A Mapping Model for Transforming Nested XML Documents

Gang Qian,† and Yisheng Dong††,

Department of Computer Science and Engineering, Southeast University, Nanjing, China

Summary
XML has been extensively adopted in many modern data sharing applications. Following the relational setting, we might expect to directly use XQuery expressions to describe semantic mappings between schemas for transforming nested XML documents. Yet we observe that such an undecorated mapping representation is becoming one of the sources that complicate the ubiquitous tasks such as constructing, maintaining, and refining schema mappings. In this paper we propose a model, called *mapping & correlation* (*Macor*), to represent nested schema mappings. With Macor, a full mapping is modeled as a number of simple, partial atomic ones that are correlated with correlations. As a result, the full mapping can be modeled incrementally in a piecemeal fashion, and when refining or maintaining the mappings, Macor makes it possible to locate modifications to few atomic mappings and related correlations, and reuse other parts of the full mapping.

Key words:
Schema mapping, Data Transformation, XQuery, XML.

Introduction

Nowdays, there is a rapid growth of requirements for integrating, exchanging and transforming data stored in different autonomous sources. To achieve interoperability, modern data-sharing architectures use schema mappings to specify how data instances over one schema are transformed to data instances over another. In data integration, for example, schema mappings are used to unfold or rewrite a user query over the global mediated schema into sub-queries over the source schemas [9]; in data exchange, the mappings are used to translate data from the sources into the target databases [1]. To enable data sharing, the user or the system manager 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 mappings.

A number of tools have recently been developed to assist the user in such processes by increasing the abstraction level [2, 13], semi-automatically discovering mappings [15, 17, 3] or preserving their semantics as schemas evolve [22, 26]. However, 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 [21]. To motivate our work, we begin with the following situations mostly encountered in practice.

Mapping Refinement. Due to subtle semantics hidden in the schemas, the user often needs to continuously refine a mapping to obtain the final *desired* one [25]. On the other hand, each refinement essentially is a semantically fine tuning over the mappings, e.g., changing a join from an inner join to an outer join, modifying a correspondence between the target and source elements, or appending some sub-mappings, etc. Mapping expression provides no mechanism for facilitating the action of refinement. On the contrary, a locally semantic adjustment may lead to a drastically, and then tediously syntactic modification on the expression.

Schema Evolution. As application changes, schemas may change their structure and even semantics, then the initially obtained mappings have to be updated or adapted to the new contexts. Besides the tedious modification on the mappings, here another necessary task for the user (or tools, e.g., [22]) is to locate sub-mappings that may be affected potentially by the change of the schemas or the application requirements. Since there are no explicit sub-mappings in an expression (e.g., they may correlate each other), such task also is intractable in general, especially in dynamic environment like the web, where both the target and source schemas may change frequently.

Complex Application. Large, complex schemas now are becoming prevalent on the web. For example, many public DTDs have up to several hundred elements and several thousand attributes. An effective strategy for solving complex problems, e.g., creating mapping between such schemas, is to divide it into simple sub-problems and then conquer them sequentially. Yet most of the available mapping languages only provide id-based merging mechanism to stitch up partial mappings. We believe richer language facilities would make a more flexible division of the problem.

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

Manuscript received February 25, 2006.
Manuscript revised February 28 , 2006.

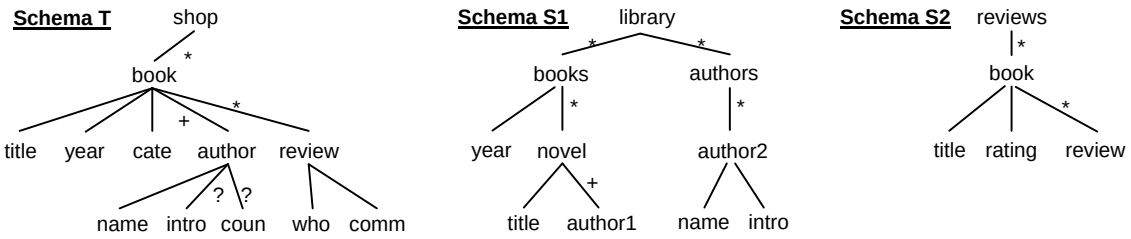


Figure 1. Example of nested schemas

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*. Except the explicit correlations, in a Macor tree there are no other relationships between the atomic mappings. 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.

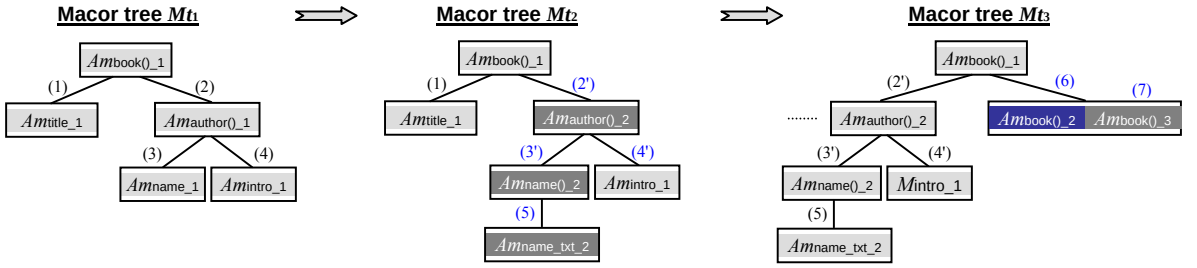
A preliminary work has been reported in [18]. In this paper we precisely define the Macor model formally and present a case study about the usage of Macor. The rest of the paper is organized as follows. An overview of Macor and its intuition are first given in Section 2. Then the detailed model is presented in Section 3. Section 4 presents a case study, and Section 5 discusses related work. Finally, Section 6 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, S1 and S2. The schema T depicts a user interface for accessing an online bookshop, and S1 and S2 indicate formats of the books and reviews information stored in different sources, respectively. For instance, in the source (of) S1, books are grouped by year, and then categorized by styles such as novel. 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, S1 and S2 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.

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 transformed instances should conform to the target schema S_T, as well as the application requirements, e.g., for the above scenario, the favorites of the bookshop. To illustrate the intuition behind Macor, 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*. Specifically, our discussions consider simplified XML schemas that have tree structures like in Figure 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 nesting structure, element sequence, element occurrence frequency, and referential relationship [5]. 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, one single schema element can be regarded as the simplest schema without any constraints. Instead of directly expressing a mapping through a unique complex transformation (e.g., an XQuery query), to model the full mapping between the schemas in Figure 1, Macor first models mappings between the source schemas (i.e., S1 and S2) 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.



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, $count()>0$, ε) (7) ($Merge$, $\$n_3$/title $=_v \$b_1$/title)

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

$Am_{book()\_1}$: **for** $\$n_1$ **in** doc("S1")//novel **return** <book/>
$Am_{title()\_1}$: **for** $\$n_2$ **in** doc("S1")//novel, $\$t_1$ **in** $\$n_2$/title **return** <title/>
$Am_{title\_txt\_1}$: **for** $\$t_2$ **in** doc("S1")//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. As will be seen in the sequel, the correlations can also be used to constrain data instances computed by the atomic mappings. This ability is useful to model application-specific requirements. 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. Figure 2 shows examples of Macor trees that model mappings for our scenario. Note that for legibility Figure 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.

As a result, the task of constructing the full mapping is to construct the corresponding Macor tree, which can be done step by step. Consider the Macor tree Mt_1. For example, one can fist correlate the atomic mappings $Am_{author()\_1}$ and Am_{name_1}, and then connect them to $Am_{book()\_1}$ and others. The correlations can be assigned by the user, or suggested by a discoverer like Clio [3]. After obtaining the Macor tree, how does the user know that it transforms instances as expected? Often, a schema only can depict partial semantics of the domain. Though, to some extent, a discoverer as presented in [15, 17] can semi-automatically suggest semantic-valid mappings, to get the desired, application-specific schema mappings, the user usually needs to further explore the semantics hidden in the schemas [25], and refine, execute (as motivated in [14]), and even debug the initially obtained mappings. With Macor, a complex mapping is partitioned into related simple atomic mappings. We believe such a piecemeal fashion would facilitate the tasks such as specifying, refining, and modifying schema mappings. After presenting the semantics of Macor, we shall provide a case study in Section 4 to illustrate the usage of the model.

3. Mapping Model

The goal of Macor is to provide mechanisms for modeling mappings and then obtaining maintainability. Having illustrated the intuition, 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-caluse. 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. First, a path query *sp* is evaluated at a *context node v* (i.e., either a document root or a node bound to $*v*) of *D*, and its result is the set of nodes of *D* reachable via *sp* from *v*, denoted by *v*□*sp*□. At a context node *v*, a unitary *cond* holds iff *v*□*sp*□ contains a node satisfying *cond*. Interpretation for binary condition expression is similar. Let $b=\{\$v_1: t_1, \$v_2: t_2, \ldots\}$ denote a tuple of bindings of the sequence of 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 $Am_{\text{title\_1}}$ 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 3.1 A **combination path** between Am_1 and Am_2 is a conjunction of the conditional items $sp_1(\$v_1)$ θ $sp_2(\$v_2)$, 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* (e.g., [11]), 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 3.2 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 3.3 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 $i \geq 1$, does 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 still is 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 3.4 Consider the atomic mappings and correlations shown in Figure 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.

On the other hand, the *Merge* correlation is used to declare constraints over the same type of data nodes. For another example, consider the merging compatible atomic mappings $Am_{\text{book()\_1}}$ and $Am_{\text{book()\_3}}$ 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 having a rating greater than 3, could a new book node be returned.

In light of the semantic definitions of the correlations between two atomic mappings Am_1 and Am_2, we write the following syntactical correlation rules, respectively.

C1
```
for V1 in SP1 where cond1 return <a>{
        for V2 in SP2 where cond2 and cpath
        return atomic_item
}</a>
```

C2
```
for V1 in SP1
where cond1 and count(
    for V2 in SP2 where cond2 and cpath
    return atomic_item
    )>0
return <a>{
        for V2 in SP2
        where cond2 and cpath return atomic_item
}</a>
```

C3
```
for V1 in SP1
for V2 in SP2
where cond1 and cond2 and cpath
return <a></a>
```

Note that *atomic_item* represents the return items of the atomic mappings. The rule **C1** nests Am_2 within the return clause of Am_1. Similarly, the rule **C2** nests Am_2 within Am_1, except that a condition count()>0 of joining Am_1 and Am_2 is introduced additionally. In the rule **C3**, the for, where and return clauses of Am_1 and Am_2 are merged respectively. Following XQuery [24], checking the semantics of the above rules is trivial. For example, the rule **C1** 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. Specifically, a simplified version of the rule **C2** may be obtained by introducing a let variable in XQuery, since it can be pushed into the where and return clauses [12], like the rule **C2**.

***Join*$^+$.** In terms of the correlation rule **C2**, 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 **C2**, represents an expression over Am_2. As shown in the Macor tree Mt_3 in Figure 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*$^+$.

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 3.5 A **Macor tree** is a 4-tuple $<T, atoms, \lambda_e, \lambda_n>$, 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 are nesting compatible, and to the same node are merging compatible;
- $\boldsymbol{\lambda_e}$ 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 $\{\$v \mid \$v \in Vars(m), m \in atoms(n_1) \cup atoms(n_2)\}$;
- $\boldsymbol{\lambda_n}$ is a partial function that assigns a correlation (*Merge*, *cpath*) to the node *n* where $|atoms(n)|>1$, s.t. *cpath* is defined over $\{\$v \mid \$v \in Vars(m), m \in atoms(n)\}$.

We say that a Macor tree *Mt* represents a mapping between the source schema SS and the target schema ST, which means that, given any instance conforming to SS, the resulting instances, returned by *Mt* in the way as defined in Definition 3.3, always conform to the target schema ST. Specifically, for each node *n* in *Mt*, *atoms*(*n*) jointly compute 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 $I(\text{ST})$. Let $I'(e_i)$ denote the instances of e_i in $I(\text{ST})$, 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.

Proposition 3.6 For a Macor tree *Mt*, the following properties hold:

1. In each ***inner*** (i.e., non-leaf) node of *Mt*, the contained atomic mappings are either empty or constructor types.
2. If a node of *Mt* contains constant or copy type of atomic mappings, then it must be a ***leaf*** node.
3. For each atomic mapping *Am* contained in a node *n* of *Mt*, there is only the possibility that *Am* was

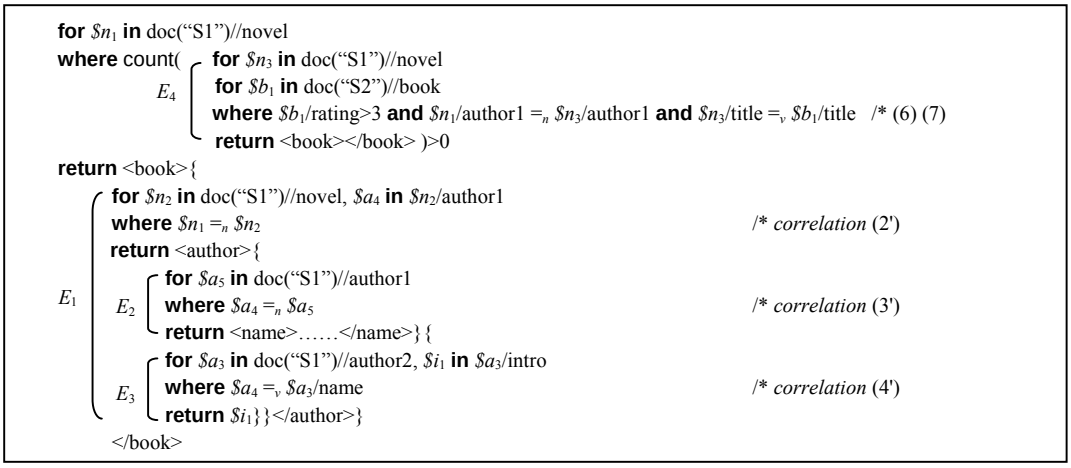


Figure 3. The fragment of the Macor tree Mt_3 in Figure 2 is translated into an expression.

correlated with those atomic mappings contained in the same, parent, or child nodes.

The above *properties 1* and *2* character a general Macor tree. Such trees provide a flexible way to construct, rectify, and modifying the schema mappings. For example, we can go on nesting corresponding atomic mappings into Mt_1 in Figure 2 and make it additionally compute other data nodes such as year, cate(gory), and review. Further, as can be seen in Example 3.4, we can obtain a mapping that only transforms those novels with good ratings. When an atomic mapping in a Macor tree changes, the correlations associated with it may need to be adjusted accordingly. Moreover, as indicated in [22], such changes could result in searching for new correlations to discover other candidate mappings. The above *property 3* reduces the search space to those atomic mappings contained in the same, parent, and children nodes, while other parts can be remained and reused.

Due to the composability of XQuery, the correlated mappings can be further correlated with other mappings, in the same way as **C1**, **C2**, or **C3**. Based on the correlation rules, the mapping expression corresponding to a Macor tree *Mt* can be recursively written out.

Procedure *toExp*
Input: A Macor tree *Mt* with the root node *n*
Output: The *mapping* expression *E* denoted by *Mt*

E ← Correlating the atomic mappings *atoms*(*n*) in the same way as **C3**
For each node $n_i \in childOf(n)$
  $e \leftarrow toMapping(n_i)$
  Switch the correlation between *n* and n_i
  Case (*Nest*, *cpath*):
    E ← Correlating *E* with *e* in sequence, in the same way as **C1**
  Case (*Join*, *cpath*):
    E ← Correlating *E* with *e* as **C2**
  Return *E*

Example 3.7 Taking the Macor tree Mt_3 in Figure 2, the procedure *toExp* outputs a query expression as shown in Figure 3. In terms of the order of the sibling nodes, the sub-expressions E_2 and E_3 are nested within E_1 in sequence. With an empty β, the $Join^+$ correlation "(7)" in then introduces a condition count()>0 over the sub-expression E_4, which serves as a condition to filter out those unsatisfied book instances computed by $M_{book()\_1}$, i.e., $I'(book) \subseteq I(book)$. Notice that in most cases the *Join* and $Join^+$ correlations can additionally be used to simulate conditions expressed by the some-clause in XQuery.

3.3 Other Facilities

For integrity, we briefly present in the following the facilities incorporated into Macor for dealing with overlapping, and missing information, though they are not the contributions of this paper.

Consider our scenario again. For each novel in the source S1, the Macor tree Mt_1 in Figure 2 will compute a new book instance. This is probably not the desired, since the same novel with several versions (e.g., different publishing years) will appear as multiple book entries, rather than as a single book in the target. To enable the desired behavior, Macor appoints $Am_{title()\_1}$ the role of *id-mapping* for mapping multiple binding instances to the same output: if two nodes created in the target have the same tag name and ID attribute, then they will fuse into one [16, 4, 8, 20]. This fusion process is repeated recursively over the combined elements.

The schema T in Figure 1 indicates that the coun(try) information associated with each author is optional. Thus the Macor tree Mt_1 in Figure 2 is semantic-valid and can always compute valid target instances. Yet if the country data of each author is required, the sources S1 and S2 provide no corresponding information. This case is usual

in practical data-sharing applications. To address this issue, Macor employees *Skolem functions*, a common technique extensively used in data integration and data exchange, to represent "unknown" values. In the following atomic mapping, for example, the Skolem term "$Sk(\$a_7/\text{name})$" computes a special value in terms of each author name.

$Am_{\text{coun\_txt\_1}}$: **for** $\$a_7$ **in** doc("S1")//author2 **return** $Sk(\$a_7/\text{name})$

4. Case Study

Due to subtle semantics hidden in the schemas, it is difficult for the user to directly specify a correct mapping that conforms to the target schema and also satisfies the application requirements. In this section, we show how Macor is able to simplify such tasks.

Consider our running example. We assume the Macor tree Mt_1 has been obtained in Figure 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 [25].

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. $Am_{\text{author()\_1}}$ is changed into $Am_{\text{author()\_2}}$ (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 $Am_{\text{name\_txt\_2}}$ is modeled explicitly to compute name values. This is because the matching schema elements (i.e., name of T and author1 of S1) have different names. Also, it is interesting to note that the highlighted subtree of Mt_2 can be modeled, modified, and tested independently, and then be inserted with the correlation "(2')". In contrast, for an undecorated representation, e.g., a unique complex XQuery expression, since there are no clear delimiters to distinguish sub-mappings, the modification has to be located to the whole mapping, though in essence it may be local.

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 $Am_{\text{book()\_3}}$ (see Figure 2) and tries to merge it with $Am_{\text{book()\_1}}$ 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 $Am_{\text{book()\_3}}$ from Mt_2, and merging it again with another atomic mapping $Am_{\text{book()\_2}}$, with the correlation "(7)" in Figure 2. Then, applying an extended *Join* correlation, $Join^+$, the sub-mapping is correlated with the atomic mapping $Am_{\text{book()\_1}}$ in Mt_2. A fragment of the refined schema mapping Mt_3 is shown in Figure 2. Specifically, by comparing the novel authors bound by $Am_{\text{book()\_1}}$ with the qualified authors, the correlation "(6)" constrains $Am_{\text{book()\_1}}$ 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 [17, 22] can semi-automatically discover or preserve semantic-valid mappings, dealing with the above application-specific semantics would go beyond their abilities.

5. Related Work

Schema mappings are extensively used in many modern applications such as the data integration and data exchange systems. A number of techniques recently have been studied to provide automated support for dealing with mapping problems. Among those, schema matching (e.g., [19, 6]) focuses on computing semantic correspondences between schema elements. Under an assumption that the desired matches have been given, [15, 17] further makes significant progress in discovering semantically valid schema-to-schema mappings. Yet, lacking of a suitable mapping model, [15, 17] deploy the automation on the whole schemas, which limits its application scopes. Though a discoverer can search for semantic relations between given element correspondences (or matches) and produce candidate sub-mappings, there are no ways to correlate them together to form a full mapping. [22] 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 [26] to adapt the mappings when schemas evolve. However, as shown in the introduction, 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.

In recent literature there are proposals that elevate schema mappings to first-class citizens. Within a general framework of model management, schemas and mappings between them can be manipulated via a set of operators [2, 13]. In their works mappings are modeled as an abstract syntactical model, and in essence the operators are defined for a set of correspondences between schema elements, which are useful in applications such as model translation. An extension in relational set-ting to executable mappings such as SQL queries has been made recently in a succeeding work [14]. Yet, manipulating mappings in these works is subject to schema (e.g., evolution), while Macor tree can be directly modified. Moreover, Macor makes it possible to reify the abstract mapping model into XML settings.

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 [4] 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 in [7] also proposes, based on attribute grammars, to express a full mapping by sub-mappings (called grammars) defined for each schema element. Yet, 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.

6. Conclusion

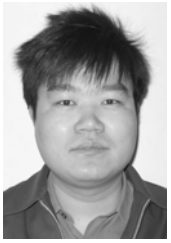
This work discussed a maintainable XQuery model, i.e., Macor, for representing nested schema mappings. With Macor, a schema mapping was modeled as a number of atomic ones related with the correlations. XQuery is a young language designed for querying XML, and we believe our work is also useful to explore its characteristics.

References

1. M. Arenas and L. Libkin. XML Data Exchange: Consistency and Query Answering. In *PODS 2005*, pages 13-24.
2. P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proc. of CIDR*, 2003.
3. Clio. http://www.cs.toronto.edu/db/clio/
4. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *proc. of* WWW, 1999.
5. A. Deutsch and V. Tannen. Containment and Integrity Constraints for XPath. In *KRDB*, 2001.
6. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *proc. of SIGMOD*, 2004.
7. W. Fan, M. Garofalakis, M. Xiong and X. Jia. Composable XML Integration Grammars. In *Proc. of CIKM*, 2004.
8. A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. In *proc. of* WWW, 2003.
9. M Lenzerini. Data Integration: A Theoretical Perspective. In *PODS 2002*.
10. B. S. Lerner. A Model for Compound Type Changes Encountered in Schema Evolution. *ACM TODS*, 25(1):83–127, March 2000.
11. J. Madhavan and A. Halevy. Composing mappings among data sources. In *Proc. of VLDB*, 2003.
12. I. Manolescu, D. Florescu, and D. Kossman. Answering XML Queries on Heterogeneous Data Sources. In *proc. of VLDB*, 2001.
13. S. Melnik, E. Rahm, P. A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *proc. of SIGMOD*, 2003.
14. S. Melnik, P. A. Bernstein, A. Halevy, E. Rahm. Supporting Executable Mappings in Model Management. In *proc. of SIGMOD*, 2005.
15. R. Miller, L. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proc. of VLDB*, 2000.
16. Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In *Proc. of VLDB*, 1996.
17. L. Popa, Y. Velegrakis, R Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proc. of VLDB*, 2002.
18. G. Qian and Y. Dong. Constructing Maintainable Semantic Mappings in XQuery. In *WebDB'05*, pages 121-126, 2005.
19. E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. The *VLDB Journal*, 10(4): 334-350, 2001.
20. E. Rahm, A. Thor, D. Aumueller, H-. Do, N. Golovin, T. Kirsten. iFuice – Information Fusion utilizing

Instance Correspondences and Peer Mappings. In *WebDB'05*.

21. A. Sahuguet. Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask. In *WebDB'00*, 2000.
22. Y. Velegrakis, R. J. Miller, and L. Popa. Preserving mapping consistency under schema changes. The *VLDB Journal*, 13(3): 274-293, 2004.
23. XQEngine. http://www.fatdog.com
24. XQuery. http://www.w3.org/XML/Query
25. L. Yan, R. J. Miller, L. M. Hass, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. In *proc. of SIGMOD*, pages 485-496, 2001.
26. C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. In *Proc. of VLDB*, 2005.



Gang Qian received the B.S. and M.S. degrees from Hohai University in 1995 and 2002, respectively, and is currently a PH.D. candidate in Southeast University.

His research interest includes database and information system.

Yisheng Dong was born in 1940, and is a professor at Southeast University.

His research interest includes database and information system.