The VLDB Journal (2004) 13: 274–293 / Digital Object Identifier (DOI) 10.1007/s00778-004-0136-2

Preserving mapping consistency under schema changes

Yannis Velegrakis[1], **Renée J. Miller**[1], **Lucian Popa**[2]

[1] University of Toronto, 40 St. George Str. Toronto, ON, Canada (e-mail: {velgias,miller}@cs.toronto.edu)
[2] IBM Almaden Research Center, 650 Harry Road, San Jose, CA, 95120, USA (e-mail: lucian@almaden.ibm.com)

Edited by M. Carey. Received: January 13, 2004 / Accepted: March 26, 2004
Published online: August 12, 2004 – © Springer-Verlag 2004

Abstract. In dynamic environments like the Web, data sources may change not only their data but also their schemas, their semantics, and their query capabilities. When a mapping is left inconsistent by a schema change, it has to be detected and updated. We present a novel framework and a tool (ToMAS) for automatically adapting (rewriting) mappings as schemas evolve. Our approach considers not only local changes to a schema but also changes that may affect and transform many components of a schema. Our algorithm detects mappings affected by structural or constraint changes and generates all the rewritings that are consistent with the semantics of the changed schemas. Our approach explicitly models mapping choices made by a user and maintains these choices, whenever possible, as the schemas and mappings evolve. When there is more than one candidate rewriting, the algorithm may rank them based on how close they are to the semantics of the existing mappings.

1 Introduction

A broad variety of data is available in distinct heterogeneous sources, stored under different formats: database formats (in relational and object-oriented models), document formats (SGML/XML), browser formats (HTML), message formats (EDI), etc. The integration, transformation, and translation of such data is increasingly important for modern information systems and e-commerce applications. Views, and more generally transformation specifications or *mappings*, provide the foundation for many data transformation applications.

A mapping specifies how data instances of one schema correspond to data instances of another. Mappings are often specified in a declarative, data-independent way (for example, as queries or view definitions). However, they necessarily depend on the schemas they relate. When these schemas change, the mappings must be updated or adapted to the new schemas. In this work, we consider the *adaptation* and management of mappings as schemas evolve.

To motivate our work, we first consider a number of applications and environments in which mappings are used extensively. Our discussion highlights not only the ubiquity of mappings in modern data management tasks but also the considerable effort that must be put into defining and verifying mappings and their semantics. We will argue that we can ill afford to re-create mappings from scratch as schemas change but should instead reuse previous mappings. Furthermore, mapping creation, although aided tremendously by modern tools that create mappings [31], still requires input from human experts. It is the semantic decisions input by these experts that we will especially try to manage and preserve in order to save the most precious administrative resource, human time.

Data integration. In data integration, a unified, virtual, view is used to query a set of heterogeneous data sources [18]. The process of creating this view is called schema (or view) integration. Numerous algorithms and tools have been proposed to automate or semi-automate schema integration ([33] and others). However, at its core, schema integration is a schema design problem. Some integration choices will necessarily be subjective and different users or designers may wish to make different choices or alter a heuristic choice made by a tool. Some tools anticipate this and for a limited set of alternative designs will still produce a correct mapping between the source schemas and the selected integrated schema [33]. Others will permit users to use a set of composable schema transformation operators to produce an integrated (transformed) schema (with a composed mapping) [11]. However, these approaches in general do not permit arbitrary changes to the integrated schema. Even a simple horizontal decomposition of an integrated table based on a user-defined predicate will typically require the designer to manually edit the mapping. Furthermore, changes in the source schema (even modest ones) are not supported. Such changes require the schema integration algorithm to be rerun.

Data exchange. In data exchange, mappings are used to transform an instance of a source schema into an instance of a different target schema [9]. The source and target schemas may be inconsistent, so for a given source instance there may be no target instance that represents the same information. While we have algorithms for detecting large classes of such inconsistencies, designers may wish to modify either the source or target schema to make them consistent. This may be done by cleaning inconsistent data in the source and adding a constraint to the source schema (or modifying its structure) or by modifying the target. Efficiently and effectively adapting a mapping to such constraint or structure modifications (in either the source or target) has not yet been considered.

Physical data design. Physical storage wizards, which permit the customization of physical schemas and storage structures, must maintain a mapping between the physical and logical schemas. A common example of such wizards are tools for customizing the relational storage of XML data [2]. Such tools evaluate (or help a designer to evaluate) the relative cost of different physical relational designs. However, they consider only a fixed set of physical schemas, each with a built-in mapping to the given logical (XML) schema. To permit a designer to suggest schema designs outside of this limited set, the tool would have to be able to adapt the XML to relational mapping to the ad hoc user-proposed schema change.

Other applications that rely on mappings include modeling of source descriptions [21], modeling of query capabilities [37], and view management [3,17]. In all of these applications, mappings provide the main vehicle for data sharing and data transformation. Yet, current solutions in these areas typically assume that the schemas are relatively static.

We advocate a novel framework that maintains the consistency of mappings under schema changes by finding rewritings that try to preserve as much as possible the semantics of the mappings. The semantics of a mapping is the relationship it establishes between instances of one schema and instances of another schema. This semantics is effected by the parts of the schema it uses (i.e., the elements, tables, attributes, etc.) and by the way it uses them (i.e., the join paths or selection conditions). We call this problem *mapping adaptation* to differentiate it from view adaptation [13], view synchronization [20], and view maintenance [39]

One way to approach this problem is to have a predefined finite set of interesting changes. Indeed, this is the approach used in several of the application areas that we have mentioned, including in physical design tools. For each such change, a modified mapping is stored ("hard-coded" if you will). The advantage of this approach is that we will know exactly how to handle each change. The disadvantage is that the way in which the schema can evolve is restricted to a set of predefined schemas, though if the set is rich enough, it may embrace all the possible schemas that are important for a specific application. A second alternative is to allow schemas to evolve and then find the changes that took place by comparing the modified schema (S') to the original version (S). For example, we could use a matching tool to find corresponding portions of the two schema versions [32] and then use a mapping creation tool to add semantics to these correspondences [31]. This will produce a mapping from S' to S that can be composed with the original mapping. Such an approach is complementary to the approach we consider here.

Our approach is to use a mapping adaptation tool in which a designer can change and evolve schemas. The tool detects mappings that are made inconsistent by a schema change and incrementally modifies the mappings in response. The term *incrementally* means that only the mappings and, more specifically, the parts of the mappings that are affected by a schema change are modified while the rest remain unaffected. This approach has the advantage that we can track semantic decisions made by a designer either in creating the mapping or in earlier modification decisions. These semantic decisions are needed because schemas are often ambiguous (or semantically impoverished) and may not contain sufficient information to make all mapping choices. We can then reuse these decisions when appropriate.

Our main contributions are the following.

1. We motivate the problem of adapting mappings to schema changes and we present a simple and powerful model for representing schema changes.
2. We consider changes not only to the structure of schemas (which may make the mapping syntactically incorrect [3]) but also to the schema semantics that may make mappings semantically incorrect.
3. We develop an algorithm for enumerating possible rewritings for mappings that have become invalid or inconsistent. The generated rewritings are consistent not only with the structure but also with the semantics of the schema.
4. We define a metric for the semantic similarity between two mappings that is used to rank the candidate rewritings.
5. We consider changes not only in the source schemas but also in the target. This is equivalent to adapting mappings to reflect changes in both their interface and the base schema.
6. We support changes not only on atomic elements but also on more complex structures including relational tables or complex (nested) XML structures.
7. We present a mapping adaptation algorithm that efficiently computes rewritings by exploiting knowledge about user decisions that is embodied in the existing mappings.

2 Related work

Schema evolution is a broad research area

that includes problems related to schema changes. It has been studied in different contexts and under different assumptions.

In *object-oriented database management systems (OODBMS)* the main problem studied is how to minimize the cost of updating the instance data when the schema has been modified. Banerjee et al. [4] give a taxonomy of the changes that may occur in OODBMS and provide an implementation for each one of them. Those changes are local to a single type, e.g., renaming an attribute or changing the position of a class in the class hierarchy. Lerner [19] extends the above work to include complex changes that span multiple classes and provides templates for the most common changes. None of this work investigates how views are affected when the schema is modified. *Incremental view maintenance* [7,28] is a related problem that deals with the methods for efficiently updating materialized views when the base schema data are updated. *View adaptation* [13,23] is a variant of *view maintenance* that investigates methods of keeping the data in a materialized view up to date in response to changes in the view definition itself. View adaptation may be required after mapping adaptation; hence we view this work as complementary to ours. View adaptation is a part of a broader problem called *view management* that includes any issue related to the creation and manipulation of views, e.g., reusing views to optimize query answering or data storage in cases of materialization [16]. View management is a long-standing problem in the commercial database system community where there is a need for detecting views (materialized or not) that become invalid due to schema changes in the base tables [3].

A different approach in schema evolution has been followed by McBrien and Poulovassilis [27], who combine schema evolution and schema integration in one unified framework. By using a series of primitive schema transformations one can map a local schema to a global schema. Each transformation must be accompanied by a query that describes its semantics. This query has to be manually specified by the user. Their approach enables easy composition of the transformations and permits optimization. In our approach, the user does not need to manually specify such queries. The EVE [20] system investigated the *view synchronization* problem, that is, how a view definition has to be updated when the base relational schema is modified. This work is very close to ours. However, in EVE, a user who defines a view is required to specify how the system should behave under changes. Furthermore, the supported changes are restricted to only deletion and renaming. Changes such as moving and copying attributes as well as constraint changes are not considered.

Our work can be seen within a general framework of model management in which schemas and views or mappings between them are considered and manipulated as first-class citizens. Schema matching [32] is a common first step that generates a set of syntactic correspondences between portions of two schemas. Notice that matchers do not create mappings that associate instances of schemas. However, schema mapping tools like Clio [25,31] can use these correspondences and (by using the semantics embedded in the schemas) generate mappings that associate instances. We take the mappings generated by a mapping tool or defined by a user and adapt them when schemas are changed in order to preserve the mapping consistency. Bernstein and Rahm [5] propose a different approach for dealing with schema changes. For each modified schema, they propose independently generating a mapping from the old version to the new one. This mapping is composed with the existing ones to generate the adapted mappings between the modified schemas [29]. This composition can be aided by the recent results reported in [24,10]. However, the resulting mappings need to be checked by an expert user to select only those that most accurately reflect the semantics of the transformation. Furthermore, it has been suggested in [24] and formally proven in [10] that the composition of conjunctive mappings cannot always be expressed as a conjunctive mapping. The advantage of our incremental approach is that it allows us to more easily detect the mappings that most accurately describe the transformation to the new version of the schema by reusing semantic choices made by the user that are embedded in the original mappings. Furthermore, our approach is guaranteed to always produce conjunctive rewritings of conjunctive mappings.

3 Mapping system

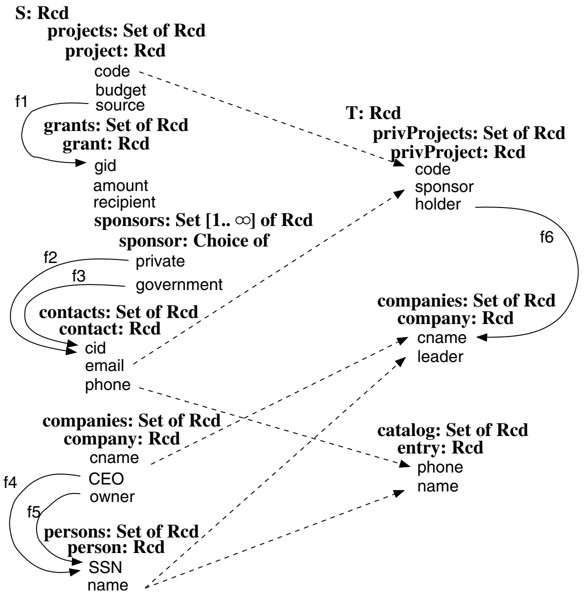
We consider a very general form of mapping that subsumes a large class of mappings used in a variety of applications. A *mapping* m from a schema $\mathcal{S}$ (called the *source* schema) to schema $\mathcal{T}$ (called the *target* schema) is an assertion of the form: $Q^S \rightsquigarrow Q^T$, where Q^S is a query over $\mathcal{S}$ and Q^T is a query over $\mathcal{T}$ [18]. Most commonly the queries are restricted to (type-compatible) queries that return sets of tuples and the relation $\rightsquigarrow$ is the subset-or-equals relation $\subseteq$; such mappings are called *sound* mappings [18]. Potential type incompatibilities can be resolved through type transformation functions. The queries Q^S and Q^T are conjunctive nested queries. They can be seen as tuple-generated dependencies from schema $\mathcal{S}$ to schema $\mathcal{T}$ [9]. Note that, although the queries are restricted to return sets of tuples, the schemas may be nested schemas and may contain complex or abstract types. This form of mapping is very general and includes as special cases the GAV (global-as-view) [25] and LAV (local-as-view) [21] views used in data integration systems or the GLAV (global-and-local-as-view) mappings used in transforming data between independent schemas [31], in peer-to-peer query answering [24], and in data exchange [9].

Other types of mappings include *complete* ($Q^S \supseteq Q^T$) and *exact* ($Q^S{=}Q^T$) mappings. Exact LAV mappings have been considered by Abiteboul and Duscha [1], while Grahne and Mendelzon [12] consider exact, sound, and also complete LAV mappings. These mappings have been considered mostly in the context of query answering. Since mapping adaptation does not perform any query evaluation over a database instance, the methods presented in this paper can be easily extended to include the case of exact or complete mappings. In addition to the nested conjunctive queries that we consider, there has been some work on query answering over mappings containing queries that include comparisons and negations [1] or recursion, which we do not consider here. Finally, there has been some work on query answering over mappings of the form $Q^S{\subseteq}Q^T$, where Q^S and Q^T may contain predicates ranging over both the source and target schemas [14]. However, this work is very preliminary and it is not clear that such general mappings can be used in practice. Understanding when such mappings are useful and how they can be maintained is a topic of our future work.

Definition 1 *A* **mapping system** *is a triple* $<\mathcal{S},\mathcal{T},\mathcal{M}>$ *where* $\mathcal{S}$ *and* $\mathcal{T}$ *are source and target schemas and* $\mathcal{M}$ *is a set of mappings between* $\mathcal{S}$ *and* $\mathcal{T}$.

Before defining mappings and schemas formally, we give an example to show how mappings may determine or constrain the placement of source data in the target.

Example 1 Consider the mapping system of Fig. 1. The schemas are shown in a nested relational representation that is used as a common data model. The specific model can support recursive data structures, allows efficient manipulation of the schemas and mappings, and has standard formal semantics. The left-hand schema S represents a source XML Schema with information about projects, grants, contacts, companies, and persons. Each project has a specific grant. Each grant has a *nonempty* set of sponsors that are either private individuals or government sponsors. Companies have an owner and a CEO. Relationships between different schema elements are specified via foreign keys (shown with solid lines in the figure). Foreign keys or in general referential constraints are considered part of the input. They can be found by looking at the schema definition, i.e., by querying the catalog tables of a relational schema or by referring to its DDL statements. Alternatively, foreign



```
m1: foreach S.projects p, S.grants g, S.contacts c,
            g.grant.sponsors n, n.sponsor.private r
      where p.project.source=g.grant.gid and
            r=c.contact.cid
     exists T.privProjects i, T.companies o
      where o.company.cname=i.privProject.holder
       with i.privProject.code=p.project.code and
            i.privProject.sponsor=c.contact.email

m2: foreach S.companies c, S.persons p
      where p.person.SSN=c.company.owner
     exists T.companies o
       with o.company.cname=c.company.cname and
            o.company.leader=p.person.name

m3: foreach S.contacts c, S.persons p
      where p.person.SSN=c.contact.cid
     exists T.catalog e
       with e.entry.name=p.person.name and
            e.entry.phone=c.contact.phone
```

Fig. 1. A mapping system with three mappings

keys may be specified by a user or discovered using a dependency miner [15]. The right-hand schema T is a relational schema that also contains information about projects and companies. However, it contains only projects with private funds and associates each project with the company in charge of the project. Three mappings (m_1, m_2, and m_3) have been defined from S to T. They are expressed in a nested relational representation (defined formally below) that can easily be transformed to other representations [31], e.g., XQuery. Each mapping has the form $Q^S \rightsquigarrow Q^T$. These mappings specify a containment assertion ($\subseteq$): for each tuple returned by Q^S, there must *exist* a corresponding tuple in Q^T. In our notation, the **foreach** clause defines Q^S while the **exists** clause defines Q^T. We use the **with** clause to make explicit how the source and target elements relate to each other. The "**foreach** A **exists** A'" part of the mapping is then a shorthand for the "**foreach** select * from A **exists** select* from A'". For example, mapping m_2 in Fig. 1 is a compact representation of:

```
m2: foreach
      select *
      from   S.companies c, S.persons p
      where  p.person.SSN=c.company.owner
    exists
      select *
      from   T.companies o
    with     o.company.cname=c.company.cname and
             o.company.leader=p.person.name
```

The precise meaning of the $*$ symbol in the select clause will be explained later but, informally, it represents all atomic elements contained in the elements specified in the from clause.

Mapping m_2 is a mapping that specifies how to populate the target with companies consisting of the company name from the source and the name of the company owner as the leader. Mapping m_1 is a mapping that populates the target with projects that have private sponsors. Note that the `holder` of a project is asserted to be equal to a company name `cname`. However, the **with** clause does not relate any source elements to either of those two target elements. So m_1 constrains the target but does not completely specify a target instance. The third mapping m_3 generates catalog entries in the target by joining persons and contacts in the source. □

Schemas and instances. We use a nested relational data model as a common platform to represent both relational and XML Schemas. The model is based on the well-studied relational model with extensions to support the nested structures and constraints that appear in XML Schemas [36]. Let Atomic represent a basic set of atomic data types such as String or Integer. A type τ is defined by the grammar: τ::= Atomic | Set of τ | Rcd$[l_1{:}\tau_1, \ldots, l_n{:}\tau_n]$ | Choice$[l_1{:}\tau_1, \ldots, l_n{:}\tau_n]$. A Set is a collection type. A value of type Set of τ is an unordered set of values of type τ. Types Rcd and Choice are complex types. A value of type Rcd$[l_1{:}\tau_1, \ldots, l_n{:}\tau_n]$ is an unordered tuple of pairs $[l_1{:}v_1, \ldots, l_n{:}v_n]$, where v_i is a value of type τ_i with $1 \leq i \leq n$. A value of type Choice$[l_1{:}\tau_1, \ldots, l_n{:}\tau_n]$, on the other hand, is a pair $l_i{:}v_i$, where v_i is a value of type τ_i with $1 \leq i \leq n$. The symbols $l_1 \ldots l_n$ are referred to as *labels* or *attributes*. With respect to XML Schema, we use Set to model repeatable elements (or repeatable groups of elements), while Rcd and Choice are used to represent the "all" and "choice" *model groups* [38], respectively. We do not consider order. A Set type represents an unordered set. An XML Schema "sequence" is modeled the same way "all" is modeled.

A *schema* is a set of labels (called roots), each with an associated type. For example, `S` and `T` in Fig. 1 are such roots for the source and target schema, respectively. An *instance* of a schema is a set of label-value pairs $l_i{:}v_i$ (at most one for each schema root), where value v_i is of the type τ_i associated to the schema root l_i.

For queries, we adopt a *select-from-where* syntax enhanced with choice type selections. An expression e is defined by the grammar $e ::= S|x|e.l$, where x is a variable, S a schema root, l a record label, and $e.l$ a record projection. The type of a schema root S is the type associated to label S. The type of expression $e.l$ is τ, where e must be an expression of type Rcd$[\ldots, l{:}\tau, \ldots]$. Queries have the following form, where e_i, c_i, and c'_i are expressions formed with bound variables in the <u>from</u> clause:

$$\begin{array}{ll} \underline{\textsf{select}} & e_0, e_1, ..., e_m \\ \underline{\textsf{from}} & P_0\ x_0, P_1\ x_1, ...\ P_n\ x_n \\ \underline{\textsf{where}} & c_0{=}c'_0\ \underline{\textsf{and}}\ c_1{=}c'_1\ \underline{\textsf{and}}\ ...\ \underline{\textsf{and}}\ c_k{=}c'_k \end{array}$$

Each P_i in the <u>from</u> clause is either an expression e with type Set of τ or an expression $e.l$, where e is an expression with a type Choice$[\ldots, l{:}\tau, \ldots]$ representing the selection of attribute l. In the former case, the variable x_i will bind to the individual values of type τ in the set. In the latter case, the variable x_i will bind to the values of type τ under the choice l of e. The query is well formed if the variable x_k (if any) used in P_i has $k{<}i$. The conditions in the <u>where</u> clause are optional. The '*' symbol can be used in the <u>select</u> clause to denote all possible valid atomic type expressions. The specific form of queries is generic and can represent the core of SQL and XQuery.

We will use queries to represent elements within schemas. A schema element is identified with the query that can be used to retrieve all the instance values of that element.

Definition 2 *A* **schema element** *is a query*

$$\underline{\textsf{select}}\ e_{n+1}\ \underline{\textsf{from}}\ P_0\ x_0,\ P_1\ x_1\ ,\ ...\ P_n\ x_n$$

where each P_k with $k{\geq}1$ uses variable x_{k-1}, P_0 starts at a schema root, and expression e_{n+1} uses variable x_n. If the <u>from</u> clause is empty, e_{n+1} starts at a schema root. When the details of the <u>from</u> clause are unimportant, the schema element can be noted as <u>select</u> e <u>from</u> P. A schema element may also be defined relative to another schema element <u>select</u> e' <u>from</u> P'. In that case, P_0 starts at expression e' instead of the schema root.

Example 2 For the source schema in Fig. 1, the schema elements `amount` and `private` under `grant` are formally defined, respectively, by the following two queries:

a_1: <u>select</u> g.`grant.amount` <u>from</u> `S.grants` g
a_2: <u>select</u> s <u>from</u> `S.grants` g, g.`grant.sponsors` n, n.`sponsor.private` s

Notice that since expression `S.grants` is of type Set of Rcd$[grant\ :\ Rcd[\ldots]]$, variable g in query a_1 is bound to the record type values of the set (i.e., Rcd$[grant\ :\ Rcd[\ldots]]$); hence, in order to get element `amount`, variable g has to be first record projected on `grant` and then on `amount`. On the other hand, since expression n.`sponsor.private` is a choice selection, variable s in query a_2 is bound to the atomic type values described by expression n.`sponsor.private`; hence, variable s is used in the <u>select</u> clause as is. □

For *schema constraints* we consider a very general form of referential constraints called *nested referential integrity constraints (NRIs)* [31] extended to support choice types. NRIs capture naturally relational foreign-key constraints as well as the more general XML Schema *keyref* constraints. The simplest form of NRI relates two schema elements and represents an inclusion constraint between them.

Definition 3 *Given two schema elements <u>select</u> e_1 <u>from</u> P_1 and <u>select</u> e_2 <u>from</u> P_2, both defined relative to schema element <u>select</u> e_0 <u>from</u> P_0, an* **NRI constraint** *is an expression of the form*

<u>foreach</u> P_0 *[***<u>foreach</u>** P_1 **<u>exists</u>** P_2*,* **<u>with</u>** C*]*

where C is the equality $e_1{=}e_2$. The element <u>select</u> e_0 <u>from</u> P_0 is referred to as the **context element** *of the constraint.*

We can naturally extend this definition to associate multiple elements. An NRI **<u>foreach</u>** X **<u>exists</u>** Y **<u>with</u>** C can associate a list X of n atomic elements with a list Y of n atomic elements. The **<u>with</u>** clause will be a conjunction of n equalities. Note that the elements of Y may be denoted relative to some variable of X.

Example 3 The foreign key f_2 on the source schema of Fig. 1 is expressed as follows:

f_2: **<u>foreach</u>** `S.grants` g, g.`grant.sponsors` n, n.`sponsor.private` r
<u>exists</u> `S.contacts` c
<u>with</u> c.`contact.cid`$=r$ □

4 Semantically valid mappings

When a schema changes, we need to rewrite the affected mappings. Our goal is to find rewritings that are consistent with the semantics of the new schema and with the current semantics of the mapping. To achieve the former (consistency with the new schema), we use an extension of the Clio mapping creation framework [31] in which mappings are created based on the semantics of the schemas. While Sect. 5 will give the algorithms necessary for adapting such mappings when schemas change, in this section we describe in detail the mappings that we consider.

We define the notion of *association* to describe a set of associated atomic type schema elements. Intuitively, an association is a query that returns all the atomic type elements mentioned in a query.

Definition 4 *An* **association** *is a query*

$$\begin{array}{l} \underline{\textsf{select}}\ *\ \underline{\textsf{from}}\ P_1\ x_1,\ P_2\ x_2,\ ...\ P_n\ x_n \\ \underline{\textsf{where}}\ e_1{=}e'_1\ \underline{\textsf{and}}\ e_2{=}e'_2\ \underline{\textsf{and}}\ ...\ \underline{\textsf{and}}\ e_n{=}e'_n \end{array}$$

Example 4 The following query defines an association containing the elements `private`, `gid`, `amount`, and `recipient` of Schema `S` in Fig. 1.

A_2: <u>select</u> * <u>from</u> `S.grants` g, g.`grant.sponsors` n, n.`sponsor.private` s □

Definition 5 *The* **union** *of two associations A and B (denoted as $A{\sqcup}B$) is an association with its <u>select</u>, <u>from</u>, and <u>where</u> clause consisting of the contents of the respective clauses of A and B.*

If C is a set of equalities $e{=}e'$, we will abuse the notation and we will use $A{\sqcup}C$ to denote the association A with the equalities in C appended in its <u>where</u> clause.

Mappings are very general interschema constraints between a source and a target association.

Definition 6 *A* **mapping** *is a constraint* **foreach** A^S **exists** A^T **with** *C, where* A^S *is an association on a source schema S,* A^T *is an association on a target schema T, and C is a nonempty conjunction of equalities relating atomic type expressions over* A^S *with atomic type expressions over* A^T.

While our techniques are designed to manage very general mappings of this form, we will make use of two important special classes of mappings. The first is the class of *correspondences* and the second is the class of semantically valid mappings considered in detail in the next section, in the context of schema evolution.

Correspondences are primitive forms of mappings used to describe how the value of an atomic target schema element is generated from a source schema element. The advantage of the correspondences is that they can be easily defined by a user [25] or they can be automatically generated by schema-matching tools [22].

Definition 7 *A* **correspondence** *from a source element* select e_S from P_S *to a target element* select e_T from P_T *is a mapping*

foreach P_S **exists** P_T **with** $e_S{=}e_T$.

Example 5 Correspondences are depicted in Fig. 1 with the dotted arrows between the schemas. The correspondence between the `name` in the source schema and the `leader` in the target is represented as

```
v: foreach S.persons e
   exists  T.companies c
   with    c.company.leader=e.person.name          □
```

To understand and reason about mappings and rewritings of mappings, we must understand (and be able to represent) relationships between associations. We use renamings (1-1 functions) to express a form of query subsumption between associations.

Definition 8 *An association A is* **dominated** *by association B (denoted* $A \dot{\preceq} B$*) if there is a renaming (1-1 function) h from the variables of A to the variables of B such that the* from *and* where *clauses of* $h(A)$ *are subsets, respectively, of the* from *and* where *clauses of B.*

Domination can naturally extend to mappings as follows. Mapping m_1: **foreach** A_1^S **exists** A_1^T **with** C_1 is dominated by mapping m_2: **foreach** A_2^S **exists** A_2^T **with** C_2 (denoted $m_1 \dot{\preceq} m_2$) if $A_1^S \dot{\preceq} A_2^S$, $A_1^T \dot{\preceq} A_2^T$ and for every equality $e{=}e'$ in C_1, $h_1(e) = h_2(e')$ is in C_2 (or implied by C_2), where h_1 and h_2 are the renaming functions from A_1^S to A_1^T and from A_2^S to A_2^T, respectively.

Definition 9 *A correspondence v:* **foreach** P_S **exists** P_T **with** *D* **is covered** *by the pair of associations* $<A^S, A^T>$ *if* $P_S \dot{\preceq} A^S$ *and* $P_T \dot{\preceq} A^T$. *The correspondence is also* **covered** *by mapping m if* $v \dot{\preceq} m$.

Given a mapping m between two schemas it is always possible to extract the correspondences that are covered by that mapping.

To test whether a schema element plays any role in a constraint, we need to check whether there is a renaming from the variables of the element to the variables of the constraint and whether the renaming of its select clause expression is used in any of the expressions of the constraint. Recall that mappings are constraints and correspondences are special forms of mappings.

```
P1^S : select * from S.projects p
P2^S : select * from S.grants g, g.grant.sponsors n,
                     n.sponsor.private r
P3^S : select * from S.grants g, g.grant.sponsors n,
                     n.sponsor.government m
P4^S : select * from S.contacts a
P5^S : select * from S.companies c
P6^S : select * from S.persons i

P1^T : select * from T.privProjects p
P2^T : select * from T.companies p
P3^T : select * from T.catalog c
```

Fig. 2. Source and target structural associations

Definition 10 *An element e:* select e_{n+1} from $P_0\ x_0$, $P_1\ x_1$, ..., $P_n\ x_n$ **is used (or participates)** *in an association A if there is a renaming function h from the variables of e to the variables of A and expression* $h(e_{n+1})$ *is used to form any of the expressions in the* from *or* where *clause of A. The element e is used in constraint F:* **foreach** *X* **exists** *Y* **with** *C if it is used in association* $X \sqcup Y \sqcup C$.

There are three ways in which semantic relationships between schema elements can be encoded. The first is through the structure of the schema. Elements may be related by their placement in the same record type or more generally through the nesting structure of the schema. This structure encodes semantic relationships between elements that the schema designer chose to explicitly model through the schema. An association containing elements that are related only through the schema structure is referred to as a *structural association*. Structural associations correspond to the primary paths used in [31], where it is shown that they can be computed by one time traversal over the schema.

Definition 11 *A* **structural association** *is an association* select * from $P_1\ x_1$, $P_2\ x_2$, ... $P_n\ x_n$ *with no* where *clause, expression* P_1 *starting at a schema root, and every expression* P_k, $k{>}0$ *starting with variable* x_{k-1}.

Semantic relationships within a schema may also be modeled using schema constraints. Chasing is a classical relational method [26] that can be used to assemble elements that are semantically related through constraints. A chase is a series of chase steps. A chase step of association R with an NRI F: **foreach** X **exists** Y **with** C can be applied if, by definition, the association R contains (a renaming of) X but does not satisfy the constraint. The result of the chase step is association R with the Y clause and the C conditions (under the respective renaming) added to it. The chase can be used to enumerate logical join paths, based on the set of dependencies in a schema. We use an extension of a nested chase [30] that can handle choice types and NRIs [35].

```
A1: select *
    from   S.projects p,S.grants g,g.grant.sponsors n,
           n.sponsor.private r, S.contacts c
    where  c.cid=r and g.gid=p.source
A2: select *
    from   S.projects p,S.grants g,g.grant.sponsors n,
           n.sponsor.government r, S.contacts c
    where  c.cid=r and g.gid=p.source
A3: select *
    from   S.grants g,S.contacts c,g.grant.sponsors n,
           n.sponsor.private r,
    where  c.cid=r
A4: select *
    from   S.grants g,S.contacts c,g.grant.sponsors n,
           n.sponsor.government r
    where  c.cid=r
A5: select * from S.contacts c
A6: select *
    from   S.companies c, S.persons p, S.persons w
    where  c.company.CEO=p.person.SSN and
           c.company.owner=w.person.SSN
A7: select * from S.persons c
A8: select *
    from   S.contacts c, S.persons p,
    where  p.contact.cid=p.person.SSN

B1: select *
    from   T.privProjects p, T.companies c
    where  p.privProject.holder=c.company.cname
B2: select * from T.companies c
B3: select * from T.catalog c
```

Fig. 3. Logical associations for schemas S and T

Definition 12 *A* **logical association** *R is the result*[1] *$chase_{\mathcal{X}}(P)$ of chasing an association P with the set $\mathcal{X}$ of all the NRIs of the schema.*

Example 6 The fact that name, CEO, and owner are all under the company element indicates that they are semantically associated since they all refer to the same company. These three elements form the structural association P_5^S of Fig. 2. This figure gives all the structural associations of the two schemas in Fig. 1. As another example of structural association, P_2^S and P_3^S represent the associations that exist between a grant and the two kinds of sponsors. The cardinality constraint on the sponsors (i.e., Set$[1\ldots\infty]$) has been used to infer that there cannot be a structural association containing grants but no sponsors. The foreign key f_5 on the source schema indicates that a person and a company can be associated through an owner relationship. Similarly, the foreign key f_4 indicates that they can also be related through the CEO. Chasing structural relation P_5^S with constraints f_5 and f_4 results in the logical association A_6 of Fig. 3. Logical associations A_1 to A_7 and B_1 to B_3 in the same Fig. 3 are the logical associations that resulted from the chase of the structural associations of Fig. 2 with all the constraints defined on the two schemas. □

The result of the chase of the structural associations represents semantic relationships between schema elements that are explicitly encoded in the schema by the schema designer. However, there may be hidden semantic relationships that were not encoded in the schema. Some of these relationships may be exposed by the set of mappings that are provided by the user or by a mapping tool. If an association used in a mapping is chased with the schema constraints and the result is not among the results of the chase of the structural associations of the schema, then that association provides some semantics that are not encoded in the schema and is referred as a *user association.*

Definition 13 *Let $\mathbb{S}$ be the set of structural associations of a schema S. An association A of schema S that is used in a mapping m is a* **user association** *if $chase_{\mathcal{F}}(A) \nsubseteq \{B \mid B{\in}chase_{\mathcal{F}}(X) \wedge X{\in}\mathbb{S}\}$, where $\mathcal{F}$ is the set of constraints in schema S.*

Example 7 Mapping m_3 joins contacts and persons based on the SSN and cid, which indicates that a person can be associated with contact information through the SSN. This gives the source schema association

```
A8: select *
    from S.contacts c, S.persons p,
    where c.contact.cid=p.person.SSN
```

Chasing the specific association with the set of schema constraints does not add any new elements. It can also be noticed that this association is not among the associations A_1 to A_7 of Fig. 3 that resulted from chasing the source schema structural associations. Thus, A_8 is a *user association* and the result of its chase (which is itself) is also added in the set of logical associations of Fig. 3. □

In a similar way, one can extract user associations from queries in query workload descriptions or queries used in view definitions.

For the remainder of the paper, we will consider as logical associations only those that have been generated by the chase of a structural or a user association.

We now give a formal definition of a semantically valid mapping.

Definition 14 *Let $\mathcal{S}$ and $\mathcal{T}$ be a pair of source and target schemas and $\mathcal{M}$ a set of mappings between them. Consider C to be the set of correspondences covered by the mappings in $\mathcal{M}$. A* **semantically valid mapping** *is a mapping* **foreach** *A^S* **exists** *A^T* **with** *D, where A^S and A^T are logical associations in the source and the target schema, respectively, and D is the conjunction of the conditions of the correspondences in C that are* covered *by the pair $<A^S, A^T>$ (provided that at least one such correspondence exists).*

Example 8 The two correspondences of the mapping m_2 are covered by the pair $<A_6, B_2>$ in two ways. Each one generated a different semantically valid mapping. The first gives

```
m_u: foreach S.companies c, S.persons p, S.persons p'
       where c.company.CEO=p.person.SSN and
             c.company.owner=p'.person.SSN
      exists T.companies o
      with   o.company.cname=c.company.cname and
             o.company.leader=p.person.name
```

[1] In general, the chase may produce multiple logical associations, in which case $chase_{\mathcal{X}}(P)$ is a set.

which after join minimization step becomes:

```
mo: foreach S.companies c, S.persons p
     where  c.company.CEO=p.person.SSN
    exists  T.companies o
    with    o.company.cname=c.company.cname and
            o.company.leader=p.person.name
```

The second way of covering the correspondences gives mapping m_2 of Fig. 1. The difference between m_o and m_2 is that m_o uses the CEO as the leader of the company while m_2 uses the owner. This example shows how our approach captures different join paths in the schema to produce semantically different mappings. □

We have to note here that the set of semantically valid mappings includes the mappings produced by the mapping generation tool Clio [31] with the addition of those based on user choices and those including choice types. This set will be our search space when looking for possible rewritings as a result of a schema evolution.

Definition 15 *Given a source and a target schema $\mathcal{S}$ and $\mathcal{T}$, along with a set of mappings $\mathcal{M}$ from $\mathcal{S}$ to $\mathcal{T}$, a* **mapping universe** $\mathcal{U}^{\mathcal{M}}_{\mathcal{S},\mathcal{T}}$ *is the set of all the semantically valid mappings.*

5 Handling schema evolution

Schemas usually evolve to adapt to new data requirements and semantics. When a schema changes, we need to rewrite the affected mappings in a way that is consistent with the semantics of the new schema and with the semantics of the existing mappings. To achieve the former we exploit information provided by the schema structure and semantics (constraints) by extending the algorithm presented in [31]. We provide algorithms to efficiently compute the schema semantics incrementally when a change to the schema structure or constraints occurs. For the latter, we present new techniques for modeling and reusing the semantics embedded within a mapping. When the semantics of a mapping must change, we make the minimum changes necessary to achieve a mapping that is consistent with the new schema.

Our algorithm accepts as arguments a mapping system, i.e, a pair of schemas $\mathcal{S}$, $\mathcal{T}$ and a set of mappings $\mathcal{M}$ from $\mathcal{S}$ to $\mathcal{T}$. It consists of two phases. The first is a preprocessing step in which the mappings are analyzed and turned into semantically valid mappings (if they are not). In particular, the set $\mathcal{C}$ of correspondences covered by the mappings in $\mathcal{M}$ is first extracted and then the mappings in $\mathcal{M}$ are analyzed. For each mapping $m \in \mathcal{M}$ of the form **foreach** A^S **exists** A^T **with** D, associations A^S and A^T are taken apart and are chased with the schema constraints to produce new associations $A^S_1, ..., A^S_n$ and $A^T_1, ..., A^T_l$, respectively. For each pair $<A^S_i, A^T_j>$ a new mapping m_{ij} of the form **foreach** A^S_i **exists** A^T_j **with** D' is created, where D' includes the conditions D, plus the conditions of all the correspondences that are covered by the pair of associations $<A^S_i, A^T_j>$ but were not covered by the pair $<A^S, A^T>$. This is because associations A^S and A^T are always dominated by associations A^S_i and A^T_j, respectively. Note that the set of all those semantically valid mappings m_{ij} is a subset of the mapping universe $\mathcal{U}^{\mathcal{M}}_{\mathcal{S},\mathcal{T}}$. Any existing user association is also identified during this process.

The second phase of the algorithm takes the set of semantically valid mappings generated during the first phase and maintains them through schema changes. In particular, for each kind of change that occurs in the source or the target schema, each mapping is modified as appropriate. This is done for each mapping in turn. Note that mappings generated in the first phase are potentially more complete than those entered by a user. Hence, we use the generated mappings within the adaptation algorithm since they extend the user mappings with the semantics embedded in the schema structure and constraints. In the following subsections, we present the algorithms that adapt the mappings for each kind of change that may occur on the schemas. Each algorithm accepts as input a set of semantically valid mappings $\mathcal{M}$ and returns the set of adapted semantically valid mappings $\mathcal{M}'$. We have identified a number of primitive schema changes and categorized them into three categories. The first one contains operations that change the schema semantics by adding or removing constraints, the second includes modifications to the schema structure by adding or removing elements, while the third category includes changes that reshape the schema structure by moving, copying, or renaming elements. To make the presentation less verbose, we will often assume that the schema changes occur in the source schema. However, the algorithms apply equally in the case where the changes occur in the target schema.

5.1 Constraint modifications

Adding constraints: Adding a new constraint on a schema does not make any of the existing mappings invalid, i.e., syntactically incorrect. However, it may make some of the mappings inconsistent, in the sense that they will no longer reflect the semantics of the schema. More precisely, a mapping may fall out of the mapping universe (recall Definition 15) as a result of adding a constraint. Let $<\mathcal{S}, \mathcal{T}, \mathcal{M}>$ be a mapping system and $\mathcal{C}$ be the set of correspondences dominated by all the mappings in $\mathcal{M}$ (which can always be extracted from the set of mappings $\mathcal{M}$ as mentioned in previous section). Assume that a new constraint F: **foreach** X **exists** Y **with** C is added in the source schema.

We first detect the mappings that are affected by the change, that is, mappings that are not semantically valid any more according to the new constraint.

A mapping m: **foreach** A^S **exists** A^T **with** D, with $m \in \mathcal{M}$ of a mapping system $<\mathcal{S}, \mathcal{T}, \mathcal{M}>$, needs to be adapted after the addition of a source constraint **foreach** X **exists** Y **with** C if X is dominated by A^S ($X \dot{\preceq} A^S$), with a renaming h, but there is no extension of h to a renaming from $Y \sqcup C$ to A^S. In other words, the addition of the new constraint caused A^S not to be closed under the chase, i.e., A^S is no longer a logical association.

If mapping m: **foreach** A^S **exists** A^T **with** D, with $m \in \mathcal{M}$ of the mapping system $<\mathcal{S}, \mathcal{T}, \mathcal{M}>$, needs to be adapted, the association A^S is chased with the set of the old schema constraints enhanced with the new constraint F. Note that it is not enough to chase only with F. After applying a chase step with F, additional chasing may be possible with existing constraints. The result is a set of new logical associations. For each such association A, a new mapping is generated in the form m_c: **foreach** A **exists** A^T **with** D'. The set D' consists of the conditions derived from the **with** clause of the correspondences in $\mathcal{C}$ that are covered by the pair $<A, A^T>$. Since A^S is always dominated by A, naturally, $D \subseteq D'$. Each mapping m_c generated by the above procedure, for which $m \dot{\preceq} m_c$, is added to $\mathcal{M}$ and mapping m is removed. Algorithm 1 gives a brief description of the steps taken when a new constraint is added.

```
Algorithm 1 – Constraint addition
Input:
  Set of mappings $\mathcal{M}$ from schema $\mathcal{S}$ to schema $\mathcal{T}$
  New constraint $F$: foreach $X$ exists $Y$ with $C$ on $\mathcal{S}$
Body:
  $\mathcal{X} \leftarrow$ constraints in $\mathcal{S}$,   $\mathcal{M}' \leftarrow \emptyset$
  $\mathcal{C} \leftarrow$ compute correspondences from $\mathcal{M}$
  For every m←(foreach $A^S$ exists $A^T$ with $D$) $\in \mathcal{M}$
      if ($X \dot{\preceq} A^S$ with renaming $h$ and $h(Y \sqcup C) \dot{\not\preceq} A^S$)
        For every $A \in chase_{\mathcal{X} \cup \{F\}}(A^S)$
            For every coverage of $<A, A^T>$ by $D' \subseteq \mathcal{C}$
                $m_r \leftarrow$ foreach $A$ exists $A^T$ with $D'$
                if ($m \dot{\preceq} m_r$)  $\mathcal{M}' \leftarrow \mathcal{M}' \cup \{m_r\}$
      else  $\mathcal{M}' \leftarrow \mathcal{M}' \cup \{m\}$
Output:
  New set of mappings $\mathcal{M}'$
```

Example 9 Assume that the following new constraint is added in the source schema of Fig. 1:

```
f7: foreach S.grants g
    exists  S.companies c
    with    c.company.cname = g.grant.recipient
```

allowing each grant to specify the company that receives the grant. Mappings m_2 and m_3 will not be affected since the **foreach** part of the constraint is not dominated by the **foreach** part of those mappings. Indeed, the fact that we can now determine the company that receives each grant has nothing to do with those two mappings since they deal with companies and persons only. On the flip side, this change greatly affects mapping m_1. Remember that the specific mapping was populating the target schema with projects receiving private funds and the associated companies, but the information of what company is related to each project was not available in the source schema. After the addition of the new constraint, this information becomes available, so mapping m_1 may need to be adapted to the new schema semantics. We detect this by verifying that the **foreach** clause of the constraint is dominated by (contained in) association A_1 used in mapping m_1 through a renaming h, but there is no extension of h such that the union of the **exists** and **with** clauses of the mapping is also dominated by association A_1 through that extension. When chased with the new set of constraints that includes F, association A_1 gives a new logical association:

```
A1a: select *
     from   S.projects p, S.grants g, g.grant.sponsors n,
            n.sponsor.private r, S.contacts c,
            S.companies o, S.persons e, S.persons e'
     where  p.project.source=g.grant.gid and
            r=c.contact.cid and
            g.grant.recipient=o.company.cname and
            o.company.CEO=e.person.SSN and
            o.company.owner=e'.person.SSN
```

This association generates, in turn, two rewritings for m_1, depending on how the `leader` element in the target is mapped: as the name of the `CEO` of a company or as the name of the `owner` of the company. (In Algorithm 1 terms, we say that there are two ways to cover the correspondence from person name to company leader by the pair $<A_{1a}, B_1>$). The first rewriting (after join minimization[2]) is

```
m'1a: foreach S.projects p, S.grants g, g.grant.sponsors n,
              n.sponsor.private r, S.contacts c,
              S.companies o, S.persons e
        where p.project.source=g.grant.gid and
              r=c.contact.cid and
              g.grant.recipient=o.company.cname and
              o.company.CEO=e.person.SSN
       exists T.privProjects j, T.companies m
        where j.privProject.holder=m.company.cname
       with   m.company.cname=o.company.cname and
              m.company.leader=e.person.name and
              j.privProject.code=p.project.code and
              j.privProject.sponsor=c.contact.email
```

while the second mapping m'_{1b} is the same as m'_{1a} except for the last condition in the first **foreach** where clause that is o.`company.owner`=e.`person.SSN` instead of o.`company.CEO`=e.`person.SSN`. This means that the first mapping populates the target schema with private projects and companies, using the CEO as the leader of the company while the second uses the owner. □

As the above example indicates, mapping adaptation may produce rewritings with different (maybe conflicting) semantics. In the absence of any additional information, it may not be possible to choose one mapping rewriting in favor of another. All the generated rewritings are consistent with the new schema and are as semantically close as possible to previously defined mappings (i.e., they are valid members of the new mapping universe). To accept or to reject some of them requires human intervention. In Sect. 6 we describe a methodology for ranking the generated rewritings based on their semantic closeness to the previous mappings. This ranking can be used to assist the user in such a decision.

A special, yet interesting, case is when the chase does not introduce any new schema elements in the association but only extra conditions. These conditions indicate new ways (join paths actually) to relate the elements in the association. This enhances the mapping universe with new semantically valid mappings; however, none of the existing mappings is adapted. The intuition behind this is that there is no indication (from existing mappings or from schema structure and constraints) that these new mappings are preferred over those that already exist. Since our goal is to maintain the semantics of existing mappings as much as possible, we perform no adaptation unless necessary.

[2] We perform join minimization on both the source and target queries.

Example 10 Assume that a new foreign-key constraint is added in the source schema of Fig. 1 from the element `budget` to the element `amount`. Mappings m_2 and m_3 are not affected by this change. Chasing association A_1 with the schema constraints introduces a new join path through which a `project` and a `grant` can be associated. However, there is no reason to assume that mapping m_1 needs to be adapted since the initial way of joining projects and grants (through the foreign key f_1) is still valid. □

Removing constraints: As with the addition of a constraint, the removal of a constraint has no effect on the validity of the existing mappings but may affect the consistency of their semantics. The reason is that mappings may have used assumptions that were based on the constraint that is about to be removed. As before, we assume that a source constraint is removed. (The same reasoning applies for the target case.) We consider a mapping to be affected if its source association uses some join condition(s) based on the constraint being removed. More precisely, a mapping m: **foreach** A^S **exists** A^T **with** D, with $m \in \mathcal{M}$ of a mapping system $<\mathcal{S}, \mathcal{T}, \mathcal{M}>$, needs to be adapted after the removal of a source constraint F: **foreach** X **exists** Y **with** C if $X \sqcup Y \sqcup C \stackrel{\cdot}{\preceq} A^S$.

Once we detect that a mapping m needs to be adapted, we apply the following steps. (Algorithm 2 provides a succinct description of these steps.) The intuition of Algorithm 2 is to take the maximal independent sets of semantically associated schema elements of each affected association used by the mapping. We start by breaking apart the source association A^S into its set $\mathcal{P}$ of structural and user associations, that is, we enumerate all the structural and user associations of the source schema that are dominated by A^S. We then chase them by considering the set of schema constraints *without* F. The result is a set of new logical associations. Some of them may include choices (due to the existence of choice types) that were not part of the original association A^S. We eliminate such associations. The criterion is based on dominance, again: we only keep those new logical associations that are dominated by A^S. Let us call this set of resulting associations $\mathcal{A}'$. By construction, the logical associations in $\mathcal{A}'$ will contain only elements and equalities between them that were also in A^S; hence they will not represent any additional semantics. For every member A_a in $\mathcal{A}'$ and for every way A_a can be dominated by association A^S (i.e., for every renaming function $h{:}A_a \rightarrow A^S$), a new mapping m_a is generated of the form **foreach** A_a **exists** A^T **with** D'. The set D' consists of those correspondences that are covered by the pair $<A_a, A^T>$ and such that their renaming h is included in D or implied by it. In other words, D' consists of the correspondences of D that are covered by $<A_a, A^T>$ and $<A^S, A^T>$ in the same way. Let us call M^* the set of mappings m_a. From the mappings in M^* we need to keep only those that are as close as possible to the initial mapping m. This is achieved by eliminating every mapping in M^* that is dominated by another mapping in M^*, so we keep only the one that has the maximum number of elements. The following example illustrates the algorithm.

Example 11 Consider the mappings m'_{1a}, m'_{1b}, m_2, and m_3 in example 9 and let us remove the constraint f_7 we added there. It is easy to see that mappings m_2 and m_3 are not affected because they do not include a join between `S.grants` and `S.companies`. However, both m'_{1a} and m'_{1b} are affected. Consider the mapping m'_{1a} (the other is handled in a similar way). Its source association, A_{1a} of example 9, is broken apart into the structural associations (recall Fig. 2): P_1^S, P_2^S, P_4^S, P_5^S, and P_6^S. Those structural associations are chased and result in a set of logical associations. P_1^S results in (recall Fig. 3) A_1 and A_2, but the latter is eliminated since it is not dominated by the original association A_{1a}, which requires the existence of a `private` sponsor independently of the existence of a `government` sponsor. The chase of P_2^S, P_4^S, P_5^S, and P_6^S will result in associations A_3, A_5, A_6, and A_7 respectively. Each of the resulting associations will be used to form a new mapping. Association A_1 generates mapping m_1 (which is the one we started with in example 9). Association A_6 generates mappings (recall example 8) m_o and m_2. However, m_o covers the correspondence on the `leader` through a join on the `CEO`, while m_2 does it through a join on the `owner`. Since the initial mapping m'_{1a} covers the `leader` through a join on the `owner`, mapping m_o is eliminated. The mappings generated by A_3 and A_5 are dominated by mapping m_1, while the one generated by A_7 is dominated by m_2. Hence, those mappings are also eliminated and the final result of the algorithm, for the case of m'_{1a}, consists of the mappings m_1 and m_2. The algorithm will give similar results for the case of m'_{1b}, while mappings m_2 and m_3 will not be affected by the removal of constraint f_7 and will not be modified. □

Algorithm 2 – Constraint Removal

```
Input:
  Set of semantically valid mappings $\mathcal{M}$
  Constraint $F$: foreach $X$ exists $Y$ with $C$ of schema $\mathcal{S}$
Body:
  $\mathcal{X} \leftarrow$ constraints in $\mathcal{S}$,   $\mathcal{M}' \leftarrow \emptyset$
  For every $m \leftarrow$(foreach $A^S$ exists $A^T$ with $D$) $\in \mathcal{M}$
    If $(X \sqcup Y \sqcup C \stackrel{\cdot}{\preceq} A^S)$ {
      $\mathcal{P} \leftarrow \{P \mid P$ structural or user association $\wedge\ P \stackrel{\cdot}{\preceq} A^S\}$
      $\mathcal{A}' \leftarrow \{A \mid A \in chase_{\mathcal{X}-\{F\}}(P)\ \wedge\ P \in \mathcal{P}\ \wedge\ A \stackrel{\cdot}{\preceq} A^S\}$
      $\mathcal{M}^* \leftarrow \emptyset$
      For every $A_a \in \mathcal{A}'$ and every renaming $h{:}A_a \rightarrow A^S$
        $D' \leftarrow \{e_1 = e_2 \mid e_1\ (e_2)$ well-defined expressions over
              $A_a\ (A^T)\ \wedge$ "$h(e_1) = e_2$" in or implied by $D\}$
        $m_a \leftarrow$ (foreach $A_a$ exists $A^T$ with $D'$)
        $\mathcal{M}^* \leftarrow \mathcal{M}^* \cup \{m_a\}$
      $\mathcal{M}^{**} \leftarrow \{m' \mid m' \in \mathcal{M}^*\ \wedge\ \nexists\, m'' \in \mathcal{M}^*{:}\ m' \stackrel{\cdot}{\preceq} m''\}$
      $\mathcal{M}' \leftarrow \mathcal{M}' \cup \mathcal{M}^{**}$
    else include $m$ in $\mathcal{M}'$
Output:
  New set of mappings $\mathcal{M}'$
```

5.2 Schema pruning or expansion

Among the most common changes used in schema evolution systems are those that add or remove parts of the schema structure, for example, adding a new attribute on a relational table or removing an XML Schema element.

When a new structure is added to a schema, it may introduce some new structural associations. Those structural associations can be chased and generate new logical associations. Using those associations new semantically valid mappings can be generated, and hence the mapping universe is expanded. However, they are not added to the set of existing mappings. The reason is that there is no indication of whether they describe any of the intended semantics of the mapping system. This can be explained by the fact that there is no correspondence covered by any of the new mappings that is not covered by any of those that already exist. On the other hand, since the structure and constraints used by the existing mappings are not affected, there is no reason for adapting any of them.

Example 12 Consider the case in which the source schema of Fig. 1 is modified so that each company has nested within its structure the set of laboratories that the company operates. This introduces some new mappings in the mapping universe, for example, a mapping that populates the target schema only with companies that have laboratories. Whether this mapping should be used is something that cannot be determined either from the schemas or from the existing mappings. On the other hand, mapping m_2 that populates the target with companies, independently of whether they have labs, remains valid and consistent. □

In many practical cases, a part of the schema is removed either because the owner of the data source does not want to store that information or because she may want to stop publishing it. The removal of an element forces all the mappings that are using that element to be adapted. We consider first the removal of atomic type elements.

When atomic element e is removed, each constraint F in which e is used is removed by following the procedure described in Sect. 5.1. If the atomic element e to be removed is used in a correspondence V, then every mapping m that is covering V has to be adapted. More specifically, the equality in the **with** clause of the mapping that corresponds to V is removed from the mapping. If mapping m was covering only V, then the **with** clause of m becomes empty; thus m can be removed. If the atomic element e is used neither in a correspondence nor in a constraint, it can be removed from the schema without affecting any of the existing mappings. Algorithm 3 describes the steps followed to remove an atomic element. To remove an element that is not atomic, its whole structure is visited in a bottom-up fashion starting from the leaves and removing one element at a time following the procedure described in Algorithm 3. A complex type element can be removed if all its attributes (children) have been removed.

Algorithm 3 – Atomic element deletion

```
Input:
  Mapping system $<\mathcal{S},\mathcal{T},\mathcal{M}>$
  Atomic element $e$
Body:
  While exists constraint $F$ that uses $e$
      remove $F$
  $\forall$ $m\leftarrow$(foreach $A^S$ exists $A^T$ with $D$) $\in M$
      $D \leftarrow \{q \mid c$ is correspondence covered by $m \wedge$
                  $c$ is not using $e \wedge$
                  $q$ is the with clause of $c\}$
      if $D=\emptyset$ remove $m$ from $M$
Output:
  The updated set $M$
```

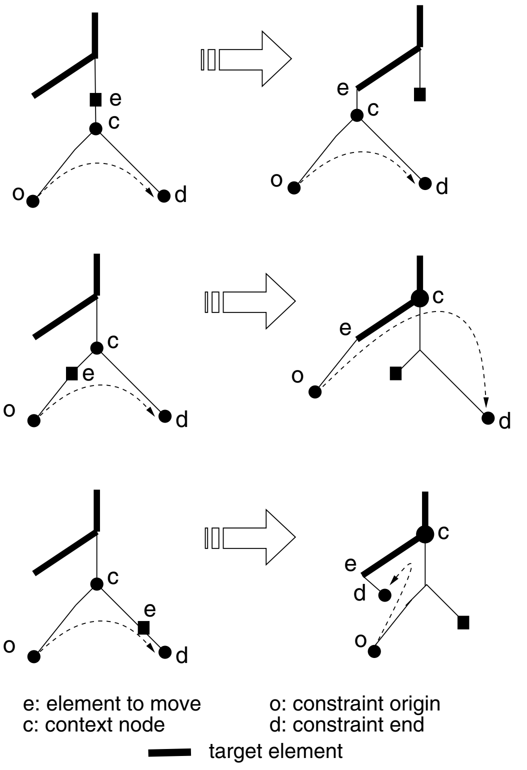


Fig. 4. Updating a constraint after an element move

Example 13 In the mapping system of Fig. 1, removing element `budget` from `project` will not affect any mappings since it is used neither in either a constraint nor a correspondence. On the other hand, removing `code` will invalidate mapping m_1, which populates the target with project codes and the sponsor of privately funded projects. After the removal of `code`, the element `projects` does not contribute to the population of the target schema. However, according to our algorithm the only modification that will take place in mapping m_1 will be the removal of the equality i.`privProject.code`=p.`project.code` from the **with** clause. This reflects the basic principle of our approach to preserve the semantics of the initial mappings by performing the minimum required changes during the adaptation process. □

Another common operation in schema evolution is updating the type of an element e to a new type t. This case will not be considered separately since it is easy to show that it is equivalent to removing element e and then adding a new one of type t and with the same name to e.

5.3 Schema restructuring

One way a schema may evolve is by changing its structure without removing or adding elements. There are three common operations of this kind of evolution that we consider: rename, move, and copy. The first renames a schema element, and it is mainly a syntactic change. It requires visiting all the mappings and updating every reference to the renamed element with its new name. The second operation moves a schema element to a different location, while the third does the same but moves a replica of the element instead of the element itself. When an element is copied or moved, it carries with it design choices and semantics it had in its original location, i.e., schema constraints. Mapping selections and decisions that were used in the original location should also apply in the new location.

5.3.1 Adapting schema constraints

Assume that a schema element e is to be moved to a new location. Due to this move, constraints that are using the element e may become invalid and must be adapted. Recall that the constraints we consider are NRIs (Definition 3) that naturally include the constraints of relational and XML Schemas. An NRI constraint F is of the form **foreach** P_0 [**foreach** P_1 **exists** P_2 **with** C], where the P_1 and P_2 start from the last variable of the P_0, which represents the context element, and C is of the form $e_1{=}e_2$, where e_1 and e_2 are expressions depending on the last variable of P_1, respectively, P_2. To realize how a constraint F is affected by the change, we have to consider the relative position in the schema of element e with respect to the context element of F in the schema structure. Figure 4 provides a graphical explanation of how F is adapted to the move of element e. In the figure, for constraint F, we use c, o, and d to denote, respectively (both before and after the move), the context element, the element identified by the query **select** e_1 **from** P_0, P_1 (also called the origin element), and the element identified by the query **select** e_2 **from** P_0, P_2 (also called the destination element). If element e is an ancestor of the context node c, then the nodes c, o, and d move rigidly with e. The modified constraint will have the form **foreach** P'_0 [**foreach** P'_1 **exists** P'_2 **with** C], where P'_0 is from the query determining the new context node c. The expression P'_1 is the same as P_1 except that the starting expression is updated so that it corresponds to the new location of the context node (a similar change applies to P'_2 as well). If the context node is an ancestor of e, then e is used in either P_1 or P_2. Assume that e is used in P_1 (the other case is symmetric). This case is shown in the second part of Fig. 4. Node o moves rigidly with e to a new location, while d remains in the same position. We then compute a new context node as the lowest common ancestor between the new location of o and d. The resulting constraint is then **foreach** P'_0 [**foreach** P'_1 **exists** P'_2 **with** C'], where P'_0 determines the new context node and P'_1 and P'_2 are relative to query P'_0 and determine the new location of o and d. The C' is the result of changing C so that it uses the end points of paths P'_1 and P'_2.

Example 14 Assume that the designer of schema S in the mapping system of Fig. 1 has decided to store grants nested within each company so that each company contains the set of its grants. This translates to a move of the element `grants` under the element `company`. Consider the constraint f_2 of example 3 specifying that each grant having a private sponsor refers to its contact information. Once the grants are moved, this constraint becomes inconsistent since there are no grant elements under the schema root `S`. To adapt the constraint, we use the previously described algorithm: we are in the second case shown in Fig. 4, in which the element that moves is between the context element c (the root, in this case) and o. The element `grants` in its new location is

select `a.company.grants` **from** `S.companies` a

The variable binding of a does not exist in f_2 so it is appended to it, and every reference to expression `S.grants` is replaced by the expression `a.company.grants`. The final form of the adapted constraint f_2 is shown below. (The **exists** clause need not be changed since the new context element continues to be the root.)

foreach `S.companies` a, a`.company.grants` g,
g`.grant.sponsors` n, n`.sponsor.private` p
exists `S.contacts` c
with c`.contact.cid=`p □

5.3.2 Adapting mappings

When an element is moved to a new location, some of the existing logical associations that were using it become invalid and new ones have to be generated. To avoid redundant recomputation by regenerating every association, we exploit information given by existing mappings and computation that has already been performed. In particular, we first identify the mappings that need to be adapted by checking whether the element that is moved is used in any of the two associations on which the mapping is based. Let A be an association that is using the element e that is about to move, and let t be the element in its new location. More precisely, assume that e and t have the following forms:

e : **select** e_{n+1} **from** $e_0\ x_0, e_1\ x_1, ..., e_n\ x_n$
t : **select** t_{m+1} **from** $t_0\ y_0, t_1\ y_1, ..., t_m\ y_m$

We first identify and isolate the element e from association A by finding the appropriate renaming from the **from** clause of e to A. For simplicity, assume that this renaming is the identity function, that is, A contains literally the **from** clause of e. In the next step, the **from** clause of t is inserted in the front of the **from** clause of A. We then find all *usages* of e_{n+1} within A and replace them with t_{m+1}. After these replacements, it may be the case that some (or all) of the variables $x_0, \ldots, x_n$ have become redundant (i.e., not used) in the association. We eliminate all such redundant variables. Let us denote by A' the resulting association.

Since the element t in the new location may participate in its own relationships (based on constraints) with other elements, those elements have to be included as well in the new adapted version of association A'. We do this by chasing A' with the schema constraints. The chase may produce multiple associations $A'_1, \ldots, A'_k$ (due to the choice types). Finally, any mapping using the old association A, say, **foreach** A **exists** B **with** D, is removed from the list of mappings and replaced with a number of mappings m_i: **foreach** A'_i **exists** B **with** D', one for each association A'_i. The conditions D' correspond to the correspondences in D *plus* any additional correspondences that may be covered by the pair $<A'_i, B>$ (but not by the original pair $< A, B >$).

As an important consequence of our algorithm, all the joins that were used by the original mapping and were not affected by the schema change will be used, unchanged, by the new, adapted mapping. Hence, we preserve any design choices that might have been made by a human user based on the original schemas. We illustrate the adaption algorithm with the following example.

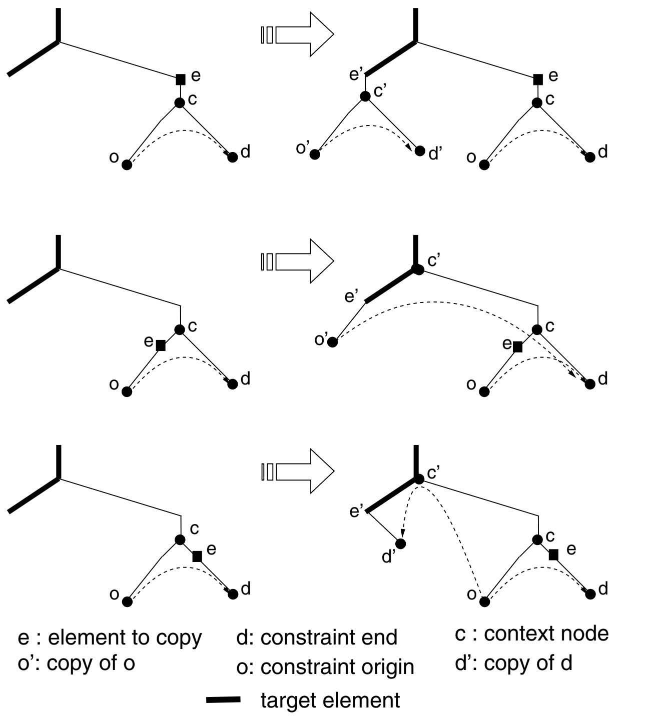


Fig. 5. Introducing a new constraint after an element copy

Example 15 Assume that in the mapping system of Fig. 1 grants are moved under company as in example 14. This change affects neither mapping m_3 nor mapping m_2. [Recall from Sect. 5.2 that just the addition of a new structure (grants, in this case) for m_2 does not require m_2 to be adapted]. However, mapping m_1, based on the logical association A_1 (Fig. 3), is affected. First, schema constraints are adapted as described in example 14. Then we run the mapping adaption algorithm, described above, for e: select S.grants from _ and t: select o.company.grants from S.companies o (denote here by _ the empty from clause). The clause "S.companies o" is added to the from clause of A_1. Next, all occurrences of S.grants are replaced by o.company.grants. After this, the resulting association is chased with the source schema constraints. The (adapted) constraints f_1, f_2, and f_3 are already satisfied and hence not applicable. However, f_4 and f_5 will be applied. The chase ensures coverage of the two correspondences on cname and name, the last one in two different ways. Hence, two new mappings are generated. The first is:

```
m'1a: foreach S.companies o, S.projects p,
              o.company.grants g, g.grant.sponsors n,
              n.sponsor.private r,
              S.contacts c, S.persons e
        where p.project.source=g.grant.gid and
              r=c.contact.cid and
              o.company.CEO=e.person.SSN
       exists T.privProjects j, T.companies m
        where j.privProject.holder=m.company.cname
       with   m.company.cname=o.company.cname and
              m.company.leader=e.person.name and
              j.privProject.code=p.project.code and
              j.privProject.sponsor=c.contact.email
```

while the second is the one that considers the owner of the company as the leader instead of the CEO. Note how the algorithm preserved the choice made in mapping m_1 to consider private projects and how the initial relationships between projects and grants as well as grants and contacts in mapping m_1 were also preserved in the new mapping. □

In the above analysis, we considered the case of moving an element from one place in the schema to another. In the case where the element is copied instead of being moved, the same reasoning is used and the same steps are executed. The only difference is that the original mappings and constraints are not removed from the mapping system as in the case of a move. Schema constraints and mapping choices that have been made continue to hold unaffected after a structure in the schema is copied. Figure 5 demonstrates how a new constraint is introduced when an element is copied.

6 A ranking mechanism for rewritings

In the previous sections, we presented a methodology for adapting mappings that are affected by a change in the schema structure or constraints. The specific methodology detects the mappings that are affected by the change and generates a number of semantically valid rewritings. All those rewritings are consistent with the semantics of the schemas, the structure, and the user choices that have been made in the past and are encoded in the existing mappings. An affected mapping can be replaced with one, two, or even all of its generated rewritings. The existence of more than one candidate is natural since the new modified schema has new semantics. However, there are cases in which not all of the generated mappings describe the semantics of the intended data transformation between the two schemas. Selecting only those that do requires human intervention. In order to assist the user in this selection, we developed a model for systematically ranking the rewritings. A number of approaches in the literature [6,23] have used the cost of updating a materialized target as a measurement of the importance of the mapping. Although the maintenance cost is important, in our context, and since our main goal is to preserve, as much as possible, the semantics of the mapping that is to be adapted, we believe that a different criterion is more appropriate. For that, we introduce a new model for measuring the similarity of two mappings.

The similarity measure is based on the observation that two mappings are semantically similar if they use common schema elements. The more schema elements and equalities they have in common, the more (semantically) similar they are. Conversely, the more schema elements in which they differ, the less (semantically) similar they are.

Definition 16 *Let $|m|$ represent the sum of the number of schema elements used in a mapping m and the number of equalities between these elements in that mapping. The* relative similarity *of mapping m_1 to mapping m_2 (denoted $\mathcal{S}(m_1, m_2)$) is defined as the weighted sum*

$$\mathcal{S}(m_1, m_2) = \rho_1 \frac{|m_1 \cap m_2|}{|m_2|} + \rho_2 \frac{|m_1 - m_2|}{|m_1|} \quad (1)$$

where $|m_1 \cap m_2|$ is the number of elements and equalities that are used in both mappings m_1 and m_2, and $|m_1 - m_2|$ is the *number of elements and equalities that are used in m_1 but are not used in m_2. The quantities ρ_1 and ρ_2 are constant values for which $\rho_1+\rho_2=1$.*

Intuitively, the first fraction of the *relative similarity* $\mathcal{S}(m_1, m_2)$ measures how many elements mapping m_1 has in common with mapping m_2, while the second measures how many surplus elements mapping m_1 has compared to mapping m_2. This is similar to the notion of recall and precision in information retrieval. The first fraction measures the recall and the second the precision. Factors ρ_1 and ρ_2 determine the importance of the two fractions in the relative similarity. Note that relative similarity is not a symmetric relationship.

In some cases, the relative similarity is not enough to determine if one mapping is more preferable than another. Consider, for example, the case where a new constraint is added in one of the schemas. A mapping m_o needs to be rewritten since one of its associations is affected by this change. The association is chased and results in a set of new associations, each one of which is used to form a new candidate rewriting. To rank the rewritings, the relative similarity of each one them with mapping m_o is calculated. From the way the rewritings were generated, they will all contain the elements and conditions of the mapping m_o. Thus, deciding which one is more preferable depends on the number of surplus elements and conditions. For two or more mappings, that number may be the same, making the relative similarity of these mappings the same. In order to be able to select one over another, we introduce a new quantity called the support level. Intuitively, the support level utilizes knowledge of the remaining existing mappings to guess which of the rewritings that have the same relative similarity is more likely closer to the semantics that the user has in mind, and it does so by computing the relative similarities of the rewriting with every other existing mapping.

Definition 17 *Let M be a set of mappings. The* support level *of mapping m from the set M is defined as*

$$\mathcal{L}_M(m) = \frac{\Sigma_{m' \in M} \mathcal{S}(m, m')}{|M|} \qquad (2)$$

Example 16 Consider the two mappings m'_{1a} and m'_{1b} in example 9, which were generated as a result of the addition of a new constraint. Recall that the difference between those two mappings is that the first one considers the CEO as the leader of the company while the second considers the owner. The relative similarity of those mappings to mapping m_1 is the same. However, one can observe that the existing mapping m_2 considers the owner of the company as the company leader. Based on that, the rewriting mapping m'_{1b} seems to be more appropriate than m'_{1a}. Indeed, the support level of mapping m'_{1b} is higher than the support level of mapping m'_{1a}. □

It has to be noted that the ranking mechanism is not used to eliminate any of the rewritings. All the mappings generated by the algorithms presented in the previous sections are semantically valid mappings and valid rewritings. The ranking mechanism is used to rank these mappings according to what the system judges to be the most likely (based on the schema structure, the semantics, and, most of all, the existing mappings).

7 Performance analysis

A main performance feature of the presented approach is the incremental maintenance of the mappings, which leads to less computation than required by a mapping tool if the mappings were regenerated from scratch after modification of the schemas. In the latter case, the mapping tool would have first generated all the mappings in the mapping universe and then an expert user would have to select only those that describe the intended semantics of the transformation. A great deal of these semantic user choices are embodied in the existing mappings. Our algorithms use that knowledge to avoid computation of mappings that would be anyway rejected by the user and also computations that will give results that are already available in the existing mappings. For example, when a new constraint is added and an association of a mapping needs to be updated, the chase does not start from the structural associations, as a mapping tool like Clio [31] would have done, but from the existing association in the mapping.

To determine what mappings are affected by a schema change, the algorithms check for dominance. Dominance is checked by looking for one-to-one renaming functions between the variables of two queries. Finding such a renaming in the general case requires time exponential in the size of the queries. However, the NRI constraints and the schema elements are based on linear paths for which a renaming one-to-one function can be found (if it exists) in linear time. Furthermore, the problem of finding whether an association A is dominated by association B, when either A or B has an empty where clause, can be reduced to the problem of finding a subtree isomorphism, which requires $O((k^1.5/\log k)n)$ [34].

Another operation used in the algorithms is the chase. In general, the chase may not terminate. However, if we restrict the class of schemas to those with acyclic constraints and non-recursive types, the chase always terminates in time polynomial in the size of the schema and the constraints [31]. This is a natural restriction since we can still capture the majority of the constraints met in real systems (including XML Schema key references) and also we avoid having an infinite number of rewritings generated by the existence of recursive types or cyclic constraints.

8 Mapping adaptation experience

To evaluate the effectiveness and usefulness of our approach, we have implemented a prototype tool called ToMAS[3] and we have applied it to a variety of real application scenarios. In this section, we describe ToMAS and report our experience using it to adapt mappings between real schemas. The results of the experiments indicate that indeed it is worth using a mapping adaptation system like ToMAS to automatically maintain the mappings between schemas. Specifically, we show that (i) the time needed for incrementally updating the mappings under schema changes is negligible and (ii) this incremental adaptation requires much less effort than a "from-scratch" rebuilding of the mappings. At the end, we highlight the benefits of using a mapping adaptation tool in concert with a physical design tool that manipulates schemas. In particular, we show how our

[3] ToMAS stands for **To**ronto **M**apping **A**daptation **S**ystem

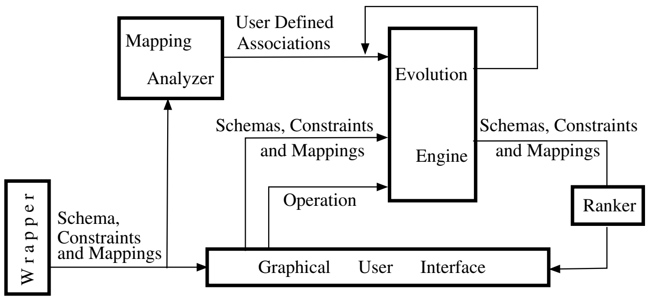

Mapping Analyzer
User Defined Associations
Evolution Engine
Schemas, Constraints and Mappings
Schemas, Constraints and Mappings
Operation
Ranker
Wrapper
Schema, Constraints and Mappings
Graphical User Interface

Fig. 6. ToMAS architecture

tool can facilitate the selection of a good relational schema for storing XML data.

8.1 Architecture

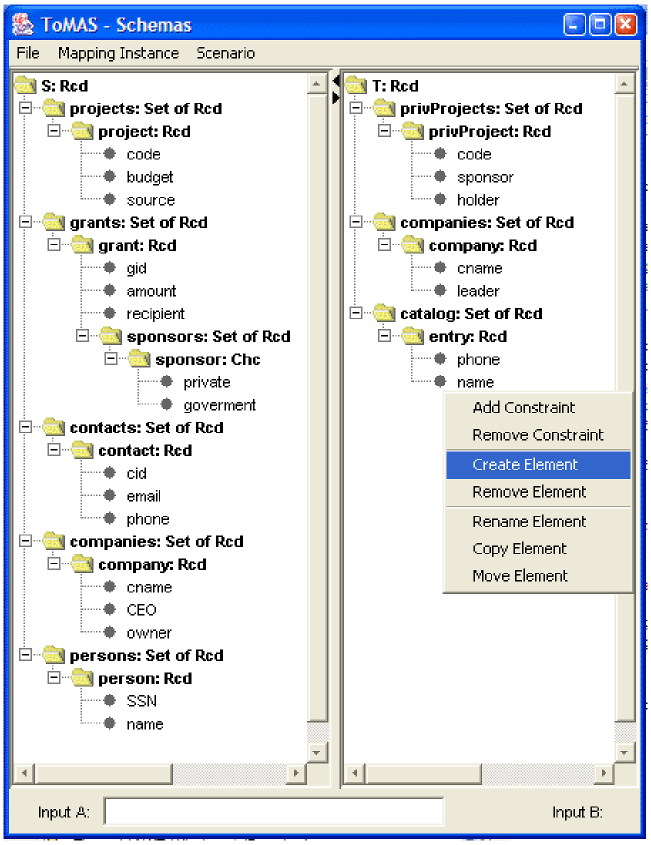
ToMAS follows a modular architecture, as can be seen in Fig. 6. At the heart of ToMAS is the *evolution engine* that contains implementations for each of the evolution operators we described earlier. ToMAS manages a mapping system (a pair of schemas and a set of mappings between them). As the schemas evolve, ToMAS detects mappings that may need to be updated, and based on the schema constraints and the user associations it creates a set of potential rewritings that are consistent with the modified schemas.

To update the mappings, apart from the schema and constraint information, the evolution engine has to know the user associations that are contained in the mappings. This information is provided by the *Mapping Analyzer*. The Mapping Analyzer constructs all the structural associations of the schemas and chases them to generate the set $\mathcal{A}_S$ of logical associations that are based on the schema structure and constraints. Each association A used in an existing mapping is then chased to become a logical association so that the mapping will become a semantically valid mapping. If the resulting logical association is not in the set $\mathcal{A}_S$, then the association A that the mapping was using represents a choice that has been explicitly stated by the user; thus, it is a *user association*. The set of user associations found, along with the semantically valid mappings, is provided to the Evolution Engine. This step is performed once at the beginning of the evolution process, right after the schemas are loaded.

The system is equipped with a set of pluggable schema *wrappers* used to import schemas and mappings from different data models into the internal nested relational representation. Currently, relational and XML wrappers have been implemented.

The schemas and the mappings are presented to the user through a *graphical user interface* (Fig. 7). Through this interface, the user may also modify the schemas. For each modification, the schemas, the requested modification, and the existing mappings are provided to the evolution engine that updates both the mappings and the schemas. The results are returned back to the interface for presentation to the user and further modifications.

The last component of ToMAS is the mapping *ranker*. Its role is to rank the candidate rewritings generated by the



```
S: Rcd
  projects: Set of Rcd
    project: Rcd
      code
      budget
      source
  grants: Set of Rcd
    grant: Rcd
      gid
      amount
      recipient
      sponsors: Set of Rcd
        sponsor: Chc
          private
          goverment
  contacts: Set of Rcd
    contact: Rcd
      cid
      email
      phone
  companies: Set of Rcd
    company: Rcd
      cname
      CEO
      owner
  persons: Set of Rcd
    person: Rcd
      SSN
      name
```

```
T: Rcd
  privProjects: Set of Rcd
    privProject: Rcd
      code
      sponsor
      holder
  companies: Set of Rcd
    company: Rcd
      cname
      leader
  catalog: Set of Rcd
    entry: Rcd
      phone
      name
```

Fig. 7. ToMAS user interface

evolution engine according to the ranking criterion that was presented in Sect. 6 before sending them to the Interface.

8.2 Performance

We now investigate the efficiency of our proposed incremental mapping adaptation algorithms. We conducted a series of experiments on some schemas, both relational and XML, that vary in terms of size and complexity. Their characteristics are summarized in Table 1. The size is shown in terms of atomic schema elements, and within the brackets is the number of schema constraints. We used two versions of each schema to generate mappings from the first version to the second. Either the different versions of each schema were available on the Web (representing two different evolutions of the same original schema), or, whenever a second version was not available, it was manually created. Using the Clio mapping generation tool a number of correspondences were used to generate a set

Table 1. Test schema characteristics

Schema	Size	Corresp/ces	Mappings
ProjectGrants	16 [6]	6	7
DBLP	88 [0]	6	12
TPC-H	51 [10]	10	9
Mondial	159 [15]	15	60
GeneX	88 [9]	33	2

of semantically meaningful mappings (the last two columns of Table 1 indicate their exact numbers). From them, two mappings were selected as those representing the intended semantics of the correspondences. These mappings were the mappings we had to maintain through schema changes.

A random sequence of schema changes was generated and applied to each schema. Even for only two mappings, due to the large size of the schemas, it was hard for a user to understand how the mappings were affected by the changes and how they should be adapted. We considered two alternative adaptation techniques. The first was to perform all the necessary modifications on the schemas and at the end use a mapping generation tool (e.g., Clio) to regenerate the mappings. Due to the fact that the names of the attributes might have changed and elements might have moved to different places in the schema, it was hard to use schema matching tools to reinfer the correspondences. This means that the correspondences had to be entered manually by the user. Once this was done, the mapping generation tool produced the complete set of semantically valid mappings and the user had to browse through all of them to find those that were describing the intended semantics. The second alternative was to perform the schema changes and let ToMAS maintain the mappings. ToMAS returned only a small number of mappings since it utilized knowledge about choices embedded in the initial set of mappings. At the end, the user had to go through only the small number of adapted mappings (ranked according to our ranking criterion) and verify their correctness. We performed and compared both techniques experimentally.

8.2.1 Time performance

For each change, we measured the time needed for the schema to be updated and the mappings to be adapted. Figure 8 summarizes the results of this experiment. The time indicated for each operator is the average time of this kind of operator in the mapping system. What we noticed was that the time of each operator depends not only on the nature of the schemas and the mappings but also on the sequence of changes that have taken place before. The reason is that operators have different tasks to perform depending on the schema on which they are applied. We calculated the average time of completion for each operator, and we present it in Fig. 8.

Creation of new elements in the schema does not require any updates on the mappings. However, the inserted structure has to be type-checked for consistency before being inserted into the schema. This checking means the "create element" operation takes slightly more time compared to renaming.

Copying and moving of an element are mostly syntactic modifications. They require deletion of the elements to be moved (copied) in a mapping or in a constraint. At the end of the move (copy), some chasing takes place at the new location. This chasing accounts for the time difference between these two restructuring operators and element renaming or creation. TPC–H is the schema with the largest number of constraints (compared to the size of the schema), which makes chasing take longer than in any other schema. On the other hand, if we exclude the ProjectGrant schema that is really small, the shortest adaptation time is given by DBLP, which is the only schema with no constraints.

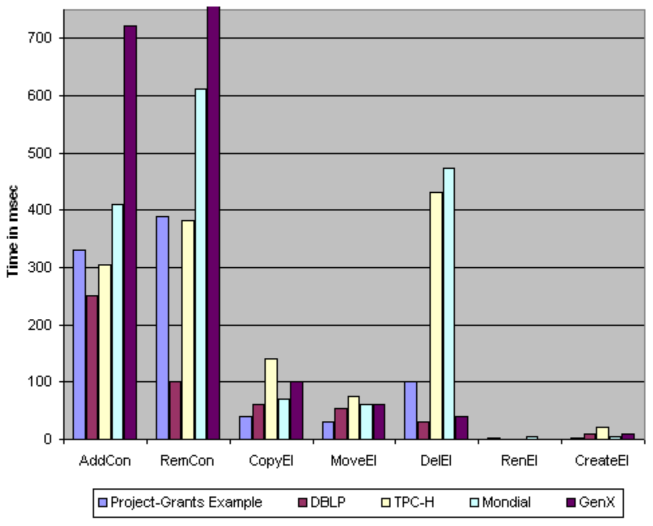


Fig. 8. Average mapping adaptation times

Addition or removal of constraints, as expected, are expensive operations since they involve chasing. Adding a new constraint requires chasing an affected association, while removal involves the reconstruction of a number of structural associations, their chase, and the checking of whether the results of the chase are subsumed by the affected association. All those operations make mapping adaptation under constraint removal more expensive than adaptation under constraint addition. The surprising result is the DBLP schema, in which in every test we have conducted the constraint removal outperformed the constraint addition. This may be happening because DBLP is the only schema that has initially no constraints. Constraints that are removed are those that have been generated through an "addConstraint" operation and are not creating long chains of consecutive constraints. As a consequence, the chase never had more than one chase step. Furthermore, DBLP is very shallow, so the time required for the construction of the structural associations was almost negligible. An interesting observation is the performance of the GeneX schema under constraint removal, which is the only one that took more than 1 s, even though there was only one mapping that had to be updated. The reason is that the mapping was really large, involving 29 joins; hence the corresponding association had 29 variables. Finding one-to-one renamings was the expensive operator for such a complex mapping.

Deletion of an element requires first the removal of the correspondences and the constraints that are using the specific element. This means that the cost of an element deletion is at least as much as the constraint removal. On the other hand, if the element is not used by any constraint, it can be removed in time that is almost equal to the time of renaming. On average, the first kind of removal operation is balanced by the low cost of the second kind; hence the average performance is as shown in Fig. 8.

The above analysis shows that, despite the size and complexity of the schemas, the time for mapping adaptation is short enough to permit its use both in interactive applications and automatic mapping adaptation.

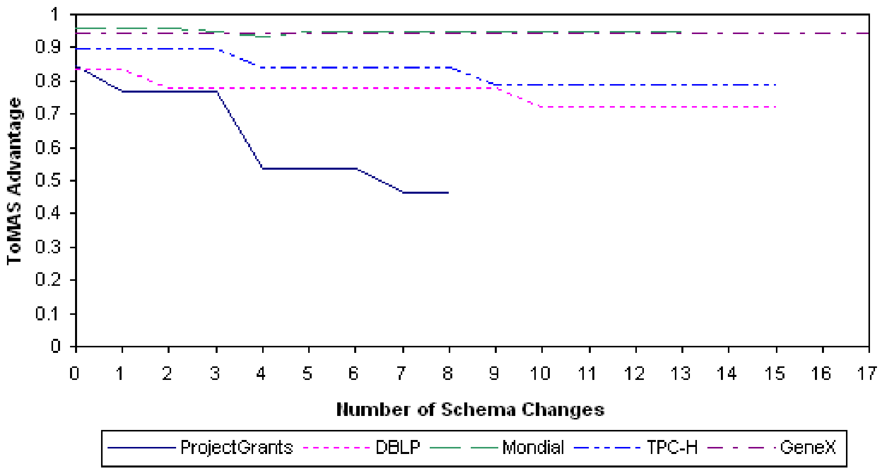


Fig. 9. The benefit of ToMAS for different schemas as a function of the number of changes

8.2.2 Benefit

We compared the user effort required in the two approaches. In the first approach, where mappings have to be regenerated from scratch, the effort of the user was measured as the number of correspondences that have to be respecified, plus the number of mappings that the mapping generation tool produces and which the user has to browse to select those that describe the intended semantics of the correspondences. On the flip side, if ToMAS is used, the effort required is just the browsing and verification of the adapted mappings. As a comparison measurement we used the following quantity, which specifies the advantage of ToMAS against the "from-scratch" approach. A value of 0.7, for example, means that ToMAS saves 70% of the effort required in the "from-scratch" alternative.

$$1 - \frac{\textit{mappings generated by ToMAS}}{\textit{mappings generated by the mapping tool} + \textit{correspondences}}$$

Figure 9 provides a graphical representation of how the above quantity changes as a function of the number of changes for the various schemas. As the number of changes becomes larger, and the modified schemas become very different than their original version, the advantage of ToMAS is reduced, but the rate of reduction is small. Notice, for example, the line that corresponds to the ProjectGrants example used in the paper. After eight changes, ToMAS has lost almost half of its advantage. The reason is that the schema is small and after eight changes it has been completely restructured, describing new data semantics. In such a case, it makes no sense to try to preserve the transformation described by the initial mappings (as ToMAS does) while schemas continue to evolve. On the flip side, for the larger Mondial and GeneX schemas it can be noticed that even after 17 changes, ToMAS still has an advantage. The reason is that even after the 17 changes the schemas have not changed that much and keep describing the semantics of the data that they were describing initially, in which case it makes sense to try to preserve the initial transformation. Fortunately, practice has shown that schemas do not change radically. The new evolved schemas are not dramatically different from their original version, and in these cases, ToMAS is the right tool to use.

An interesting alternative to investigate is how the ToMAS approach compares to the solution suggested in Rondo [29].

TABLE Show	**TABLE Show1**	**TABLE NYTRev**
ShowId,	ShowPart1Id,	ReviewsId,
type,	type,	review,
title,	title,	parentShow
year,	year,	**TABLE Reviews**
boxOffice,	boxOffice,	ReviewsId,
videoSales,	videoSales	review,
seasons,	**TABLE Show2**	parentShow
description	ShowPart2Id,	**TABLE Episode**
TABLE Review	type,	EpisodeId,
ReviewsId,	title,	name,
tilde,	year,	parentShow
reviews,	seasons,	
parentShow	description	
TABLE Episode		
EpisodeId,		
name,		
parentShow		
a	**b**	

Fig. 10. Two relational designs for the IMDB DTD

In particular, according to this approach, when a schema is modified, a new mapping must be generated between the old and the new version of the schema describing how exactly the schema has changed. This mapping is combined with the old mapping using the "compose" operator of model management to form the new adapted mapping. The drawback of this approach is that, when the mapping between the old and the new schema is formulated, the existing mappings are not taken into consideration. As a result, and as our preliminary experiments have shown, this may produce more rewritings than ToMAS would have produced. On the flip side, the advantage of the mapping composition approach is that it works even when the difference between the old and the new schema is large and is not given as a list of incremental changes but by an arbitrary set of mappings. It is part of our future plans to further investigate this issue in order to identify and have a clear understanding of the cases in which ToMAS is more preferable than the Rondo approach and vice versa.

8.3 Case study: ToMAS in physical design

We present our experience using ToMAS within one important application: physical data design. In the last few years, there has been a growing interest in storing XML data in

```
<imdb>
  FOR $x0 IN $doc/DB/Review, $x1 IN $doc/DB/Show
  WHERE $x0/ReviewsId/text() = $x1/ShowId/text()
  RETURN
    <Show>
      <type> $x1/type/text() </type>
      <title> $x1/title/text() </title>
      <year> $x1/year/text() </year>
      FOR $x0L1 IN $doc/DB/Review, $x1L1 IN $doc/DB/Show
      WHERE
        $x0L1/ReviewsId/text() = $x1L1/ShowId/text() AND
        $x1/videoSales/text() = $x1L1/videoSales/text() AND $x1/boxOffice/text() = $x1L1/boxOffice/text() AND
        $x1/type/text() = $x1L1/type/text() AND $x1/title/text() = $x1L1/title/text() AND
        $x1/year/text() = $x1L1/year/text()
      RETURN
        <review> $x0L1/reviews/text() </review>
      <boxOffice> $x1/boxOffice/text() </boxOffice>
      <videoSales> $x1/videoSales/text() </videoSales>
    </Show>
  FOR $x0 IN $doc/DB/Episode, $x1 IN $doc/DB/Show, $x2 IN $doc/DB/Review
  WHERE $x0/EpisodeId/text() = $x1/ShowId/text() AND $x2/ReviewsId/text() = $x1/ShowId/text()
  RETURN
    <Show>
      <type> $x1/type/text() </type>
      <title> $x1/title/text() </title>
      <year> $x1/year/text() </year>
      FOR $x0L1 IN $doc/DB/Episode, $x1L1 IN $doc/DB/Show, $x2L1 IN $doc/DB/Review
      WHERE
        $x0L1/EpisodeId/text() = $x1L1/ShowId/text() AND $x2L1/ReviewsId/text() = $x1L1/ShowId/text() AND
        $x1/seasons/text() = $x1L1/seasons/text() AND $x1/description/text() = $x1L1/description/text() AND
        $x1/type/text() = $x1L1/type/text() AND $x1/title/text() = $x1L1/title/text() AND
        $x1/year/text() = $x1L1/year/text()
      RETURN
        <review> $x2L1/reviews/text() </review>
      <seasons> $x1/seasons/text() </seasons>
      <description> $x1/description/text() </description>
      FOR $x0L1 IN $doc/DB/Episode, $x1L1 IN $doc/DB/Show, $x2L1 IN $doc/DB/Review
      WHERE
        $x0L1/EpisodeId/text() = $x1L1/ShowId/text() AND $x2L1/ReviewsId/text() = $x1L1/ShowId/text() AND
        $x1/seasons/text() = $x1L1/seasons/text() AND $x1/description/text() = $x1L1/description/text() AND
        $x1/type/text() = $x1L1/type/text() AND $x1/title/text() = $x1L1/title/text() AND
        $x1/year/text() = $x1L1/year/text()
      RETURN
        <episode> $x0L1/name/text() </episode>
    </Show>
  </imdb>
```

Fig. 11. An initial mapping for the IMDB

relational database systems in order to be able to reuse their well-developed features (e.g., concurrency control, query processing, etc.). A number of approaches have been proposed to tackle the mismatch between the nested semistructured nature of the XML data and the relational model [8]. Unfortunately, no approach is universally accepted since none has been found to perform well in all cases. LegoDB [2] is a physical database design tool for designing (optimized) relational storage structures for XML data. LegoDB helps a user to evaluate some of the many XML-to-relational "shredding" options [8]. Since different XML-to-relational translations are best for different work loads and data characteristics, LegoDB provides an automated wizard for finding good relational designs. For this case study, we use the Internet movie database DTD example described in [2].

```
    <!ELEMENT imdb (show*, director*, actor*)>
<!ELEMENT show (title, year, reviews*,
        ((boxOffice, videoSales) |
            (seasons, description, episode*)))>
```

The two relational schemas of Fig. 10 represent two different shredding methods for the above DTD. Mappings between each of these two schemas and the DTD might be output by a design tool or might be created with a mapping tool like Clio. Although this is a very simple example, it is important to note that the mappings from the schema in Fig. 10a to the DTD are complicated enough. Figure 11 indicates the specific mapping in XQuery form and indicates the complexity of the mapping that has eight joins between the three participating tables of the source schema and a nesting depth of two. So even for this simple example, generating the correct mapping is clearly hard if it is to be done manually.

In tools like LegoDB, each shredding method is accompanied by the corresponding mapping. For the second shredding method of Fig. 10b, for example, the mapping to the DTD has to be generated in advance and hard-wired in the LegoDB cost engine. Manual generation of such mapping, or generation through a mapping tool, can be avoided with the use of ToMAS. Using ToMAS, one can take the DTD, the first shredding method, and the generated mapping of Fig. 11 and start modifying the schema to bring it in the form of Fig. 10b. Throughout this process, ToMAS will be maintaining the mappings, and at the end it will be able to provide automatically the mapping from the relational tables of the new shredding method to the DTD.

We have performed the above test. After most operations, the mappings could be automatically updated without user intervention, but for some operations there was a choice of what semantics to use. During the entire evolution, the ToMAS user had to make very few choices and we were able to verify that the resulting mapping is the one we would have created if it had been done manually, or using a mapping tool like Clio.

Notice that our approach permits design tools like LegoDB to explore new storage schemes that might not be part of their (predefined) search space of designs for efficiency reasons. For example, using ToMAS we can permit a designer to suggest a different, ad hoc, vertical or horizontal decomposition of the relations (one not suggested by the workload). If the cost-based engine of LegoDB selects such a user provided design, ToMAS can generate the mapping needed to use the original XML Schema as a view over this design.

Additionally, our ability to transform the relational schema of Fig. 10a to the one of Fig. 10b indicates the kind of transformations we are supporting. We do not consider only simple structural changes that take place locally on a table or on a class. We can have complex schema modifications involving many schema structures (in the specific example, relations) like copying an attribute from one table to another. In short, we have found that with our small set of primitive operators, we can support the majority of compound schema changes that exist in the literature [19].

9 Conclusion

In this paper, we identified the problem of mapping adaptation in dynamic environments with evolving schemas. We motivated the need for an automated system to adapt mappings and we described several areas in which our solutions can be applied. We presented a novel framework and tool that automatically maintains the consistency of the mappings as schemas evolve. Our approach is unique in many ways. We consider and manage a very general class of mappings including GLAV [18] mappings. We consider changes not only on the schema structure but also on the schema semantics (i.e., schema constraints) either in the source or in the target. Finally, we support schema changes that involve multiple schema elements (e.g., moving an attribute or subtree from one type to another).

We described the implementation of a mapping management and adaptation tool based on the above. We measured its performance and presented its application in two different domains, schema mapping and physical data design. A part that requires further investigation is the ranking mechanism, where an extensive experimentation is required with real users to investigate whether the rewritings getting the highest ranking are indeed those that the user would choose. The effectiveness of the ToMAS system can be extended by considering more complicated mappings like those that include "group by" clauses and aggregation functions, i.e., min, max, etc. Our long-term goal is to integrate the functionality of ToMAS with Clio and continue extending them in order to build an integrated metadata management tool. The development of such a tool will support data administrators by enabling the management of large complicated schemas and mappings with far less human supervision.

References

1. Abiteboul S, Duschka OM (1998) Complexity of answering queries using materialized views. In: PODS, pp 254–263
2. Bohannon P, Freire J, Haritsa JR, Ramanath M, Roy P, Siméon J (2002) LegoDB: Customizing relational storage for XML documents. In: VLDB, pp 1091–1094
3. Bertino E, Haas LM, Lindsay BG (1983) View management in distributed data base systems. In: VLDB, pp 376–378
4. Banerjee J, Kim W, Kim H, Korth HF (1987) Semantics and implementation of schema evolution in object-oriented databases. In: SIGMOD, pp 311–322
5. Bernstein P, Rahm E (2003) Data warehouse scenarios for model management. In: ER, pp 1–15
6. Claypool KT, Jin J, Rundensteiner EA (1998) SERF: Schema evolution through an extensible re-usable and flexible FRAMEWORK. In: CIKM, pp 314–321
7. Ceri S, Widom J (1991) Deriving production rules for incremental view maintenance. In: VLDB, pp 277–289
8. Florescu D, Kossmann D (1999) Storing and querying XML data using an RDMBS. IEEE Data Eng Bull 22(3):27–34
9. Fagin R, Kolaitis PG, Miller RJ, Popa L (2003) Data exchange: semantics and query answering. In: ICDT, pp 207–224
10. Fagin R, Kolaitis P, Popa L, Tan W (2004) Composing schema mappings: second-order dependencies to the rescue. In: PODS
11. Gyssens M, Lakshmanam L, Subramanian IN (1995) Tables as a paradigm for querying and restructuring. In: PODS, pp 93–103
12. Grahne G, Mendelzon AO (1999) Tableau techniques for querying information sources through global schemas. In: ICDT, pp 332–347
13. Gupta A, Mumick I, Ross K (1995) Adapting materialized views after redefinition. In: SIGMOD, pp 211–222
14. Halevy A, Ives Z, Suciu D, Tatarinov I (2003) Schema mediation in peer data management systems. In: ICDE, pp 505–517
15. Kantola M, Mannila H, Räihä K-J, Siirtola H (1992) Discovering functional and inclusion dependencies in relational databases. Int J Intell Sys 7(7):591–607
16. Kotidis Y, Roussopoulos N (1999) DynaMat: a dynamic view management system for data warehouses. In: SIGMOD, pp 371–382
17. Kotidis Y, Roussopoulos N (2001) A case for dynamic view management. ACM Trans Database Sys 26(4):388–423
18. Lenzerini M (2002) Data integration: a theoretical perspective. In: PODS, pp 233–246
19. Lerner BS (2000) A model for compound type changes encountered in schema evolution. ACM Trans Database Syst 25(1):83–127

20. Lee AJ, Nica A, Rundensteiner EA (2002) The EVE approach: view synchronization in dynamic distributed environments. Trans Knowl Data Eng 14(5):931–954
21. Levy AY, Rajaraman A, Ordille JJ (1996) Querying heterogeneous information sources using source descriptions. In: VLDB, pp 251–262
22. Madhavan J, Bernstein P, Rahm E (2001) Generic schema matching with Cupid. In: VLDB, pp 49–58
23. Mohania MK, Dong G (1996) Algorithms for adapting materialised views in data warehouses. In: CODAS, pp 309–316
24. Madhavan J, Halevy AY (2003) Composing mappings among data sources. In: VLDB
25. Miller RJ, Haas LM, Hernandez M (2003) Schema mapping as query discovery. In: VLDB, pp 77–88
26. Maier D, Mendelzon AO, Sagiv Y (1979) Testing implications of data dependencies. ACM Trans Database Syst 4(4):455–469
27. McBrien P, Poulovassilis A (2002) Schema evolution in heterogeneous database architectures, a schema transformation approach. In: CAiSE, pp 484–499
28. Mumick IS, Quass D, Mumick BS (1997) Maintenance of data cubes and summary tables in a warehouse. In: SIGMOD, pp 100–111
29. Melnik S, Rahm E, Bernstein P (2003) Rondo: a programming platform for generic model management. In: SIGMOD, pp 193–204
30. Popa L, Tannen V (1999) An equational chase for path-conjunctive queries, constraints, and views. In: ICDT, pp 39–57
31. Popa L, Velegrakis Y, Miller RJ, Hernandez MA, Fagin R (2002) Translating Web data. In: VLDB, pp 598–609
32. Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. VLDB J 10(4):334–350
33. Spaccapietra S, Parent C (1994) View integration: a step forward in solving structural conflicts. Trans Knowl Data Eng 6(2):258–274
34. Shamir R, Tsur D (1999) Faster subtree isomorphism. J Algorithms 33(2):267–280
35. Velegrakis Y (2004) Managing schema mappings in highly heterogeneous environments. PhD thesis, University of Toronto (in preparation)
36. Velegrakis Y, Miller RJ, Popa L (2003) Mapping adaptation under evolving schemas. In: VLDB, pp 584–595
37. Vassalos V, Papakonstantinou Y (1997) Describing and using query capabilities of heterogeneous sources. In: VLDB, pp 256–265
38. W3C (2001) XML Schema Part 0: Primer. http://www.w3.org/TR/xmlschema-0/, W3C Recommendation
39. Widom J (1995) Research problems in data warehousing. In: CIKM, pp 25–30