ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.4, April 2007, pp.1026−1038
DOI: 10.1360/jos181026

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

Macor:一种表示嵌套模式映射的可维护 XQuery 模型*

钱 钢+, 董逸生

(东南大学 计算机科学与工程学院,江苏 南京 210096)

Macor: A Maintainable XQuery Model for Representing Nested Schema Mappings

QIAN Gang+, DONG Yi-Sheng

(School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

+ Corresponding author: Phn: +86-25-84410962, E-mail: qiangang@seu.edu.cn, http://www.seu.edu.cn

Qian G, Dong YS. Macor: A maintainable XQuery model for representing nested schema mappings. *Journal of Software*, 2007,18(4):1026−1038. http://www.jos.org.cn/1000-9825/18/1026.htm

Abstract: This paper proposes a model called mapping & correlation (Macor) to represent nested schema mappings. With Macor, a full mapping is modeled as a number of simple atomic ones that are correlated with correlations. The expressive power of Macor by a fragment of XQuery called CoXQ is studied and the issues in implementing Macor are addressed. Preliminary experimental results show that with Macor a full mapping can be modeled incrementally in a piecemeal fashion, and in refining or maintaining the mappings, Macor makes it possible to locate modifications to few atomic mappings and correlations, and reuse other parts of the full mapping.
Key words: schema mapping; mapping model; XQuery; XML; data exchange; data sharing

摘 要: 提出了一种称为 Macor 的模型来表示嵌套模式之间的映射关系.Macor 将一个完整的模式映射表示成众多简单的原子映射,并用关联关系将它们连接在一起.进一步根据XQuery分析了Macor的表达能力,并处理了Macor的实现问题.初步的实验结果表明,通过 Macor 模型,一个复杂的映射不仅可以按照增量方式逐步建立,还能将修改和维护限制在局部的原子映射和关联关系上.
关键词: 模式映射;映射模型;XQuery;XML;数据交换;数据共享
中图法分类号: TP311 文献标识码: A

1 Introduction

Nowdays, there is a rapid growth of requirements for integrating, exchanging and transforming data stored in different autonomous heterogeneous sources. Modern data-sharing architectures use schema mappings to specify how data instances over source schemas correspond to data instances over a target schema[1,2]. To enable data sharing, the user has to first construct the semantic mappings between the target and the source schemas. Also, as the application requirements or the schemas change, the user has to maintain and modify the early constructed

* Supported by the High-Tech Project of 'Tenth Five-Year-Plan' of Jiangsu Province of China under Grant No.BG2001013 (江苏省十五高科技项目)

Received 2005-11-25; Accepted 2006-04-27

mappings. A number of tools have recently been developed to assist the user in such processes by increasing the abstraction level[3,4], semi-automatically discovering mappings[5,6] or preserving their semantics as schemas evolve[7,8]. However, it should be noted that these processes are expected to involve human input, and in practice it is still inevitable for the user to manually construct and maintain the mappings. Currently, schema mappings are mainly represented as undecorated expressions such as SQL or XQuery queries. Besides the structural and semantic discrepancies existing in different schemas, such naive representation can be another source that complicates the above processes. Worse of all, this problem is becoming more acute with the pervasive adoption of XML model in data sharing applications, where a mapping may be very large, because it computes nested XML documents and, therefore, is as complex as the target schema[9].

In light of the above observations, we propose a model, called *mapping* & *correlation* (*Macor*), to represent schema mappings between nested schemas. In Macor, a full mapping consists of a number of simple, partial *atomic mappings*, which are correlated using explicit *correlations*. To some extent, atomic mappings define local views of each single schema element and correlations denote semantic relationships between the atomic mappings. In contrast with an undecorated representation, Macor structurally models a mapping as a *Macor tree*. As a result, to construct a full mapping for the whole target schema, the user can first independently construct atomic mappings for each single target schema element, and then incrementally correlate them using the correlations. Such *flexibility* in mapping construction makes Macor adapt well to complex applications. On the other hand, in maintaining or refining mappings, Macor makes it possible to locate modifications to sub-mappings (instead of the full mapping), i.e., certain local atomic mappings and related correlations, and remain and reuse the other parts of the mappings.

From a Macor tree, it is possible to compute an equivalent transformation, expressed as a single query. In particular, we identify a certain fragment of XQuery, called *CoXQ*, which captures the main features of XQuery and can express mostly the practical schema mappings, to characterize the expressiveness of Macor. This characterization is useful. For those mappings having been expressed directly as XQuery expressions, if they fall into CoXQ, then we can transform them into the corresponding Macor trees to facilitate the maintenance. Further, to compute the actual data transformations, a transformation in the opposite direction can also be done from a Macor tree into an equivalent CoXQ expression, which then can be optimized and executed by some XQuery engines.

A preliminary work has been reported in Ref.[10]. In this paper, the following new contributions are made.

(i) We precisely define the Macor model formally;

(ii) Using CoXQ, we characterize the expressiveness of Macor, and show the translation between the CoXQ expressions and Macor trees;

(iii) We present a case study and report some preliminary experimental results about the approach;

(iv) In implementation we provide additional mechanisms, e.g., *quasi Macor tree* and *Macor operations*, to combine Macor with current semi-automatic mapping techniques.

The paper is organized as follows. An overview of Macor and its intuition are first given in Section 2. Then the detailed model is presented in Section 3. Section 4 presents a case study, and Section 5 studies the expressive power of Macor. Section 6 briefly addresses the issues in implementing Macor, and reports the experimental results. Related work is discussed in Section 7. Finally, Section 8 concludes.

2 An Overview

We provide an overview of Macor by means of a user scenario. Figure 1 shows portions of three nested schemas (e.g., DTD) *T*, *S*1 and *S*2. The schema *T* depicts a user interface for accessing an online bookshop, and *S*1 and *S*2 indicate formats of the books and reviews information stored in different sources, respectively. For

expository reasons, we assume that each book is identified by its title value, and author is by name. Note that the element author2 is complex type, while author1 is atomic type and denotes author's name. Element occurrence frequency (i.e., ?, *, +, ε) is associated with the corresponding edges. Throughout the paper we use the schemas *T*, *S*1 and *S*2 to discuss schema-to-schema mappings. Though these schemas are much simpler than in practice, they are enough to illustrate the usefulness of a flexible, maintainable mapping representation model.

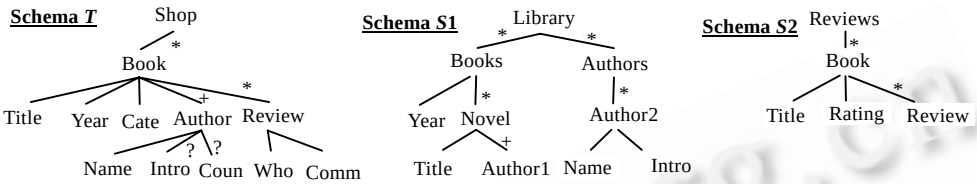


Fig.1 Example of nested schemas

Given a (set of) source *schema*(*s*) S_S and a target schema S_T, a *mapping M* between S_S and S_T is a *query* that, given an input of instances conforming to S_S, could always compute *semantic-valid* and *application-specific* target instances, that is, the (virtually) computed target instances would conform to the target schema S_T, as well as the application requirements, e.g., for the above scenario, the favorites of the bookshop. Intuitively, we regard a schema as a pair of ⟨*elems*,*cons*⟩, where *elems* denotes a finite set of single schema elements with different names and *cons* are constraints over *elems*. Our discussions consider the simplified XML schemas with the tree structures like in Fig.1. For such a schema tree, *elems* includes *empty*, *tag* and *text* types of schema elements. For example, the elements such as book and tile are tag type; those like name_txt are text type (not shown in Fig.1 for readability). Further, *cons* may be defined by the constraints such as *child*, *next-sibling*, *cardinality* and *reference*, which respectively model the nesting structure, element sequence, element occurrence frequency, and referential relationship[11]. Instances of a schema element can be empty, tagged or text data nodes, and if the instances of the schema elements in *elems* satisfy all the constraints in *cons*, then they form instances (e.g., DOM trees) over the corresponding schema. Note that the empty data nodes have intuitive semantics, i.e., they can be safely removed from a data tree.

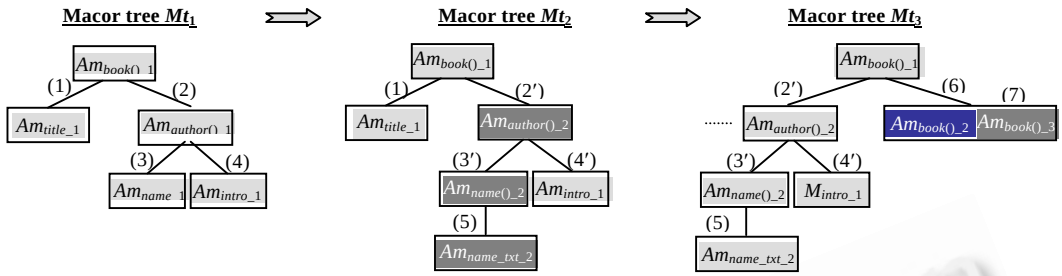
Following the above observations, Macor first models mappings between the source schemas (i.e., *S*1 and *S*2) and the single schema elements in the target schema (i.e., *T*). Such mappings are referred to as *atomic mappings*. Employing XQuery, we illustrate sample atomic mappings as follows.

$Am_{book()\_1}$: for $\$n_1$ in doc("*S*1")//novel return ⟨book⟩⟨/book⟩

$Am_{title()\_1}$: for $\$n_2$ in doc("*S*1")//novel, $\$t_1$ in $\$n_2$/title return ⟨title⟩⟨/title⟩

$Am_{title\_txt\_1}$: for $\$t_2$ in doc("*S*1")//novel/title return $\$t_2$/text()

When applied to data, atomic mappings transform single nodes into nodes (instead of subtrees into subtrees). Macor provides another language facility (called *correlation*, including *Nest*, *Join*, and *Merge*) to state how these data pieces must be glued together to obtain the target data trees. For example, the *Nest* correlation can state that the title nodes computed by $Am_{title()\_1}$ should be nested within the book nodes computed by $Am_{book()\_1}$. Further, by correlating the atomic mapping $Am_{title\_txt\_1}$, the text values can be assigned to the corresponding title nodes. With correlations, atomic mappings are organized into a tree (called *Macor tree*), where each node contains atomic mappings and each edge is labeled with a type of correlation that correlates the atomic mappings contained in the parent-child nodes. Examples of Macor trees are given in Fig.2 and will be explained in the following sections. Note that for legibility, Fig.2 uses a single atomic mapping Am_{title_1} instead of a correlation of $Am_{title()\_1}$ and $Am_{title\_txt\_1}$ to copy the title nodes together with the related text nodes from the source.



Atomic mappings	Correlations

Am_{book}()_1: **for** $\$n_1$ **in** doc("S1")//novel **return** ⟨book⟩⟨/book⟩
Am_{title}_1: **for** $\$n_2$ **in** doc("S1")//novel, $\$t_1$ **in** $\$n_2$/title **return** $\$t_1$
Am_{author}()_1: **for** $\$a_1$ **in** doc("S1")//author2 **return** ⟨author⟩⟨/author⟩
Am_{name}_1: **for** $\$a_2$ **in** doc("S1")//author2, $\$na_1$ **in** $\$a_2$/name **return** $\$na_1$
Am_{intro}_1: **for** $\$a_3$ **in** doc("S1")//author2, $\$i_1$ **in** $\$a_3$/intro **return** $\$i_1$
	(1) ($Nest$,$\$n_1=_n\$n_2$)
	(2) ($Join$,$\$n_1$/author1$=_v\$a_1$/name)
	(3) ($Nest$,$\$a_1=_n\$a_2$)
	(4) ($Nest$,$\$a_1=_n\$a_3$)

Am_{author}()_2: **for** $\$n_2$ **in** doc("S1")//novel, $\$a_4$ **in** $\$n_2$/author1 **return** ⟨author/⟩
Am_{name}()_2: **for** $\$a_5$ **in** doc("S1")//author1 **return** ⟨name⟩⟨/name⟩
Am_{name_txt}_2: **for** $\$a_6$ **in** doc("S1")//author1 **return** $\$a_6$/*text*()
	(2′) ($Nest$,$\$n_1=_n\$n_2$)
	(3′) ($Nest$,$\$a_4=_n\$a_5$) (5) ($Nest$,$\$a_5=_n\$a_6$)
	(4′) ($Nest$,$\$a_4=_v\$a_3$/name)

Am_{book}()_2: **for** $\$n_3$ **in** doc("S1")//novel **return** ⟨book⟩⟨/book⟩ Am_{book}()_3: **for** $\$b_1$ **in** doc("S2")//book **where** $\$b_1$/rating>3 **return** ⟨book/⟩	(6) ($Join^+$,$\$n_1$/author1$=_n\$n_3$/author1,*count*()>0,$\varepsilon$) (7) ($Merge$,$\$n_3$/title$=_v\$b_1$/title)

Fig.2 Examples of Macor trees that model mappings between the schemas in Fig.1: At first Mt_1 is created incrementally; then Mt_2 and Mt_3 are derived in sequence by modifying Mt_1 partially

3 Macor Model

We present Macor in detail in this section. Specifically, we focus on the language facilities provided by Macor for transforming data instances from the sources into the target. Checking whether the mapping is consistent or not is beyond the scope of the paper. Thus, in the following the terms *mapping* and *query* are interchangeable.

3.1 Atomic mapping

We define an atomic mapping *Am* as a restricted XQuery expression: One for, one return and one optional where clauses.

Am::=for *V* in *SP* (where *cond*)? Return ()|*constant*|*sp*|⟨*a*⟩⟨/*a*⟩

sp::=(doc(*constant*)|$*v*)((/|//)*constant*)*

cond::=*sp* θ (*sp*|*constant*)|*cond* and *cond*

The symbol *a* denotes XML tags, θ comparison predicates, *sp* a path query, and *cond* conditional expressions. We distinguish two kinds of equivalence comparison operators: $=_n$ and $=_v$, which compare the identities and values of two operands, respectively. For brevity, we write a single clause "*V* in *SP*" instead of "$\$v_1$ in sp_1, $\$v_2$ in sp_2,…". All variables used should be defined in the same atomic mapping for the query to be safe. In terms of the return clause, the atomic mapping is referred to as *empty*, *constant*, *copy*, or *constructor* type. An extension is straightforward to include computing expressions about *sp* (e.g., sp_1*sp_2, etc) in the condition *cond* and the return clause. In the sequel we use the notations *Vars*(*Am*) to denote the variables defined in *Am*, and *sp*($\$v$) to explicitly declare that *sp* is relative to the variable $\$v$.

Taking an XML tree *D* conforming to the source schema, an atomic mapping *Am* computes a sequence of new data trees. Let b={$\$v_1$:$t_1$,$\$v_2$:$t_2$,…} denote a tuple of bindings of the variables defined in the for clause, where t_i corresponds to a node bound to $\$v_i$. For each binding tuple *b* satisfying the condition *cond*, *Am* returns a data tree *d*.

Corresponding to the type of *Am*, the data tree *d* may be an empty node, a text node, a copied sub-tree of *D*, or a tagged node.

3.2 Correlation

Let Am_1 and Am_2 be the atomic mappings. We assume the prefix variables such as $\$n_2$ (possibly renamed) defined in Am_{title_1} have been defined explicitly in the atomic mappings (if not, they can be introduced dynamically and change no semantics of the atomic mappings). A correlation is a pair of (*cop*,*cpath*), where *cop* is one of the *Nest*, *Join* and *Merge* operators, and *cpath* is a *combination path* defined as:

Definition 1. A combination path between Am_1 and Am_2 is a conjunction of the conditional items $sp_1(\$v_1)\theta\ sp_2(\$v_2)$, where $\$v_1 \in Vars(Am_1)$ and $\$v_2 \in Vars(Am_2)$.

We also say that *cpath* is defined over the variables set $\{Vars(Am_1) \cup Vars(Am_2)\}$. Different from mapping (or query) *composition*[12], where one query can be answered directly using the results of another query, correlating two mappings is a "parallel" type of connection, which respectively combines the heads (i.e., the return clause) and the bodies (i.e., the for and where clauses) of the mappings. Informally, applying *Nest* or *Join* correlation between Am_1 and Am_2, the instances computed by Am_2 are nested within those by Am_1, while applying *Merge* the returned instances are merged. Moreover, the *Nest* and *Join* correlations restrict Am_1 and Am_2 to be *nesting compatible*, while the *Merge* correlation restricts them to be *merging compatible*.

Definition 2. Given atomic mappings Am_1 and Am_2. If Am_1 is empty or constructor type then it is nesting compatible with any type of Am_2; if (i) Am_1 (or Am_2) is empty type and Am_2 (or Am_1) is any type, or (ii) both Am_1 and Am_2 are constructor types with the same tag name *a*, then Am_1 is merging compatible with Am_2.

Definition 3. Given a pair of compatible atomic mappings Am_1 and Am_2. Let b_1 and b_2 respectively denote their variable binding tuples over some input data, and d_1 and d_2 be the corresponding computed data trees. By applying a correlation between Am_1 and Am_2, a new data tree d_3 is derived from d_1 and d_2. Semantically, with the data tree d_3, we define the following correlations, respectively.

- (***Nest***,*cpath*) indicates that for each b_1 that satisfies $cond_1$, if there are *i*, $i \geq 0$ b_2 (denoted by $b_{2,i}$) satisfying $cond_2$ and *cpath*, then d_3 is produced by appending all $d_{2,i}$ (corresponding to $b_{2,i}$) as the children of the root of d_1.
- (***Join***,*cpath*) is similar to (***Nest***,*cpath*), except that only when $i \geq 1$, d_3 is produced.
- (***Merge***,*cpath*) indicates that for each pair of (b_1,b_2) that satisfies $cond_1$, $cond_2$ and *cpath*, there is a d_3 produced that unites d_1 and d_2 by merging their roots.

Note that when merging an empty node and a text/tagged node, the result is still the text/tagged node. Intuitively, the *Nest* correlation captures an outer join relationship between b_1 and b_2, that is, the data tree d_3 is produced only if the corresponding d_1 exists. In contrast, the *Join* correlation specifies a join relationship, which is useful to filter out those undesired d_1s computed by Am_1.

Example 1. Consider the atomic mappings and correlations shown in Fig.2. The *Join* correlation "(2)" declares a value comparison between novel/author1 and author2/name: Only when it holds, is a new book node returned; otherwise, the book node is filtered out. To some extent, the *Join* correlation represents a conditional nesting: A branch (e.g., book-author) is formed by nesting if certain conditions are satisfied; otherwise the entire branch is dropped, comprising the parent. The *Merge* correlation is used to declare constraints over the same type of data nodes. For another example, consider the merging compatible atomic mappings $Am_{book()\_1}$ and $Am_{book()\_3}$ in Fig.2. They both compute new book nodes, but a *Merge* correlation (*Merge*,$\$n_1$/title=$_v\$b_1$/title) can rule that only when a source novel has a rating greater than 3, could a new book node be returned.

3.3 Macor tree

Regarding atomic mapping as a node, if Am_1 and Am_2 are correlated using the *Nest* or *Join* operator, then Am_2 is a child node of Am_1; if they are correlated using the *Merge* operator, then the nodes are united into one.

Definition 4. A Macor tree is a 4-tuple $\langle T,atoms,\lambda_e,\lambda_n\rangle$, where $T=(N,E)$ is an ordered tree, and *atoms*, λ_e and λ_n are functions defined as:

- ***atoms*** is a function that assigns i ($i\geq 1$) atomic mappings to each node $n\in N$, s.t. *each* pair of atomic mappings (Am_1,Am_2) assigned to the parent-child nodes is nesting compatible, and to the same node is merging compatible;
- λ_e is a function that assigns a correlation (*cop*,*cpath*) to each edge $e=(n_1,n_2)\in E$, s.t. *cop* is the *Nest* or *Join* operator and *cpath* is defined over $\{\$v|\$v\in Vars(m), m\in atoms(n_1)\cup atoms(n_2)\}$;
- λ_n is a partial function that assigns a correlation (*Merge*,*cpath*) to the node n where $|atoms(n)|>1$, s.t. *cpath* is defined over $\{\$v|\$v\in Vars(m),m\in atoms(n)\}$.

We say that a Macor tree *Mt* represents a mapping between the source schema S_S and the target schema S_T, which means that, given any instance conforming to S_S, the resulting instances, returned by *Mt* in the way as defined in Definition 3, always conform to the target schema S_T. Specifically, for each node n in *Mt*, *atoms*(n) jointly computes instances of the same schema element (say e_i, denoted by $I(e_i)$). Note that there may be multiple nodes in *Mt* contributing to $I(e_i)$, with the semantics of union. In light of the correlations that label the edges of *Mt*, the sets of instances $\{I(e_1),I(e_2),\ldots,I(e_m)\}$ (m is the number of nodes of *Mt*) are stitched up, denoted by $I(S_T)$. Let $I'(e_i)$ denote the instances of e_i in $I(S_T)$, then $I'(e_i)\subseteq I(e_i)$ holds, which indicates that not every instance in $I(e_i)$ will contribute to the glued target instances, e.g., some may be filtered out by the *Join* correlation.

Macor tree provides a flexible way to construct, rectify, and modifying the schema mappings. For example, we can go on nesting the corresponding atomic mappings into Mt_1 in Fig.2 and make it additionally compute other data nodes such as year, cate(gory), and review. Further, as it can be seen in Example 1, we can obtain a mapping that only transforms those novels with good ratings.

Previous work[2,13,14] has studied mapping mechanisms for dealing with overlapping and missing information. Their approaches can be applied to Macor: By appointing atomic mappings the role of *id-mappings*, Macor can be fledged with the function of object fusion; Additionally, *Skolem functions* can be used in the atomic mappings to represent “unknown” target values when there is no matching information in the sources.

4 Case Study

Consider our running example. We assume the Macor tree Mt_1 has been obtained in Fig.2, which states a transformation from novels into target books. Though such a mapping is semantic-valid, does it satisfy the application requirement? To answer this question, the user may decide first to transform it into an equivalent query expression (see next section), and then execute the query over some selected sample data sets[15]. The test may help the user find that in the transformation those novels with no corresponding author2 instances are lost. Suppose this is not the desired. Hence the user tries to make a modification over Mt_1 by changing the correlation “(2)” into a *Nest* one. Consequently, the originally lost novels can also be transformed into books, but with no information about authors. Further modifications are made again. This time the target author nodes are computed in terms of the element author1 in the source, i.e. $Am_{author()\_1}$ is changed into $Am_{author()\_2}$ (see Fig.2). This again leads to update of the related correlations and atomic mappings. Finally, an updated Macor tree Mt_2 is obtained, which transforms each novel into a book instance and associates possible author information with the book. The highlighted part in Mt_2 shows the modifications. Note that in Mt_2 the atomic mapping $Am_{name\_txt\_2}$ is modeled explicitly to compute name

values. This is because the matching schema elements (i.e., name of T and author1 of S_1) have different names.

As another example, we consider the case where the user (bookshop) only favors popular books, e.g., with good ratings. To code such requirements, the user models an atomic mapping $Am_{book()\_3}$ (see Fig.2) and tries to merge it with $Am_{book()\_1}$ in the Macor tree Mt_2. After executing the new resulting mapping, the user finds that such a constraint is too strong, and then makes a relaxation: if there is an author who has written at least one novel with a rating greater than 3, then all the novels written by him are popular. Driven by this requirement, the user first selects out those novels with qualified ratings, i.e., those qualified authors, by removing $Am_{book()\_3}$ from Mt_2, and merging it again with another atomic mapping $Am_{book()\_2}$, with the correlation "(7)" in Fig.2. Then, applying an extended *Join* correlation, $Join^+$ (see section 5), the sub-mapping is correlated with the atomic mapping $Am_{book()\_1}$ in Mt_2. A fragment of the refined schema mapping Mt_3 is shown in Fig.2. Specifically, by comparing the novel authors bound by $Am_{book()\_1}$ with the qualified authors, the correlation "(6)" constrains $Am_{book()\_1}$ to compute only the qualified books. Again, this example shows the ways to edit the schema mappings modeled with Macor. Notice that, while in terms of the constraints contained in the schemas, the mapping technologies presented in Refs.[5,6] can semi-automatically discover or preserve semantic-valid mappings, dealing with the above application-specific semantics would go beyond their abilities.

5 Expressiveness

In this section, we characterize the expressive power of Macor using a main fragment of XQuery, called *CoXQ*.

CoXQ::=⟨*a*⟩{*CoXQ*}⟨/*a*⟩|*CoXQ*, *CoXQ*

|for ($*v* in *sp*)+(where *cond_aggre*)? return *CoXQ*|*sp*|*constant*|()

cond_aggre::=*sp*θ(*sp*|*constant*)|*Agg*(*CoXQ*)θ*constan*|*cond_aggre* and *cond_aggre*

Here *cond_aggre* denotes a conditional item that additionally permits aggregate functions such as count() to compute over intermediate query results. Compared with XQuery, CoXQ rules out the abilities to bind variables to intermediate results. The following conclusion holds.

Proposition 1. CoXQ captures the expressiveness of Macor.

5.1 Translation from Macor into CoXQ

In light of the semantic definitions (see section 3.2) of the *Nest*, *Join*, and *Merge* correlations between two atomic mappings Am_1 and Am_2, we write the following syntactical correlation rules, respectively.

*C*1 for V_1 in SP_1 where $cond_1$ return ⟨*a*⟩{for V_2 in SP_2 where $cond_2$ and *cpath* return *atomic_item*}⟨/*a*⟩

*C*2 for V_1 in SP_1 where $cond_1$ and *count*(for V_2 in SP_2 where $cond_2$ and *cpath* return *atomic_item*)>0

return ⟨*a*⟩{for V_2 in SP_2 where $cond_2$ and *cpath* return *atomic_item* }⟨/*a*⟩

*C*3 for V_1 in SP_1 for V_2 in SP_2 where $cond_1$ and $cond_2$ and *cpath* return ⟨*a*⟩⟨/*a*⟩

Note that *atomic_item* represents the return items of the atomic mappings. Rule *C*1 nests Am_2 within the return clause of Am_1. Rule *C*2 is the same as *C*1 except that a condition *count*()>0 of joining Am_1 and Am_2 is introduced additionally. In rule *C*3, the for, where and return clauses of Am_1 and Am_2 are merged respectively. Following XQuery, checking the semantics of the above rules is trivial. For example, the rule *C*1 indicates that the values computed by the inner query will be nested within the tagged nodes *a*, if the corresponding conditions are satisfied. It should be noted that the syntactical rules are not unique, e.g., by introducing a let variable a simplified version of *C*2 would be obtained equivalently, since the let variable could be pushed into the where and return clauses[16].

Join$^+$. In terms of the correlation rule *C*2, we extend the *Join* correlation to a general form ($Join^+$,*cpath*,α,β), where α, substituting for *count*()>0, denotes an arbitrary condition of joining Am_1 and Am_2, and β, substituting for the special return item, i.e., Am_2 in *C*2, represents an expression over Am_2. As shown in the Macor tree Mt_3 in Fig.2,

when β is null, the correlation $Join^+$ serves as a constraint over the related atomic mapping. The discussion about the *Join* correlation also suits for $Join^+$.

Atomic mappings, and the correlations of two atomic mappings fall into CoXQ expressions. Due to the composability of XQuery, the correlated mappings can be further correlated with other mappings in the same way as *C*1, *C*2, or *C*3. This produces general CoXQ expressions. Based on the correlation rules, the CoXQ expression corresponding to a Macor tree *Mt* can be recursively written out by the procedure *toCoXQ* shown in Fig.3.

Procedure *toCoXQ*
Input: A Macor tree *Mt* with the root node *n*;
Output: The *CoXQ* expression *E* denoted by *Mt*.

E←Correlating the atomic mappings *atoms*(*n*) in the same way as *C*3
For each node $n_i \in childOf(n)$
  e←*toCoXQ*(n_i)
  Switch the correlation between *n* and n_i
  Case (*Nest*,*cpath*):*E*←Correlating *E* with *e* in sequence, in the same way as *C*1
  Case (*Join*,*cpath*):*E*←Correlating *E* with *e* as *C*2
  Return *E*

Fig.3 Procedure *toCoXQ* translates a Macor tree into a CoXQ expression

5.2 Translation from CoXQ into Macor

In the opposite direction, if an XQuery expression falls into CoXQ, then it can be transformed into a corresponding Macor tree. This is useful for those applications where the schema mappings have been directly expressed as XQuery queries. Generally, an XQuery expression is written in an abbreviated form. While bringing programming conveniences, this makes sub-mappings dependent on each other. To some extent, Macor uses redundancies to model unabbreviated mappings. To transform a CoXQ expression *E* into a Macor tree, we process in two stages: first, normalize the CoXQ expression *E*; then, break down the normalized expression *E*′.

Figure 4 shows the normalization rules. Different from those proposed in Ref.[16], where the normalization is to transform an XQuery expression into SQL queries, or minimize an expression for optimization, the rules in Fig.4 play a role of maximization in the sense that redundancies are introduced to localize the non-local variables (*R*1.1~*R*1.3) and complete the return-items (*R*2.1~*R*2.4) used in the CoXQ expression *E*. In Fig.4, the rules *N*1 and *N*2 define the notations *def* and *fwr*, respectively. Recall that the *sp*() denotes a path starting at the schema root, and *sp*($\$v$) a path relative to the variable $\$v$. The notation *def* rewrites the definition of a variable $\$v$ into an absolute path, and *fwr* maximizes a path query *sp* with a for-where-return query.

Based on the comparison of node id bound to the variables, additional conditions like $\$v' =_n \$v$ are used in the rules to keep the semantics of the query. In normalizing variable definitions, the rule *R*1.1 indicates that, if in the same for clause there are multiple variables (e.g., $\$v_2$ and $\$v_3$) relative to the same (e.g., $\$v_1$), then a redundant variable $\$v_1'$ is added such that there is a chance to divide *E* into two expressions with a *Merge* correlation. The rule *R*1.2 localizes the variable $\$v_1$ by its definition, i.e., *def*($\$v_1$). In the rule *R*1.3, the innermost query block references the variable $\$v_1$ in the outermost. In this case, to translate *E* into the Macor tree, the rule *R*1.3 fills the middle block with a copy of the crossing-block variable $\$v_1$. Further, the rules *R*2.1~*R*2.4 normalize the result pattern in the expression *E*. For example, the rule *R*2.1 substitute the path query *sp* with a complete for-where-return expression, i.e. *fwr*(*sp*), such that atomic mappings can be derived from *E* with a *Nest* (or *Join*) correlation. The other rules have similar functions.

In the second phase, the normalized expression *E*′ is decorrelated into atomic mappings and correlations, by

inversely applying the correlation rules $C1$~$C3$. For the case shown in $R2.1$, for example, an empty atomic mapping, together with $fwr(sp_1)$ and $fwr(sp_2)$ are produced. This decorrelation is straightforward, and is omitted here. Note that the rules in Fig.4 use a special (fragment of) expression to illustrate the normalization. The extension to general queries is obvious.

0. Notations

N1 ***def*($v)**. If $v is defined with the form of
(1) $v **in** *sp*(), then *def*($v)=*sp*();
(2) $v **in** $v'(/|//)*path*, then *def*($v)=*def*($v')(/|//)*path*.

N2 ***fwr*(*sp*)**.
(1) If *sp* starts at a document root, then *fwr*(*sp*) is defined as
for $v' **in** *sp*() **return** $v'
(2) If *sp* starts at a variable $v, then *fwr*(*sp*) is defined as
for $v' **in** *def*($v) **where** $v' =*n* $v **return** *sp*($v')

1. Variable normalization

R1.1 **for** v_1 **in** sp_1(), v_2 **in** sp_2(v_1), v_3 **in** sp_3(v_1) →
for v_1 **in** sp_1(), v_2 **in** sp_2(v_1) **for** $v'_1 **in** sp_1(), v_3 **in** sp_3($v'_1) **where** v_1 =*n* $v'_1

R1.2 **for** v_2 **in** sp_1(v_1) → **for** $v'_1 **in** *def*($v$_1), v_2 **in** sp_1($v'_1) **where** $v'_1 =*n* v_1

R1.3 **for** v_1 **in** sp_1 **return** ⟨a_1⟩ { **for** v_2 **in** sp_2 **return** ⟨a_2⟩
{**for** v_3 **in** sp_3(v_1) **return** …}⟨/a_2⟩}⟨/a_1⟩ →
for v_1 **in** sp_1 **return** ⟨a_1⟩{**for** v_2 **in** sp_2 **for** $v'_1 **in** sp_1 **where** $v'_1 =*n* v_1 **return** ⟨a_2⟩
{**for** v_3 **in** sp_3($v'_1) **return** …}⟨/$a_2$⟩}⟨/$a_1$⟩

2. Expression normalization

R2.1 **return** ⟨a⟩{sp}⟨/a⟩ → **return** ⟨a⟩{*fwr*(sp)} ⟨/a⟩

R2.2 **return** (sp_1, sp_2) → **return** (*fwr*(sp_1), *fwr*(sp_2))

R2.3 **for** v_1 **in** sp_1 **return** ⟨a_1⟩⟨a_2⟩⟨/a_2⟩⟨/a_1⟩ →
for v_1 **in** sp_1 **return** ⟨a_1⟩{**for** $v'_1 **in** sp_1 **where** v_1 =*n* $v'_1 **return** ⟨$a_2$⟩⟨/$a_2$⟩}⟨/$a_1$⟩

R2.4 **for** v_1 **in** sp_1 **return** ⟨a_1⟩*constant*⟨/a_1⟩ →
for v_1 **in** sp_1 **return** ⟨a_1⟩{**for** $v'_1 **in** sp_1 **where** v_1 =*n* $v'_1 **return** *constant*}⟨/a_1⟩

Fig.4 The normalization rules

6 Implementation and Experiments

With Macor, the tasks such as constructing, maintaining and refining mappings are reduced to operations over the Macor trees. In this section we present how to combine Macor with the technique of *mapping discovery*[5,6] to assist the user in defining correlations between atomic mappings. Figure 5 shows a framework of implementing Macor. Specifically, when mappings have been represented as XQuery expressions, then a translator is employed to translate them into Macor trees. As shown in the above section, any CoXQ expression can be represented as a Macor tree. Further, those XQuery expressions with variables bound to intermediate query results can be captured by a composition sequence of Macor trees. On the other hand, given atomic mappings, the correlations and the Macor trees can be constructed under the assistance of a mapping discoverer. Notice that correspondences (i.e., matches[17]) between schema elements are useful to guide the construction of atomic mappings.

Data structures. In implementation, the atomic mapping is stored as a target schema element and a tree pattern over the source schema. As in Ref.[11], the tree pattern is associated with variables. The variable names are controlled under a global manager. A physical node in the Macor tree keeps an inner correlation and an outer correlation, which respectively correspond to the correlation of the atomic mappings in the same node, and the correlation between the atomic mappings in the node and the parent node. Besides the Macor tree, our implementation also employs another structure, called *quasi Macor tree*, which is a Macor tree where each edge (or node) may be labeled with multiple correlations. A quasi Macor tree can represent exponentially many Macor trees. As indicated in Ref.[6], a mapping discoverer is to search as many candidate semantic-valid mappings as it can, and

the user is responsible for determining the desired one(s). With quasi Macor tree, the discovered candidate mappings (may be numerous) can be represented via a few structures. We believe such an optimization would benefit both the user and the system.

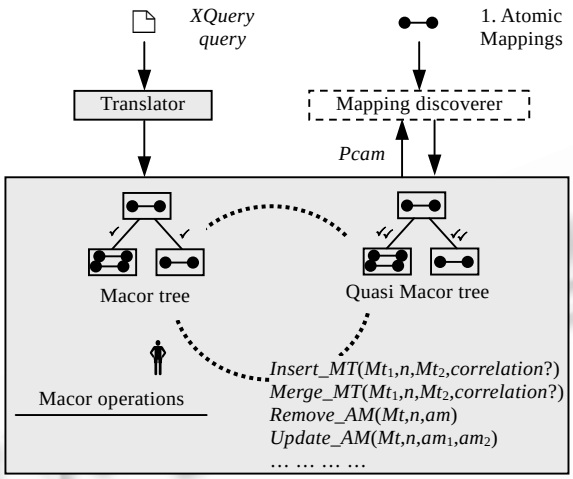
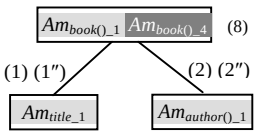


Fig.5 Implementing Macor

Macor operations. Macor operations (see Fig.5) are provided for the user to manipulate the Macor tree in a GUI or script way. The operation *Merge_MT*, for example, merges Mt_2 and Mt_1 by uniting the root node of Mt_2 and a special node n of Mt_1, i.e., the related atomic mappings are correlated with a *Merge* correlation. If the *correlation* is not assigned by the user, then the mapping discoverer is invoked to search candidate correlations among a set of potentially correlated atomic mappings (*pcam*). Taking the *pcam* as input, the discoverer searches all join paths in terms of the constraints in the schemas[6,7] and returns quasi Macor trees, which then will be specialized by choosing the desired correlations by the user.

We use an example to illustrate the principle. Consider our running example again. Assume for each author1 instance of novel there exists a corresponding name instance of author2, i.e., a referential constraint holds in $S1$. Given the Macor tree Mt_1 in Fig.2. To get a flattening target view, i.e., producing every pair of book and author instances, the shop manipulates Mt_1 by merging another Macor tree Mt_4 into it with a correlation "(8)" (see Fig.6), where Mt_4 contains only one atomic mapping $Am_{book()\_4}$. Such an operation triggers the discoverer with the $pcam$={($Am_{book()\_4}$, Am_{title_1}), ($Am_{book()\_4}$, $Am_{author()\_1}$)} and produces a quasi Macor tree as shown in Fig.6. The correlations "(1″)" and "(2″)" are suggested by the discoverer, whose principle is to search for all semantic paths (called *associations* in Refs.[6,7]) between the schema elements, e.g., "(1″)" is derived from a logical association between author2 and title in $S1$, the elements corresponding to the leaves of the tree patterns of the inputted pair of atomic mappings, i.e., $Am_{book()\_4}$ and Am_{title_1}. In this sense, Macor enables a mapping discoverer to work at the atomic mapping level, and thus provides automated support, to some extent, for dealing with application-specific mappings. The quasi Macor tree then is specialized by choosing the desired correlations, e.g., "(1)" and "(2″)". It is interesting to notice that the quasi Macor tree in Fig.6 states four significant mappings, though there exist subtle differences between their semantics.



Quasi Macor tree

Atomic mappings	$Am_{book()\_1}$: **for** $\$n_1$ **in** doc("S1")//novel **return** ⟨book/⟩
	$Am_{book()\_4}$: **for** $\$a_7$ **in** doc("S1")//author2 **return** ⟨book/⟩
	Am_{title_1}: **for** $\$n_2$ **in** doc("S1")//novel, $\$t_1$ **in** $\$n_2$/title **return** $\$t_1$
	$Am_{author()\_1}$: **for** $\$a_1$ **in** doc("S1")//author2 **return** ⟨author/⟩

Correlations	(8) (*Nest*,$\$n_1$/author1=$_v$$\$a_7$/name) (1) (*Nest*,$\$n_1$=$_n$$\$n_2$) (2) (*Nest*,$\$n_1$/author1=$_v$$\$a_1$/name) (1″) (*Nest*,$\$a_7$/name=$_v$$\$n_2$/author1) (2″) (*Nest*,$\$a_7$=$_n$$\$a_1$)

Fig.6 Example of quasi Macor tree

Experiments. The correctness of Macor is first verified. Our experiments are built on top of the XQEngine* for XQuery. We model some special Macor trees by combining different correlations, and execute the corresponding CoXQ expressions (queries) over elaborated synthetic datasets. The results indicate that the semantics of the Macor tree is correct. Further, by translating (or rewriting) the XML Query Use Case queries[18] into Macor trees, we compare the results computed respectively by the queries and the Macor trees. Just as expected, the experimental results declare no differences between the resulting data trees, except that the Macor trees may consume a bit more executing time, due to the redundancies that are not recognized by the optimizer in the query engine.

Also, we experiment on the maintainability of Macor, using a number of publicly available schemas that vary in terms of size and complexity. At the beginning, with Macor, we make an *identity mapping* for each schema, that is, the mapping between the same target and source schemas. Obviously, such a Macor tree has a shape of the target schema, and every node in it exactly contains one atomic mapping. Then we apply a random sequence of compound type changes[19] to each target schema (or source schema), and modify the corresponding Macor tree to keep the schema mapping semantic-valid. Table 1 presents the quantity of the atomic mappings modified (i.e., inserted, deleted, and updated) for each type change (the modifications on the correlations are roughly ignored). The schema size is shown in terms of schema elements in the corresponding schema trees. At the same time, the schema size also indicates the quantity of the atomic mappings in the original identity mappings. Table 1 reports partially the experimental results based on the schemas from XMark benchmark** and Mondial Database***. As it can be seen, the number of the modified atomic mappings is much smaller than the number of atomic mappings of the full mappings.

Table 1 Experimental results that show the quantity of modifications over the Macor trees, based on a scenario of schema evolution

Schema	Schema size	Schema change	Mapping modification
Auction	180	Group the continents	13
		Nest item within categories	7
		Encapsulate open_and closed_auction	9
Mondial	159	Categorize country	5
		Nest mountain within country	3
		Regroup sea, lake and river	10

In the same experiment setting we model the mappings as XQuery expressions and apply the same changes to the schemas. In essence the involved mapping modification in this case is the same as the modification in the above case where the mappings are modeled as Macor trees. However, since there are no clear delimiters to distinguish

* http://www.fatdog.com

** http://monetdb.cwi.nl/xml/

*** http://www.dbis.informatik.uni-goettingen.de/Mondial/

sub-mappings in XQuery expressions, the modification has to be located to the whole mapping, while Macor enables the mapping to be modified locally. At the same time, we notice that, even though some redundancies in the Macor tree may increase the number of modifications, for the mappings between real-data schemas, as shown in Table 1, such influence is very little on Macor's maintainability. Though the conclusion is derived from the experiments in terms of the scenario of schema evolution, the same is true for the scenarios like mapping refinement, as shown in Section 4.

7 Related Work

A number of techniques recently have been studied to provide automated support for dealing with mapping problems. Among those, schema matching[17] focuses on computing semantic correspondences between schema elements. Under an assumption that the desired matches have been given, Ref.[5,6] further make significant progress in discovering semantically valid schema-to-schema mappings. Yet, due to the lacking of a suitable mapping model, their work deploys the automation on the whole schemas, which limits its application scopes. Though a discoverer can search for semantic relations between the given element correspondences (or matches) and produce candidate sub-mappings, there are no ways to correlate them together to form a full mapping. The work in Ref.[7] studies how to locate matches affected by schema evolution and then adapts original mappings by employing a discoverer in the same way. By composing mappings, a more general method is proposed in Ref.[8] to adapt the mappings when schemas evolve. However, the requirements for modifying mappings are various. By exposing correlations in mapping languages, Macor makes it possible to employ a discoverer at the atomic mapping level and provides much flexibility in mapping construction. More importantly, with Macor, modification can be located to partial sub-mappings, no matter whether the schemas change or not.

To some significant extent, the atomic (or partial) mappings in Macor resemble subgoals in defining integrated views using datalog programs. With Skolem functions, XML query languages such as XML-QL[20] also allow for defining schema mappings in a piecemeal fashion. In contrast with such id-based mechanisms of gluing sub-mappings, Macor provides richer language facilities, i.e., correlations, which enable fine-grained sub-mappings and then fine-grained maintainability of the model. To facilitate the validation of the schema mappings, a recent work[21] also proposes, based on attribute grammars, to express a full mapping by sub-mappings (called grammars) defined for each schema element. However, in their work the specification of one sub-mapping is dependent on the specification of another, i.e., correlations are coded into the sub-mappings.

8 Conclusion

This work discussed a maintainable XQuery model, i.e., Macor, for representing nested schema mappings. With Macor, a schema mapping was modeled as a number of atomic ones related with the correlations. We presented a case study, and analyzed its expressiveness. Finally, issues were addressed in combining Macor with current semi-automatic mapping techniques. XQuery is a young language designed for querying XML, and we believe our work is also useful to explore its characteristics.

References:

[1] Lenzerini M. Data integration: A theoretical perspective. In: Popa L, ed. Proc. of the 2002 ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. New York: ACM Press, 2002. 233–246.

[2] Fagin R, Kolaitis PG, Miller RJ, Popa L. Data exchange: semantics and query answering. In: Calvanese D, Lenzerini M, Motwani R, eds. Proc. of the 9th Int'l Conf. on Database Theory. Berlin, Heidelberg: Springer-Verlag, 2003. 207–224.

[3] Bernstein PA. Applying model management to classical meta data problems. In: Proc. of the 1st Biennial Conf. on Innovative Data Systems Research (CIDR). 2003. http://www-db.cs.wisc.edu/cidr/cidr2003/program

[4] Melnik S, Rahm E, Bernstein PA. Rondo: A programming platform for generic model management. In: Halevy AY, Ives ZG, Doan A, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2003. 193−204.

[5] Miller R, Haas L, Hernández M. Schema mapping as query discovery. In: Abbadi AE, *et al.*, eds. Proc. of the 26th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2000. 77−88.

[6] Popa L, Velegrakis Y, Miller R, Hernandez MA, Fagin R. Translating web data. In: Bernstein PA, *et al.*, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2002.

[7] Velegrakis Y, Miller RJ, Popa L. Preserving mapping consistency under schema changes. The VLDB Journal, 2004,13(3): 274−293.

[8] Yu C, Popa L. Semantic adaptation of schema mappings when schemas evolve. In: Böhm K, *et al.*, eds. Proc. of the 31th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2005. 1006−1017.

[9] Sahuguet A. Everything you ever wanted to know about DTDs, but were afraid to ask. In: Suciu D, Vossen G, eds. Proc. of the 3rd ACM SIGMOD Workshop on the Web and Databases. Texas, 2000. 69−74. http://www.research.att.com/conf/webdb2000

[10] Qian G, Dong Y. Constructing maintainable semantic mappings in XQuery. In: Doan A, *et al*., eds. Proc. of the 8th ACM SIGMOD Workshop on the Web and Databases. Maryland, 2005. 121−126. http://webdb2005.uhasselt.be/program.html

[11] Deutsch A, Tannen V. Containment and integrity constraints for Xpath fragments. In: Lenzerini M, Nardi D, Nutt W, Suciu D, eds. Proc. of the 8th VLDB Workshop on Knowledge Representation meets Databases. Roma, 2001.

[12] Madhavan J, Halevy A. Composing mappings among data sources. In: Freytag JC, *et al*., eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2003. 572−583.

[13] Papakonstantinou Y, Abiteboul S, Garcia-Molina H. Object fusion in mediator systems. In: Vijayaraman TM, *et al*., eds. Proc. of the 22th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 1996. 413−424.

[14] Rahm E, Thor A, Aumueller D, Do H, Golovin N, Kirsten T. iFuice-Information fusion utilizing instance correspondences and peer mappings. In: Doan A, *et al*., eds. Proc. of the 8th ACM SIGMOD Workshop on the Web and Databases. Maryland, 2005. 7−12. http://webdb2005.uhasselt.be/program.html

[15] Yan L, Miller RJ, Hass LM, Fagin R. Data-Driven understanding and refinement of schema mappings. In: Aref WG, ed. Proc. of the 2001 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2001. 485−496.

[16] Manolescu I, Florescu D, Kossman D. Answering XML queries on heterogeneous data sources. In: Apers PMG, *et al*., eds. Proc. of the 27th Int'l Conf. on Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers, 2001. 241−250.

[17] Rahm E, Bernstein PA. A survey of approaches to automatic schema matching. The VLDB Journal, 2001,10(4):334−350.

[18] Chamberlin D, Fankhauser P, Florescu D, Marchiori M, Robie J. XML query use cases. W3C Working Draft, 2003. http://www.w3.org/TR/2003/WD-xquery-use-cases-20031112

[19] Lerner BS. A model for compound type changes encountered in schema evolution. ACM TODS, 2000,25(1):83−127.

[20] Deutsch A, Fernandez M, Florescu D, Levy A, Suciu D. A query language for XML. Computer Networks, 1999,31(11-16): 1155−1169.

[21] Fan W, Garofalakis M, Xiong M, Jia X. Composable XML integration grammars. In: Grossman D, Gravano L, Zhai C, Herzog O, Evans DA, eds. Proc. of the 2004 ACM CIKM Int'l Conf. on Information and Knowledge Management. New York: ACM Press, 2004. 2−11.

QIAN Gang was born in 1975. He is a Ph.D. candidate at the Southeast University. His current research areas are database and information system.

DONG Yi-Sheng was born in 1940. He is a professor and doctoral supervisor at the Southeast University. His current research areas are database and information system.