Standing on the Shoulders of Ants: Towards More Efficient XML-to-Relational Mapping Strategies*

Irena Mlýnková
Charles University, Faculty of Mathematics and Physics, Department of Software Engineering
Malostranské nám. 25, 118 00 Prague 1, Czech Republic
irena.mlynkova@mff.cuni.cz

Abstract

As XML has become a standard for data representation, it is inevitable to propose and implement techniques for efficient managing of XML data. A natural alternative is to exploit features and functions of (O)RDBMSs, i.e. to rely on their long theoretical and practical history. The main concern of such techniques is the choice of an appropriate XML-to-relational mapping strategy. In this paper we propose an enhancing of so-called adaptive mapping methods which evaluate several mapping possibilities and choose the one which suits the current application the most. We optimize the process of searching the optimal strategy using a heuristic called Ant Colony Optimization and we enhance the adaptivity using similarity of XML data and ideas of user-driven techniques.

1 Introduction

Since XML [5] has become an acknowledged standard for data representation, it has invoked a boom of implementations of W3C recommendations based on various storage strategies. Currently, the most practically used ones exploit robust and mature (object-)relational database management systems ((O)RDBMSs). Although the scientific world has proven that *native XML strategies* perform much better, they still lack a reliable and robust implementation verified by years of both theoretical and practical effort.

At present, there exists a plenty of works concerning database-based[1] XML data management. All major database vendors support XML and even the SQL standard has been extended with a new part (SQL/XML) which introduces operations on XML data. The main concern of the techniques is the choice of the way XML data are stored into relations – so-called *XML-to-relational mapping*. On the basis of exploitation of information from XML schema we distinguish *schema-oblivious* (e.g. [7]) and *schema-driven* (e.g. [10]) methods. From the point of view of the input data we distinguish *fixed* methods (e.g. [7, 10]) which store the data on the basis of their model and *adaptive* methods (e.g. [4, 8, 11, 12]), where also additional information on the future application are taken into account. And there are also techniques based on user involvement which can be divided to *user-defined* (see [1]) and *user-driven* (e.g. [2, 3, 9]), where in the former case a user defines both the relational schema and the required mapping, whereas in the latter case a user specifies just local mapping changes of a default storage strategy.

In this paper we introduce an enhancing of the *adaptive* mapping strategies which seem to be the most efficient ones, since they evaluate several mappings and choose the one which suits the target application (specified via sample set of XML documents and query workload) the most. We improve the strategy in three ways. Firstly, we improve the process of searching the optimal strategy using a heuristic called *Ant Colony Optimization*. This heuristic is based on a simple observation of nature and enables to find the suboptimal solution more efficiently than simple heuristics used in the existing papers. Secondly, we enhance the adaptation process using similarity of XML data. And finally, we combine it with the idea of *user-driven* techniques, i.e. the techniques where a user instead of queries and sample data provides information on required mapping strategies for selected data fragments.

The paper is structured as follows: Section 2 overviews the related works. In Section 3 we describe the proposed improvements and in Section 4 the architecture of the enhanced system. Finally, Section 5 provides conclusions and outlines future work.

*This work was supported in part by Czech Science Foundation (GAČR), grant number 201/06/0756.

[1] In the rest of the paper the term "database" represents an (O)RDBMS.

2 Related Work

Probably the first proposal of an adaptive method can be found in [8]. It is based on the idea of storing well structured parts of XML documents into relations and semi-structured parts using an *XML data type* which supports path queries and XML-aware full-text operations. Hence, the main concern is to identify the structured and semi-structured parts. For this purpose a sample set of XML documents, their XML schema and sample XML queries are analyzed. The other existing, so-called *cost-driven*, approaches [4, 11, 12] use a different strategy. They define a set of XML-to-XML transformations (e.g. inlining/outlining of an element/attribute, associativity, commutativity etc.), a fixed XML-to-relational mapping and a cost function which evaluates a relational schema against a given sample set of XML data and queries. Using a search algorithm, a space of possible mappings is searched and the optimal one is selected.

In case of user-driven strategies there seem to be just three representatives – *ShreX* [2], *XCacheDB* [3] and *UserMap* [9]. The first two approaches can be classified as *direct* since the user-specified annotations are just directly applied on the respective schema fragments; the remaining schema parts are stored using a fixed mapping strategy. The last mentioned approach can be specified as *indirect*, since it further exploits the user provided annotations in the adaptive strategy applied on the not annotated schema parts. The adaptive strategy is based on exploitation of similarity of XML schema fragments.

3 Proposed Algorithm

A general idea of fixed schema-driven XML-to-relational mapping methods is to *decompose* (*shred* or *map*) the given XML schema S_{init} into a set of relations $R = \{r_1, r_2, ..., r_{|R|}\}$ using a mapping strategy f_{map}. An extreme case is when S_{init} is decomposed into a single relation resulting in many null values. Other extreme is when for each element $e \in S_{init}$ a single relation is created resulting in numerous join operations.[2]

For the purpose of mapping S_{init} using f_{map} to R, the cost-driven mapping techniques exploit the following additional information:

1. a set of XML schema transformations $T = \{t_1, t_2, ..., t_{|T|}\}$, where $\forall i : t_i$ transforms a given XML schema S into an XML schema $t_i(S)$,
2. a set of sample data D_{sample} characterizing the future application, which consists of:
   (a) a set of XML documents $D = \{d_1, d_2, ..., d_{|D|}\}$ valid against S_{init},
   (b) a set of XML queries $Q = \{q_1, q_2, ..., q_{|Q|}\}$ over S_{init} and
3. a cost function f_{cost} which evaluates the cost of the given relational schema R with regard to D_{sample}.

The required result is an optimal relational schema $R_{opt} = f_{map}(S_{opt})$, i.e. a schema, where $f_{cost}(R_{opt}, D_{sample})$ is minimal. S_{opt} is the optimal XML schema derived from S_{init} using a particular sequence of transformations from T.

A typical example of schema transformations is *inlining* (t_{in}) or *outlining* (t_{out}) of an element into/out of its parent element or denoting a whole schema fragment as a single attribute – so-called *unshredding* (t_{un}). For instance, considering the following example of S_{init} fragment (assuming that the undeclared elements have `#PCDATA` data type):

```
<!ELEMENT employee (name, address)>
<!ELEMENT name (first, middle?, last)>
<!ELEMENT address (city, country, zip)>
```

the natural target schema[3] R would involve relation:

```
empl1(name_first, name_middle, name_last,
      address_city, address_country, address_zip)
```

But if we knew that Q involves queries of the form `//employee/name`, but not of the form `//employee/name/first`, `//employee/name/middle` or `//employee/name/last`, i.e. that the user always retrieves a employee's name as a whole, a more efficient relation would be:

```
empl2(name,
      address_city, address_country, address_zip)
```

i.e. we would apply unshredding on element `name`.

But, since the complexity of S_{init} and Q can be high, it is not possible to determine the optimal relational schema so easily. Hence, the key problem of cost-driven approaches is how to find S_{opt}, resp. R_{opt}.

A naive but illustrative search strategy can be based on using a "brute force". It first generates the set Σ of results of all possible sequences of transformations from T applied on S_{init}. Then it searches for schema $S \in \Sigma$ with minimal cost $f_{cost}(f_{map}(S), D_{sample})$, i.e. S_{opt}, and returns the corresponding optimal relational

[2] Note that since schema-oblivious mapping methods view an XML document as general directed tree with several types of nodes, we can speak about schema decomposition too.

[3] For simplicity we omit obvious keys and foreign keys.

schema $R_{opt} = f_{map}(S_{opt})$. The complexity of the naive algorithm obviously strongly depends on the set T. It can be proven that even a simple set T causes the problem of finding the optimal schema to be NP-hard [12]. We can even view the search problem as a kind of combinatorial optimization problem.

Definition 1 *A model $P = (\Sigma, \Omega, \phi)$ of a* combinatorial optimization problem (COP) *consists of a search space Σ of possible solutions to the problem, a set Ω of constraints over the solutions and an* objective function $\phi : \Sigma \to \mathbb{R}_0^+$ *to be minimized.*

In our case Ω is given by the general requirements of mapping strategies such as, e.g., completeness, correctness, losslessness etc., and $\phi(S) = f_{cost}(f_{map}(S), D_{sample})$ for $\forall S \in \Sigma$. With this view of the problem, almost any respective heuristic for COPs can be used. Surprisingly, all the existing works use one of the simplest approaches – the greedy search algorithm or its variations. Analyzing three generally efficient initial schemes, the authors of paper [12] have experimentally proven that a good choice of an initial schema is crucial and can lead to better results. But, together with the main disadvantage of greedy search strategies – the fact that they get stuck in local suboptimums – the existing approaches may not be able "to leave" S_{init}. Hence, the first enhancing we propose is optimization of the search strategy.

3.1 Ant Colony Optimization (ACO)

For the purpose of searching S_{opt} we use a modification of more efficient *ACO* heuristic [6]. It is based on observations of nature, in particular the way ants exchange information they have learnt. A set of artificial "ants" $A = \{a_1, a_2, ..., a_{|A|}\}$ iteratively search the space Σ trying to find the optimal solution $S_{opt} \in \Sigma$ s.t. $\phi(S_{opt}) \leqslant \phi(S)$ for $\forall S \in \Sigma$. In i-th iteration each $a \in A$ searches a subspace of Σ for a local suboptimum until it "dies" after performing N_{ant} steps. While searching, an ant a spreads a certain amount of "pheromone", i.e. a positive feedback which denotes how good solution it has found so far using the used transformation sequence. This information is exploited by ants from the following iterations to choose more promising search steps. The algorithm terminates either after N_{iter} iterations or if $S'_{opt} \in \Sigma$ is reached s.t. $\phi(S'_{opt}) \leqslant T_{max}$, where T_{max} is a required threshold.

Each step of an ant consists of generating of candidate steps and execution of one of them. The executed step is selected randomly, but on the basis of probability given by ϕ and feedback (i.e. pheromone) spread by other ants. (Note that the greedy search is a special case of ACO, where $|A| = 1$, $N_{iter} = 1$ and only the best possible candidate step is always executed.)

3.1.1 Generating a Set of Possible Steps

The set of possible steps should consist only of relevant transformations and, at the same time, it should not omit a step which leads to the optimum. The most complex set of transformations has been proposed in paper [4] and consists of the following operations:

- *Inlining/outlining of elements/attributes*
- *Splitting/merging of elements*: Storing shared elements into a common table/separate tables.
- *Associativity/commutativity*
- *Union distribution/factorization*: $(a, (b|c)) = ((a, b)|(a, c))$
- *Splitting/merging repetitions*: $(a+) = (a, a*)$
- *Simplifying unions*: $(a|b) \subseteq (a?, b?)$

However, the existing systems [4,11,12] support only inlining/outlining or splitting/merging of elements. In general, the ACO heuristic does not depend on the way the steps are generated, but, since the aim of this paper is to optimize different aspects of the approach, we will further assume only three types of transformations – inlining, outlining and unshredding, i.e. our $T = \{t_{in}, t_{out}, t_{un}\}$.

3.1.2 Evaluation of Steps

Having a schema $S \in \Sigma$ and a transformation $t \in T$, the cost of applying t on S, which determines the probability of executing this step is defined as:

$$step(S, t(S)) = \phi(S) - \phi(t(S)) + ph(S, t(S)) \qquad (1)$$

where ϕ is the objective function and $ph(S, t(S)) \geqslant 0$ is the amount of pheromone assigned to this step from other ants. Hence, the key aspect is how to evaluate function ϕ. A naive approach would require:

- construction of relational schema $R = f_{map}(S)$,
- loading documents $D \in D_{sample}$ into R and
- evaluation of queries $Q \in D_{sample}$ over R.

But, naturally, such operation is quite time consuming and expensive. The existing papers [4, 11, 12] optimize the naive approach using various strategies which estimate the costs of the queries. For the purpose of demonstration of main features of the proposed improvements and with regard to the specified set T, we use a simple metric which evaluates the number of join operations necessary to evaluate queries in Q.

3.1.3 Main Advantages of Using ACO

The main advantages of the idea to use ACO instead of simple greedy search are as follows:

1. ACO enables to perform multiple searches through the space Σ. Since the ants move pseudo-randomly, they search larger subspace of Σ.

2. Since the randomized search enables to move from S to $t(S)$ s.t. $\phi(S) \leqslant \phi(t(S))$, the ACO heuristic does not "get stuck" in a local optimum. However, since we still keep the best-so-far schema, also ACO ensures that $\phi(S_{init}) \geqslant \phi(S_{opt})$.

3.2 Exploitation of Similarity

The aim of cost-driven methods is to find the relational schema optimal for queries in Q which represent the future application. But S_{init} may involve elements which occur on no *access path* of queries in Q. Obviously, in this case the strategies are not able to find the optimal mapping, because they have no efficiency feedback. For instance, if the previous sample S_{init} involves also the following declarations:

```
<!ELEMENT company (co-name, address)>
<!ELEMENT co-name (title, type?)>
```

and $Q = \{$`//employee/name`$\}$, we cannot exploit the adaptive strategy for element `company` and we would store it, e.g., in the following way:

```
co1(co-name_title, co-name_type,
    address_city, address_country, address_zip)
```

But since elements `name` and `co-name` are semantically and structurally similar, we may assume that they should be stored in a similar way, because they are likely to be processed similarly:

```
co2(co-name,
    address_city, address_country, address_zip)
```

Hence, in our second improvement we result from approach proposed for system UserMap [9], where the user-provided annotations are firstly directly applied on specified schema fragments and, then, regarded as "hints" how to store particular XML patterns. The adaptive enhancing of the system iteratively searches for schema fragments similar to these patterns in the rest of the schema and maps them in the same way.

For the purpose of our approach we can only slightly modify the idea used in UserMap. Obviously, we do not have any user-provided annotated schema fragments. But, we are able to find the optimal mapping strategies for schema fragments involved in access paths of queries in Q. Hence, we may view these schema fragments as fragments annotated by a user and reuse the whole approach proposed for UserMap without complex changes.

3.3 Exploitation of Schema Annotations

In the previous section we have described a way how to exploit an idea proposed for user-driven strategies for the purpose of cost-driven strategies. But we can go even further. Our motivation is that for some schema fragments a user is able to provide sample data D and queries Q (similarly to the previous examples), but for others (e.g. XHTML fragments) it is easier to directly specify the mapping strategy (e.g. unshredding). Hence, we need to use partly cost-driven and partly user-driven strategies. However, since both are able to store distinct schema fragments using different strategies, their combination is quite natural.

From the point of view of previous proposals, we get another input information – the schema annotations. To exploit it, we use the following observation: The set T involves only simple schema transformations, but using a sequence of them we can produce various complex storage strategies. However, having an annotated schema, for selected schema fragments the complex storage strategies are directly specified by a user. For instance, if we consider the element `employee`, the relational schema `empl2` can be derived from fully outlined S_{init} using various sequences of inlining and unshredding, such as, e.g.

$$s_1 = [t_{in}(\texttt{city}),\ t_{in}(\texttt{country}),\ t_{in}(\texttt{zip}),\ t_{in}(\texttt{address}),\ t_{in}(\texttt{first}),\ t_{in}(\texttt{middle}),\ t_{in}(\texttt{last}),\ t_{in}(\texttt{name}),\ t_{un}(\texttt{name})]$$

$$s_2 = [t_{un}(\texttt{name}),\ t_{in}(\texttt{name}),\ t_{in}(\texttt{city}),\ t_{in}(\texttt{country}),\ t_{in}(\texttt{zip}),\ t_{in}(\texttt{address})]$$

But if a user annotates element `employee` e.g. with the Hybrid mapping strategy [10] which ensures inlining of all non-repeatable subelements, the respective sequence of transformations simplifies to:

$$s_3 = [t_{un}(\texttt{name})]$$

If we generalize this idea, we can extend the set of possible steps of an ant with *composite transformations* representing the user-specified annotations and hence speed-up the search process. The only obvious limitation is that the composite transformations can be applied only schema schema fragments structurally/sematically similar to the original annotated ones.

4 System Architecture

The proposed improvements can be applied to any user-driven or cost-driven system. However, we have enhanced system UserMap, because it has most suitable features. Figure 1 depicts the extensions.

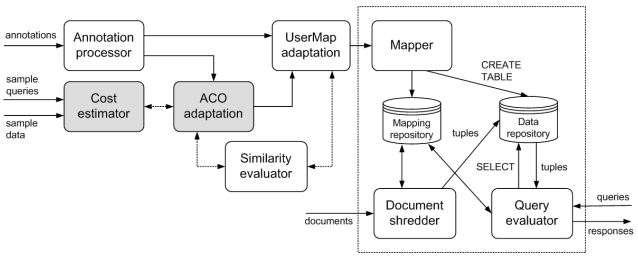

Figure 1. Architecture of extended UserMap

The dotted box involves a classical part of any XML-to-relational system: *Mapper* creates tables of schema R in a *data repository* and stores supplemental information in a *mapping repository*. *Document shredder* is responsible for shredding the incoming XML data into tables of R. And *query evaluator* transforms an input XML query into SQL query, evaluates it, and transforms the returned data back into XML format.

The remaining white modules are original modules of UserMap: *Annotation processor* checks correctness of schema annotations and provides them to a module responsible for *UserMap adaptation* which searches for similar fragments. The similarity (of the given schema fragments) is evaluated using *similarity evaluator*.

The grey modules cover our proposed improvements: *Cost estimator* estimates the cost of a given XML-to-relational mapping using sample XML data and query workload. The module responsible for *ACO adaptation* performs the ACO search strategy (see Section 3.1). For this purpose it exploits the information on user-specified annotations (see Section 3.3) from annotation processor and similarity of schema fragments (see Section 3.2, 3.3) evaluated using similarity evaluator. The ACO adaptation provides its results to UserMap adaptation, i.e. it extends the set of annotated schema fragments with the newly identified ones.

5 Conclusion

The aim of this paper was to illustrate that since the idea of database-based XML processing methods is still up-to-date, the techniques should and can be further enhanced. We have proposed an enhancing of adaptive mapping strategies using ACO heuristic, similarity of XML data and ideas of user-driven methods. These improvements enable to avoid getting stuck in local optimums, to find an optimal mapping for greater subset of source schema and to speed up the search process.

Our future work will focus mainly on the idea of dynamic adaptability. It will solve the two persisting disadvantages of adaptive approaches – the user-unfriendly requirement to provide plenty of information on the future application beforehand and the fact that the efficiency can worsen even with minor changes in the application, i.e. changes in queries. Since the ACO heuristic can be modified for dynamic systems as well, it seems that the choice of this approach is correct and promising.

References

[1] S. Amer-Yahia. *Storage Techniques and Mapping Schemas for XML*. Technical Report TD-5P4L7B, AT&T Labs-Research, 2003.

[2] S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem. In *WIDM'04*. ACM, 2004.

[3] A. Balmin and Y. Papakonstantinou. Storing and Querying XML Data Using Denormalized Relational Databases. *The VLDB Journal*, 14(1), 2005.

[4] P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML Schema to Relations: A Cost-based Approach to XML Storage. In *ICDE'02*. IEEE, 2002.

[5] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C, 2006.

[6] M. Dorigo, M. Birattari, and T. Stutzle. An Introduction to Ant Colony Optimization. Technical Report 2006-010, IRIDIA, 2006.

[7] D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDMBS. *IEEE Data Eng. Bull.*, 22(3), 1999.

[8] M. Klettke and H. Meyer. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. In *Selected papers from WebDB'00 Workshop*. Springer, 2001.

[9] I. Mlynkova. A Journey towards More Efficient Processing of XML Data in (O)RDBMS. In *CIT'07*. IEEE, 2007.

[10] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB'99*. Morgan Kaufmann, 1999.

[11] W. Xiao-ling, L. Jin-feng, and D. Yi-sheng. An Adaptable and Adjustable Mapping from XML Data to Tables in RDB. In *VLDB'02 Workshop EEXTT and CAiSE'02 Workshop DTWeb*. Springer, 2003.

[12] S. Zheng, J. Wen, and H. Lu. Cost-Driven Storage Schema Selection for XML. In *DASFAA'03*. IEEE, 2003.