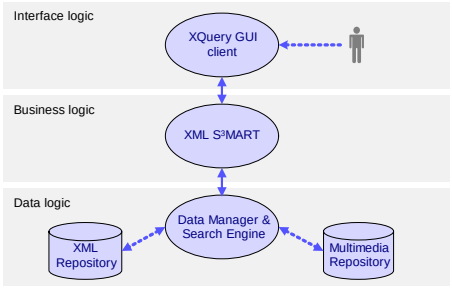

Fig. 1. The role of schema matching and query rewriting in the next-gen DL search engine

"core" services interact in order to provide an advanced search engine functionality to the user. By exploiting a graphical user interface, users can query the available XML corpora by drawing their request on one of the XML Schemas (named *source schema*). Obviously, all the documents in the Digital Library associated to such schema can be straightforwardly retrieved by submitting the given query to the search engine service. However, in the DL there usually exist large amounts of documents which, even though being associated to different XML Schemas, and thus being incompatible with the original query, could be very interesting for the user's information need. XML S^3MART makes it possible to fully exploit the whole DL content, by rewriting the query in order to search this enlarged document set. In particular, the query expressed on the source schema is automatically rewritten into a set of XML queries, one for each of the XML schemas the other useful documents are associated to. Such schemas are called *target schemas*. Then, the resulting XML queries are submitted to the underlying data manager and search engine, which efficiently access the data and return the results. Finally, XML S^3MART gathers and ranks the results and sends them to the user interface component. Notice that the returned results can be actual XML textual documents but also multimedia data for which XML descriptions are available in the DL.

In the following, we will discuss the solution adopted in the XML S^3MART service to parse a particular query and automatically rewrite it.

The basic premise the offered service relies on is that the structural parts of the documents in the DL corpora are described by XML schemas and are used to search the documents as they are involved in the query formulation. For these reasons, we rewrite the submitted query on the other useful documents which do not comply with the source schema by exploiting the similarities between their schemas, the target schemas, and the source schema itself. Indeed, due to the intrinsic nature of the semistructured data, all such documents can be used to answer the query only if, though being different, the target schemas share some similarities with the source one, both *structural* (similar structure of the underlying XML tree) and *semantical* (employed terms have similar meanings) ones. Being such similarities independent from the queries which could be issued to the service, they are identified by a preliminary *schema matching* operation. Then, using all the information extracted by such analysis, the operation of *query rewriting* can be performed in a completely automatic, effective and efficient way.

3 Semantic and Structural Schema Matching

The schema matching operation is performed through three sub-processes, the first two of which, structural expansion and semantic annotation, are needed to maximize the effectiveness of the third phase, the real matching one.

Structural Schema Expansion. In the structural schema expansion phase, the *structure* of each XML schema is modified and expanded in order to make the structural relationships between the involved elements more explicit and thus to summarize all the possible variants of the structural part of the XML documents complying with the XML schema. As a matter of fact, searches are performed on the available XML documents and an XML query usually contains paths expressing the structural relationships between elements and attributes. The resulting expanded schema thus abstracts from several complexities of the XML schema syntax, such as complex type definitions, element references, global definitions, and so on.

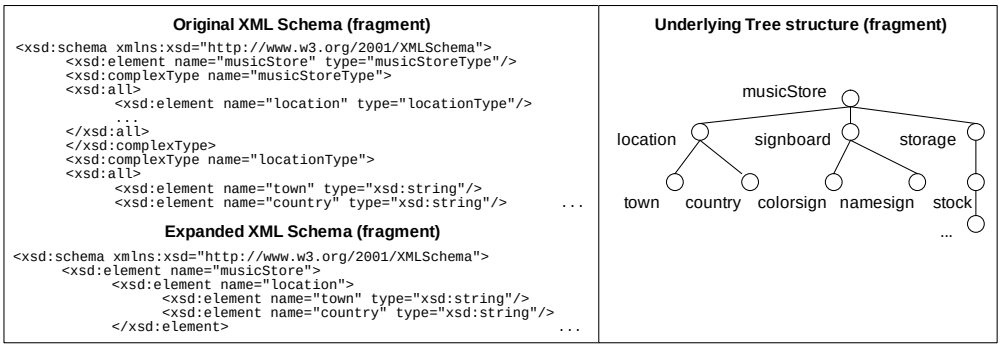

Fig. 2. Example of structural schema expansion (Schema A)

Consider, for instance, Figure 2 showing a fragment of an XML Schema describing the structural part of documents about music stores and their merchandiser, along with a fragment of the corresponding expanded schema file and a representation of the underlying tree structure expressing the structural relationship between the elements which can appear in the XML documents complying with the schema. As can be seen from the figure, the original XML Schema contains, along with the element definitions whose importance for the query rewriting task is definitely central (i.e. elements "`musicStore`", "`location`"), also type definitions (i.e. complex types "`musicStoreType`", "`locationType`") and regular expression keywords (i.e. "`all`"), which may interfere or even distort the discovery of the real underlying tree structure. In general, XML Schema constructions need to be resolved and rewritten in a more explicit way in order for the structure of the XML Schema file to be the most possibly similar to its underlying conceptual tree structure. Going back to the example of figure 2, the element "`location`", for instance, is conceptually a child of "`musicStore`": This relation is made explicit only in the expanded version of the schema, while in the original XML Schema "`location`" was the child of a "`all`" node, which was child of a complex type. Further, every complex type was defined globally and was therefore child of the root, revealing a misleading and falsely linear structure.

Semantic Schema Annotation. After having made explicit the structural relationships of a schema with the expansion process, a further step is required in order to refine and complete even more the information delivered by each schema, thus maximizing the successive matching computation's effectiveness. This time the focus is on the *semantics* of the terms used in the element and attribute definitions. In this step, each term is disambiguated, that is its meaning is made explicit as it will be used for the identification of the semantical similarities between the elements and attributes of the schemas, which actually rely on the distance between meanings. To this end, XML S^3MART exploits one of the most known lexical resources for the English language: WordNet

[10]. The WordNet (WN) lexical database is organized conceptually in synonym sets or *synsets*, representing different meanings or *senses*. Each term in WN is usually associated to more than one synset, signifying it is polysemic, i.e. it has more than one meaning. At present, term disambiguation is implemented by a semi-automatic operation where the operator is required to "annotate" each term with the right meaning. To this end, the operator can employ the XML S^{3}MART annotation module, which extracts all the terms available in the schema and simplifies the annotation process with an easy to use and complete GUI. In particular, the operator is required to associate each of these terms with the best candidate among the WN terms and to select one of its synsets. In the first task, the module helps the operator by presenting a small list of possible matching WN terms, and, for each of them, all the available WN synsets, along with their description. For instance, for the schema term "`musicStore`" the system automatically identifies its sub-components "`music`" and "`store`" as valid WN terms. Moreover, the selection of the right synset is simplified by presenting the user the hypernym tree of each of the synsets of each term. When all terms have been annotated, the system automatically outputs an *expanded+annotated* version of the schema, which has the same structure of the expanded version but where each term is associated with the chosen WN term and synset code (for instance "`musicStore`" becomes "`musicStore#store-3662968`").

Matching Computation. The Matching Computation phase performs the actual matching operation between the expanded and annotated schemas made available by the previous steps. For each pair of schemas, we identify the "best" matchings between the attributes and the elements of the two schemas by considering both the structure of the corresponding trees and the semantics of the involved terms. Indeed, in our opinion the meanings of the terms used in the XML schemas cannot be ignored as they represent the semantics of the actual content of the XML documents. On the other hand, the structural part of XML documents cannot be considered as a plain set of terms as the position of each node in the tree provides the context of the corresponding term. At the end of the matching process, the computed matchings and the corresponding similarity scores, together with the information about the schema nodes are stored in an XML document, named `matching.xml`, to be used in the rewriting phase.

The steps we devised for the matching computation are partially derived from the ones proposed in [9] and are the following:

1. the involved schemas are first converted into directed labelled graphs following the RDF specifications [7], where each entity represents an element or attribute of the schema identified by the full path (e.g. "`/musicStore/location`") and each literal represents a particular name (e.g. "`location`") or a primitive type (e.g. "`xsd:string`") which more than one element or attribute of the schema can share. As to the labels on the arcs, we mainly employ two kinds of them: "`child`", which captures the involved schema structure, and "`name`". Such label set can be optionally extended for further flexibility in the matching process. From the RDF graphs of each pair of schemas a *pairwise connectivity graph* (PCG), involving node pairs, is constructed [9]; in particular, $((x, y), l, (x', y')) \in PCG(A, B)$ iff $(x, l, x') \in A$ and $(y, l, y') \in B$, where A and B are the two involved RDF graphs and the triples $(source, label, target)$ represent the edges of the graphs.
2. Then an initial similarity score is computed for each node pair contained in the PCG. In particular, in order to exploit the semantics of the terms in the XML schemas provided in the annotation phase, we follow a *linguistic approach* in the computation of the similarities between pairs of literals (names), which quantifies the distance between the involved meanings by comparing the WN hypernyms hierarchies of the involved synsets. In this case, the scores for each pair of synsets

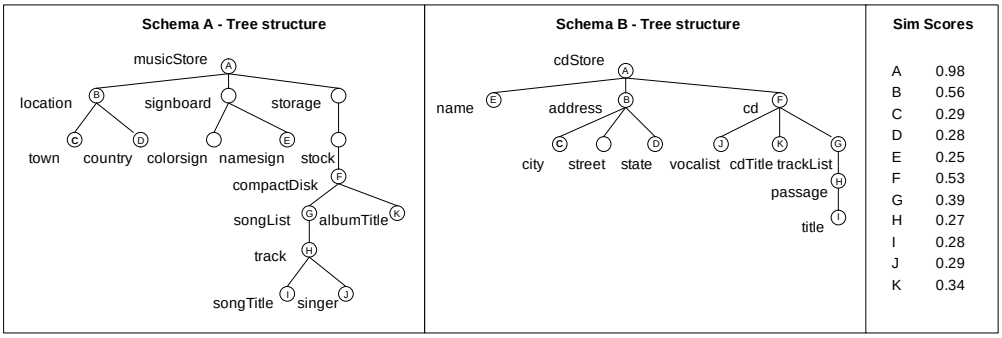

Fig. 3. Example of a schema matching result between Schema A and Schema B. Each match, represented by a letter, is associated to a similarity score (shown on the right).

(s_1, s_2) are obtained by computing the depths of the synset in the WN hypernyms hierarchy and the length of the path connecting them as follows:

$$\frac{2 * \text{depth of the least common ancestor}}{\text{depth of } s_1 + \text{depth of } s_2}$$

3. The initial similarities, reflecting the semantics of the single node pairs, are refined by an iterative fixpoint calculation as in the similarity flooding algorithm [9], which brings the structural information of the schemas in the computation. In fact, this method is one of the most versatile and also provides realistic metrics for match accuracy [4]. The intuition behind this computation is that two nodes belonging to two distinct schemes are the more similar the more their adjacent nodes are similar. In other words, the similarity of two elements propagates to their respective adjacent nodes. The fixpoint computation is iterated until the similarities converge or a maximum number of iterations is reached.
4. Finally, we apply a stable marriage filter which produces the "best" matching between the elements and attributes of the two schemas. The stable marriage filter guarantees that, for each pair of nodes (x, y), no other pair (x', y') exists such that x is more similar to y' than to y and y' is more similar to x than to x'.

Figure 3 shows an example of matching between two schemas obtained by applying the previous steps. Each match is identified by the same letter inside the nodes.

4 Automatic Query Rewriting

Exploiting the only information contained in `matching.xml`, the query rewriting service makes it possible to automatically and efficiently rewrite a given FLWOR XQuery, written w.r.t. a source schema, on the target schemas available in the matching file. Each rewrite is assigned a score, in order to allow the ranking of the results retrieved by the query. The query rewriting in XML S^3MART is simplified by the fact that the previous phases were devised for this purpose. The expanded structure of the schemas summarizes the actual structure of the XML data on which the query is formulated; the matching phase is based on such structure and produces a stable matching between the elements and attributes, identifying them with their full path. Since in a FLWOR expression paths have a key role, the rewriting of the whole query is greatly facilitated. In short, after having expanded each path in the `WHERE` and `RETURN` clause of the

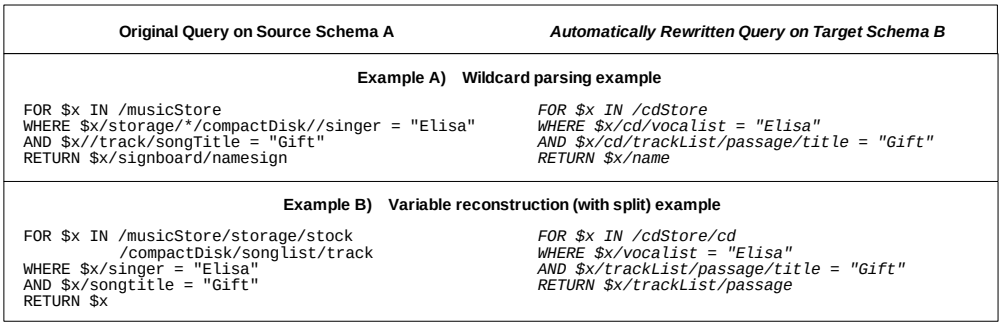

Fig. 4. Examples of query rewriting between Schema A and Schema B

submitted query by substituting each variable with the corresponding path, the XML S^{3}MART query rewriting process rewrites the query for each of the target schemas in the following way:

1. all the path tokens in the query are rewritten, exploiting the information available in the matching file for the nodes in the given source schema and target schema (e.g. path "`/musicStore/storage/stock/compactDisk`" is automatically rewritten in the corresponding best match, "`cdStore/cd`");
2. a variable is reconstructed in order to link all the rewritten path tokens (its value will be the longest common prefix of the involved paths) and a new "`FOR`" clause with its definition is inserted at the beginning of the rewritten query;
3. *a score* is assigned to the rewritten query. It is the average of the scores assigned to each path rewriting which is based on the similarity between the involved nodes, as specified in the matching file.

Figure 4 shows two examples of query rewriting. The submitted queries are written by using Schema A of Fig. 3 and the rewriting on Schema B, as produced by our query rewriting modules, is shown on the right of the figure. Example A involves the rewriting of a query containing paths with wildcards. In order to successfully elaborate them, the matching file is searched not exactly but by means of regular expressions string matching. For instance, the only path of the tree structure of schema A satisfying the pattern "`/musicStore/storage/*/compactDisk//singer`" is "`musicStore/storage/stock /compactDisk/songList/track/singer`" and its Schema B match will be the one used in the rewrite. When more than one path of the source schema satisfy a wildcard path, all the corresponding paths are rewritten and put in an OR clause. Example B demonstrates the XML S^{3}MART behavior in reconstructing a variable: The value of the "`$x`" variable in the submitted query is the path of the element "`track`" in Schema A. The corresponding element in Schema B is "`passage`" (see Figure 3, where it has the same letter: H), however directly translating the variable value in the rewritten query would lead to a wrong rewrite: While the elements "`singer`" and "`songTitle`" are descendants of "`track`", in Schema B a "split" has been performed on the paths and the corresponding elements "`songTitle`" and "`vocalist`" are not both descendants of "`passage`". Instead, the query is correctly rewritten by substituting the variable value, rewriting the paths and *then* reconstructing the variable, which holds the value of path "`cdStore/cd`".

Finally, notice that XML data used in the project only allow leaf elements to hold a textual value; however, the matching phase might produce matchings where a leaf element of a schema matches a middle element of the other. In such cases, the "`WHERE`"

conditions expressed on a source schema leaf element are rewritten as "OR" clauses on the leaf elements' descendants of the matching target element.

5 Conclusions

In this paper, we presented the preliminary results of the ongoing research activity we are carrying out in the context of approximate XML query answering when the schemas of the XML documents are available. The proposed approach has been implemented in a web service prototype, XML S^{3}MART, which is currently under testing and evaluation. The preliminary experimental results on query rewriting effectiveness are promising; as we expected, the rewriting efficiency is also particularly encouraging, thanks to the "lightness" of our method which relies on schema information and does not require explicit navigation of the XML data. In the near future, the web service will be integrated in the open architecture of the ECD project and will be presented at the ECD Industrial Day which will be held in may and where its performance will be tested on real data collections such as ECHO Project and MPEG-7 metadata. In future works we will present a detailed analysis of the enhancements in the effectiveness of search retrieval, along with more complex matching and rewriting examples, the discussion of which was not possible in this paper due to space limitations. Further, we plan to enhance our method with additional advanced features, such as automatic structural word sense disambiguation and automatic deduction of the underlying schema information from the submitted query.

References

1. S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. W3C Working Draft.
2. E. Castro-Leon. The Web within the Web. *IEEE Spectrum*, 41(2):36–40, 2004.
3. P. Ciaccia and W. Penzo. Relevance ranking tuning for similarity queries on xml data. In *Proc. of the VLDB EEXTT Workshop*, 2002.
4. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proc. of the 2nd Int. Workshop on Web Databases*, 2002.
5. H. Do and E. Rahm. COMA – A system for flexible combination of schema matching approaches. In *Proc. of the 28th VLDB*, pages 610–621, 2002.
6. S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *Proc. of ACM SIGMOD*, pages 287–298, 2002.
7. O. Lassila and R. Swick. Resource Description Framework (RDF) model and syntax specification. W3C Working Draft WD-rdf-syntax-19981008.
8. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *Proc. of the 27th VLDB*, pages 49–58, 2001.
9. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and ist Application to Schema Matching. In *Proc. of the 18th ICDE*, 2002.
10. G.A. Miller. WordNet: A Lexical Database for English. In *CACM 38*, 1995.
11. Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *Proc. of the ACM SIGMOD*, pages 455–466, 1999.
12. N. Rishe, J. Yuan, R. Athauda, S. Chen, X. Lu, X. Ma, A. Vaschillo, A. Shaposhnikov, and D. Vasilevsky. Semantic Access: Semantic Interface for Querying Databases. In *Proc. of the 26th VLDB*, pages 591–594, 2000.
13. T. Schlieder and Felix Naumann. Approximate tree embedding for querying XML data. In *Proc. of ACM SIGIR Workshop On XML and Information Retrieval*, 2000.
14. Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Inf. Process. Lett.*, 42(3):133–139, 1992.