Mapping Adaptation under Evolving Schemas

Yannis Velegrakis[†] Renée J. Miller[†] Lucian Popa[‡]

[†]University of Toronto
{velgias,miller}@cs.toronto.edu

[‡]IBM Almaden Research Center
lucian@almaden.ibm.com

Abstract

To achieve interoperability, modern information systems and e-commerce applications use mappings to translate data from one representation to another. In dynamic environments like the Web, data sources may change not only their data but also their schemas, their semantics, and their query capabilities. Such changes must be reflected in the mappings. Mappings left inconsistent by a schema change have to be detected and updated. As large, complicated schemas become more prevalent, and as data is reused in more applications, manually maintaining mappings (even simple mappings like view definitions) is becoming impractical. We present a novel framework and a tool (ToMAS) for automatically adapting 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. We consider a comprehensive class of mappings for relational and XML schemas with choice types and (nested) constraints. Our algorithm detects mappings affected by a structural or constraint change and generates all the rewritings that are consistent with the semantics of the mapped schemas. Our approach explicitly models mapping choices made by a user and maintains these choices, whenever possible, as the schemas and mappings evolve. We describe an implementation of a mapping management and adaptation tool based on these ideas and compare it with a mapping generation tool.

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.

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

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 effort to recreate mappings as schemas change, but should instead reuse previous mappings. Furthermore, mapping creation, although aided tremendously by modern tools that suggest (syntactic) schema matches [22] and full (semantic) mappings [21], 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 [13]. 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 [23][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 [23]. Others will permit users to use a set of composable schema transformation operators to produce an integrated (transformed) schema (with a composed mapping) [10]. 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 [6, 7]. 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 inconsistency, 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 [4]. Such tools evaluate (or help a designer to evaluate) the relative cost of different physical relational designs. However, they consider only a fixed large 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 including modeling of source descriptions [15], modeling of query capabilities [24], and view management [3, 11]. 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. We call this problem *mapping adaptation* to differentiate it from view adaptation [9], view synchronization [12], and view maintenance [26]

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 using a matching tool to find corresponding portions of the two schema versions [22] and then use a mapping creation tool to add semantics to these correspondences [21]. This will produce a mapping from S' to S which 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. 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. (i) We motivate the problem of adapting mappings to schema changes and we present a simple and powerful model for representing schema changes. (ii) We consider changes not only to the structure of schemas (which may make the mapping syntactically incorrect [3]) but also to the schema semantics. The latter changes may make mappings semantically incorrect. (iii) 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. (iv) 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 in the base schema. (v) We support changes not only on atomic elements, but also on more complex structures including relational tables or complex (nested) XML structures. (vi) 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 was how to minimize the cost of updating the instance data when the schema was modified. Banerjee et al. [2] gave a taxonomy of the changes that may occur in OODBMS and provided an implementation for each one of them. Those changes were local to a single type, e.g., renaming an attribute or changing the position of a class in the class hierarchy. Lerner [14] extended the above work to include complex changes that span multiple classes and provided templates for the most common changes. None of this work investigated how views were affected when the schema is modified. *Incremental view maintenance* [5, 19] is a problem closely related to ours and deals with the methods for efficiently updating materialized views when the base schema data are updated. *View adaptation* [9, 18] 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. In AutoMed [17], schema evolution and integration are combined in one unified framework. Schema evolution is described as primitive changes that are each accompanied by a query that describes the semantics of the change. In our approach, we are trying to relieve the user from the task of manually specifying such queries. The EVE [12] 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 on the target schema 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 [22] is a common first step that generates a set of syntactic correspondences between portions of two schemas. A schema mapping tool like Clio [21] can take those correspondences and (by using the semantics embedded in the schemas) generate semantic mappings. Our approach complements the above scenario. 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.

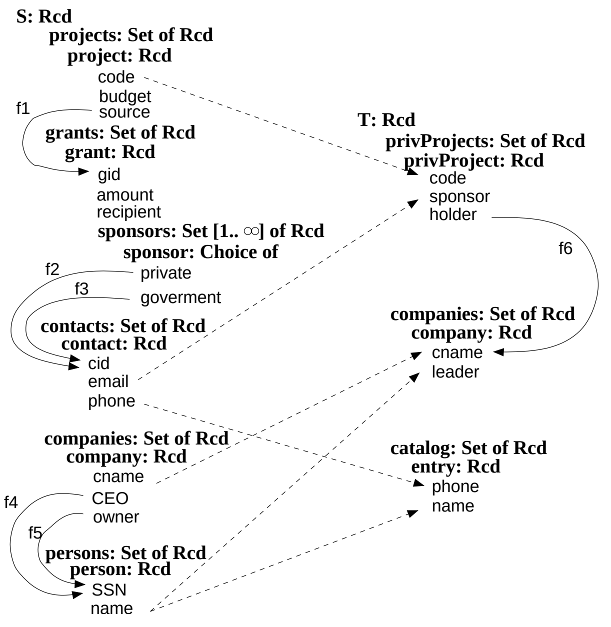
3 Mapping System

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 $\mathcal{T}$ [13]. 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 [13]. Potential type incompatibilities can be resolved through type transformation functions. Other types of mappings include *complete* ($Q^S \supseteq Q^T$) and *exact* ($Q^S{=}Q^T$) mappings [8]. 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) [13] and LAV (local-as-view) [15] views used in data integration systems, or the GLAV (global-and-local-as-view) mappings used in transforming data between independent schemas [21] and in data exchange [6].

Definition 3.1 *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 3.2 *Consider the mapping system of Figure 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 non-empty set of sponsors that are either private individuals or a government sponsors. Companies have an owner and a CEO. Relationships between different schema elements are specified via foreign keys (shown with*



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

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

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

Figure 1: A Mapping System.

solid lines in the figure). 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**. *The mappings are also expressed in a nested relational representation (defined formally below) that can easily be transformed to other representations [21], e.g., XQuery. Each mapping has the form $Q^S \rightsquigarrow Q^T$. In our notation, the* foreach *clause (with the associated* where*) defines Q^S while the* exists *clause (with the associated* where*) defines 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. We use the* with *clause to make explicit how the source and target elements relate to each other.*

Mapping m_2 specifies how to populate companies in the target schema with a company name and the owner name as the leader of the company. Mapping m_2 is a GAV mapping. Mapping m_1 populates the target with projects that receive *private funds and is a GLAV mapping. Note that m_1 respects the foreign key on the target and requires that for each project there should be an associated company with* `holder=cname`. *However, a specific value for this company* `cname` *is not specified (it is only required to exist). So m_1 constrains the target but does not completely specify a target instance (a property shared by many LAV and GLAV mappings). The third mapping m_3 generates catalog entries in the target by joining* `persons` *and* `contacts` *in the source.* ■

Schemas. 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. A *schema* is a set of labels (called roots), each with an associated type. For example, S and T in Figure 1 are such roots for the source and target schema respectively. A type τ is defined by the grammar: τ::= String | Int | SetOf τ | Rcd$[l_1{:}\tau_1, \ldots, l_n{:}\tau_n]$ | Choice$[l_1 : \tau_1, \ldots, l_n{:}\tau_n]$. Types Int and $String$ are called atomic types, Set is a collection type and the types Rcd and $Choice$ are complex types. 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*. For each set type SetOf τ, τ may be an atomic (String or Int) type, a choice or a record type. We do not consider order. A Set type represents an unordered set. An XML-Schema "sequence" is modeled as a (unordered) Rcd type (as with "all").

For queries we adopt the OQL *select-from-where* syntax [1] 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. Queries have the following form where e_i, c_i and c'_i are expressions containing only variables x_i that are bound in the from clause:

select $e_0, e_1, ..., e_m$
from x_0 in P_0, x_1 in $P_1, ... x_n$ in P_n
where $c_0{=}c'_0$ and $c_1{=}c'_1$ and ... and $c_k{=}c'_k$

Each P_i in the from clause is either: (1) an expression e with type SetOf τ; in this case, the variable x_i will bind to individual elements of the set e, or (2) $e{\to}l$ (where e is an expression with a type Choice $[\ldots, l : \tau, \ldots]$) representing the selection of attribute l of the expression e; in this case, the variable x_i will bind to the element (if any) of type τ under the choice l of e. The query is well formed if the variable (if any) used in P_i is defined by a previous in clause. The conditions in the where clause are optional. The '*' symbol can be used in the select clause to denote all the valid expressions with an atomic type that can be in the select clause. In mappings of the form $Q^S \rightsquigarrow Q^T$ the select clause of the queries is a '*' and has been omitted for better readability.

Following XQuery and OQL convention, we will use queries to identify elements within schemas. A schema element is identified with the path query that can be used (intuitively) to retrieve all the instances of that element.

Definition 3.3 *A* **schema element** *in schema S is a path query, that is a query of the form:*

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

where each P_k with $k{\geq}1$ uses variable x_{k-1}, P_0 is an expression starting at a schema root in S and expression e_{n+1} uses variable x_n.

If the details of the from clause are unimportant, we refer to a schema element using the notation select e from P.

Example 3.4 *For the source schema in Figure 1, the schema elements* `amount` *and* `private` *under* `grant` *are formally defined, respectively, by the following two path queries:*

a_1 select g.grant.amount from g in S.grants
a_2 select s from g in S.grants, n in g.grant.sponsors
s in n.sponsor→private ■

Constraints. For *schema constraints* we consider a very general form of referential constraints called *nested referential integrity constraints (NRIs)* [21] extended to support choice types. NRIs capture naturally relational foreign key constraints as well as the more general XML Schema keyref constraints.

Example 3.5 *The foreign key f_2 on the source schema of Figure 1 is expressed as the following NRI.*

f_2: foreach g in S.grants, n in g.grant.sponsors
r in n.sponsor→private
exists c in S.contacts
where c.contact.cid= r ■

The simplest form of NRI relates two schema elements select e_1 from P_1 and select e_2 from P_2. Such a constraint has the form foreach P_1 exists P_2 with $e_1 = e_2$. This is a simple unary inclusion constraint. More generally, and as in XML Schema, an NRI is relative (i.e., local) to a given schema element (the "context" element) select e_0 from P_0. Hence, an NRI has the general form: foreach P_0 [foreach P_1 exists P_2 with C], where P_1 and P_2 are now relative to (i.e, start from) the last variable of P_0. In this expression, C is a conjunction of one or more equalities $e_1 = e_2$ where e_1 and e_2 are expressions that use the last variable of P_1 and P_2, respectively. Note that such constraints can also be written as foreach X exists Y with C, where Y may be relative to some variable of X.

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 [21] in which mappings are created based on the semantics of the schemas. While Section 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 4.1 *An* **association** *is a query on schema S:*

select * from x_1 in P_1, x_2 in $P_2, ... x_n$ in P_n
where $e_1{=}e'_1$ and $e_2{=}e'_2$ and ... and $e_n{=}e'_n$

Example 4.2 *To identify the element* `private` *in Schema* **S***, we use the query* a_2 *from Example 3.4. The following (similar) query defines an association containing not only* `private`*, but also the atomic elements* `gid`, `amount`, *and* `recipient`.

```
A2  select  *   from  g in S.grants, n in g.grant.sponsors
                       s in n.sponsor→private
```
■

Associations are simply collections of atomic elements and can be used in mappings. So for our schemas, mappings are simply very general referential constraints between a source and target association.

Definition 4.3 *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 non-empty conjunction of equality conditions relating atomic type expressions over* S *with atomic type expressions over* T.

Correspondences. A *correspondence* is a specification that describes how the value of an atomic target schema element is generated from the source. A correspondence can be represented as simple inter-schema referential constraints. A correspondence from a source element select e^S from P^S to a target element select e^T from P^T is an inter-schema NRI foreach P^S exists P^T with $e^S{=}e^T$. Correspondences are implicit in the mappings (and view definitions) and can be easily extracted from them.

Example 4.4 *The correspondence between the* `name` *in the source schema and the* `leader` *in the target (depicted in Figure 1 with a dotted line) is:*

```
v: foreach e  in S.persons
   exists  c  in T.companies
   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 renaming functions to express a form of query subsumption between associations.

Definition 4.5 *An association* A *is* **dominated** *by association* B *(noted as* $A \dot{\preceq} B$*) if there is a renaming 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 as $m_1 \dot{\preceq} m_2$) if $A_1^S \dot{\preceq} A_2^S$, $A_1^T \dot{\preceq} A_2^T$ and for every equality condition $e{=}e'$ 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.

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 parent-child relationship in nested schemas. 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 [21] where it is shown that they can be computed by one time traversal over the schema.

```
P1S : select * from p in S.projects
P2S : select * from g in S.grants, n in g.grant.sponsors
                    r in n.sponsor→private
P3S : select * from g in S.grants, n in g.grant.sponsors
                    m in n.sponsor→goverment
P4S : select * from a in S.contacts
P5S : select * from c in S.companies
P6S : select * from i in S.persons

P1T : select * from p in T.privProjects
P2T : select * from p in T.companies
P3T : select * from p in T.persons
```

Figure 2: Source and target structural associations

Definition 4.6 *A* **structural association** *is an association*

select $*$ from x_1 in P_1, x_2 in $P_2, \ldots x_n$ in P_n

with no where *clause and where the expression* P_1 *must start at a schema root and every expression* P_k*,* $k{>}0$ *starts with variable* x_{k-1}.

The schema structure encodes a set of semantic relationships that a designer chose to model explicitly. A second way of encoding semantic associations is in a mapping. A mapping is an encoding of a pair of source and target associations (which may or may not be explicitly present in the schema structure). A mapping may use associations provided by a user or mapping tool. Such mappings may expose hidden semantic relations between schema elements.

Definition 4.7 *Let* M *be a set of given mappings. A* **user association** *is an association that has been provided to the system via a mapping* $m{\in}M$.

Example 4.8 *Mapping* m_3 *joins* `contacts` *and* `persons` *based on the* `SSN` *and* `cid` *which indicates that a person can be associated with its contact information through the SSN. This generates the user association:*

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

A third way to semantically relate elements is through schema constraints. Chasing is a classical relational method [16] 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, in which case the Y clause and the C conditions (under the respective renaming) are added to the association. The chase can be used to enumerate logical join paths, based on the set of dependencies in a schema. We use a variation of a nested chase [20] that can handle choice types and NRIs. Our extensions to the chase are defined formally in an extended version of this paper [25].

Definition 4.9 *A* **logical association** R *is the result of chasing a structural or a user association* P *with the set* $\mathcal{X}$ *of all the NRIs of the schema (denoted as* $chase_{\mathcal{X}}(P)$*).*[1]

Example 4.10 *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*

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

```
$A_1$: select  *
       from    p in S.projects, g in S.grants,
               n in g.grant.sponsors
               r in n.sponsor→private, c in S.contacts
       where   c.cid=r and g.gid=p.source
$A_2$: select  *
       from    p in S.projects, g in S.grants,
               n in g.grant.sponsors
               r in n.sponsor→goverment, c in S.contacts
       where   c.cid=r and g.gid=p.source
$A_3$: select  *
       from    g in S.grants, c in S.contacts,
               n in g.grant.sponsors
               r in n.sponsor→private,
       where   c.cid=r
$A_4$: select  *
       from    g in S.grants, c in S.contacts,
               n in g.grant.sponsors
               r in n.sponsor→goverment
       where   c.cid=r
$A_5$: select  * from c in S.contacts
$A_6$: select  *
       from    c in S.companies, p in S.persons, w in S.persons
       where   c.company.CEO=p.person.SSN and
               c.company.owner=w.person.SSN
$A_7$: select  * from c in S.persons
$A_8$: select  *
       from    c in S.contacts, p in S.persons,
       where   p.contact.cid=p.person.SSN

$B_1$: select  *
       from    p in T.privProjects, c in T.companies
       where   p.privProject.holder=c.company.cname
$B_2$: select  * from c in T.companies
$B_3$: select  * from c in T.persons
```

Figure 3: Logical associations for schemas S and T and the set of mappings m_1, m_2, m_3.

three elements form the structural association P_5^S of Figure 2. This figure gives all the structural associations of the schemas in Figure 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 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 the constraints f_5 and f_4 results in the logical association A_6 of Figure 3. Logical associations A_1 to A_7 and B_1 to B_3 are the logical associations that resulted from the chase of structural associations of Figure 2 with all the constraints that are defined on the two schemas. Finally, the user association we mentioned in Example 4.8 cannot be chased with any constraints, thus, it is a logical association (indicated in Figure 3 as A_8). ■

We can now give a formal definition of a semantically valid mapping:

Definition 4.11 *Let $\mathcal{S}$ and $\mathcal{T}$ be a pair of source and target schemas and M a set of mappings between them. Consider C to be the set of correspondences specified by mappings in M. A* **semantically valid mapping** *is an expression of the form* foreach A^S exists A^T with D*, where A^S and A^T are logical associations in the source and the target schema correspondingly, 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). A correspondence v:* foreach P^S exists P^T with D **is covered** *by the associations $<A^S, A^T>$ if $P^S \dot{\preceq} A^S$ and $P^T \dot{\preceq} A^T$.*

Example 4.12 *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 c in S.companies, p in S.persons, p' in S.persons
         where c.company.CEO=p.person.SSN and
               c.company.owner=p'.person.SSN
       exists  o in T.companies
       with    o.company.cname=c.company.cname and
               o.company.leader=p.person.name
```

which is a semantically valid mapping that is equivalent to

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

The second way gives mapping m_2 of Figure 1. The difference between m_o and m_2 is that m_o takes the CEO to be the leader of the company while m_2 takes 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 semantically valid mappings set includes the mappings produced by the mapping generation tool Clio [21] 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 when the schemas change.

Definition 4.13 *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 *$\mathcal{U}_{\mathcal{S},\mathcal{T}}^{\mathcal{M}}$ 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 [21]. We provide algorithms to efficiently (re)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 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 described by the mappings $\mathcal{M}$ are first extracted and then the mappings $\mathcal{M}$ 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 $A_1^S, ..., A_n^S$ and $A_1^T, ..., A_l^T$ respectively. This brings in additional joins that the user may not have been known existed. For each pair $<A_i^S, A_j^T>$ a new mapping m_{ij} of the form foreach A_i^S exists A_j^T with D' is created, where D' includes the conditions D, plus the conditions of all the correspondences that are covered by the pair of larger associations $<A_i^S, A_j^T>$ but were not covered before. Note that the set of all those semantically valid mappings m_{ij} is a subset of the mapping universe $\mathcal{U}_{\mathcal{S},\mathcal{T}}^{\mathcal{M}}$.

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 may occur in the source or the target schema, each mapping is modified as appropriate. This is done for each mapping independently. 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 extent 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 return the set of adapted semantically valid mappings $\mathcal{M}'$. We have identified the number of primitive schema changes that are usually met in practice and we have categorized them in three main 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 in which 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 4.13) 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 extracted from the mappings $\mathcal{M}$. 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 requirements of the schemas. 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 $X \cup Y \cup C$ to A^S. In other words, the addition of the new constraint caused A^S not be closed under the chase. A^S is not 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 adapt, 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 since the result of that chase step may allow further chasing with some old constraints that was not possible before. 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 correspondences in $\mathcal{C}$ that are covered by the pair $<A, A^T>$. Since A is generally a larger logical association than A^S, 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 5.1 gives a brief description of the steps taken when a new constraint is added.

Example 5.1 *Assume that the following new constraint is added in the source schema of Figure 1:*

```
f7: foreach g  in S.grants
    exists  c  in S.companies
    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 that 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 needs 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 but the union of the* exists *and* with *clause is not. When chased with the new set of constraints that includes F, association A_1 gives a new logical association:*

```
A1a: select  *  from p  in S.projects, g in S.grants,
                n   in g.grant.sponsors
                r   in n.sponsor→private, c in S.contacts
                o   in S.companies, e in S.persons,
                e'  in S.persons
         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 the way the value of `leader` *in the target can be obtained: as the name of the* `CEO` *of a company or as the name of the* `owner` *of the company. (In Algorithm 5.1 terms, we say that there are two coverages of the pair $<A_{1a}, B_1>$ by the correspondence from person name to company leader. The first rewriting (after a step of join minimization) is:*

```
m'_1a: foreach p in S.projects, g in S.grants,
               n in g.grant.sponsors
               r in n.sponsor→private, c in S.contacts
               o in S.companies, e in S.persons
         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  j in T.privProjects, m in T.companies
         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} apart from 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.* ■

Choosing one mapping rewriting in favor of another cannot always be done using the available information. All the rewritings are consistent with the new schema and the previously defined mappings (i.e., they are valid members of the new mapping universe).

A special, yet interesting, case is when the chase will not introduce any new schema elements in the association but only some extra conditions. Those conditions will introduce new ways (join paths actually) to relate the elements in the association. Despite the fact that this adds new semantically valid mappings to the mapping universe, none of the existing mappings is adapted and no new mapping is generated. The intuition behind this is that there is no indication that the new mappings are preferred over the existing mappings. Neither the existing mappings nor the schemas and constraints can specify that. Hence, since our goal is to maintain the semantics of existing mappings as much as possible, we perform no adaptation unless necessary.

Algorithm 5.1 - Constraint addition

```
Input:   Set of mappings $\mathcal{M}$
         New constraint $F$: foreach $X$ exists $Y$ with $C$ in schema $\mathcal{S}$
Body:    $\mathcal{X} \leftarrow$ constraints in $\mathcal{S}$,  $\mathcal{M}' \leftarrow \emptyset$
         $\mathcal{C} \leftarrow$ compute correspondences from $\mathcal{M}$
         For every m$\leftarrow$(foreach $A^S$ exists $A^T$ with $D$) $\in \mathcal{M}$
            if ($X \dot{\preceq} A^S$ with renaming $h$ and $h(X \cup Y \cup C) \dot{\npreceq} A^S$)
              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 $\mathcal{M}'$
```

Removing Constraints: Similarly to adding a constraint, removing one 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 $X \cup Y \cup C \dot{\preceq} A^S$.

Once we detect that a mapping m needs to be adapted, we apply the following steps. (Algorithm 5.2 provides a succinct description of these steps.) The intuition of Algorithm 5.2 is to take the maximal independent sets of semantically associated schema elements of the affected association used by the mapping. We start by breaking apart the source association A^S into its set $\mathcal{P}$ of structural associations, that is, we enumerate all the structural 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 conditions 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^*. The following example illustrates the algorithm.

Example 5.2 *Consider the mappings m'_{1a}, m'_{1b}, m_2 and m_3 in Example 5.1 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 5.1, is broken apart into the structural associations (recall Figure 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 Figure 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* private *sponsor independently of the existence of a* goverment *sponsor. The chase of P_2^S, P_4^S, P_5^S and P_6^S will result to associations A_3, A_5, A_6 and A_7 respectively. Each of the resulting associations will be used to form a new mapping. A_1 generates mapping m_1 (which is the one we started with in Example 5.1). Association A_6 generates mappings (recall Example 4.12) 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 map-*

ping m_1, while the one generated by A_7 is dominated by m_2. Hence, those mappings are eliminated and the final result of the algorithm, for the case of m'_{1a}, consists of the mappings m_1 and m_2. ■

```
Algorithm 5.2  - Constraint Removal

Input:   Set of semantically valid mappings $\mathcal{M}$
         Constraint $F$: $\underline{\text{foreach}}\ X\ \underline{\text{exists}}\ Y\ \underline{\text{with}}\ C$ of schema $\mathcal{S}$
Body:    $\mathcal{X} \leftarrow$ constraints in $\mathcal{S}$,    $\mathcal{M}' \leftarrow \emptyset$
         For every $m \leftarrow (\underline{\text{foreach}}\ A^S\ \underline{\text{exists}}\ A^T\ \underline{\text{with}}\ D) \in \mathcal{M}$
             If $(X \cup Y \cup C\ \dot{\preceq}\ A^S)$ {
               $\mathcal{P} \leftarrow \{P \mid P$ structural association $\wedge\ P\ \dot{\preceq}\ A^S\}$
               $\mathcal{A}' \leftarrow \{A \mid A \in chase_{\mathcal{X}-\{F\}}(P)\ \wedge\ P \in \mathcal{P}\ \wedge\ A \dot{\preceq} A^S\}$
               $\mathcal{M}^* \leftarrow \emptyset$
               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 $D$ }
                 $m_a \leftarrow (\underline{\text{foreach}}\ A_a\ \underline{\text{exists}}\ A^T\ \underline{\text{with}}\ D')$
                 $\mathcal{M}^* \leftarrow \mathcal{M}^* \cup \{m_a\}$
               $\mathcal{M}^{**} \leftarrow \{m' \mid m' \in \mathcal{M}^*\ \wedge\ \nexists\, m'' \in \mathcal{M}^*: m' \dot{\preceq} m''\}$
               $\mathcal{M}' \leftarrow \mathcal{M}' \cup \mathcal{M}^{**}$
             else include $m$ in $\mathcal{M}'$
Output:  New set of mappings $\mathcal{M}'$
```

5.2 Schema pruning or expansion

Among the most common changes that are 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, hence the mapping universe is expanded. However, they are not added in 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 5.3 *Consider the case in which the source schema of Figure 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 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, continues to be 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 any more, 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. An element is used in a mapping because it participates either in a correspondence or in a constraint (or both). In the relational world this is equivalent to attributes and relations that are used in the select clause of a view definition query or in the where clause as parts of a join path. We consider first the removal of atomic type elements.

An atomic element e: $\underline{\text{select}}\ e_{n+1}\ \underline{\text{from}}\ x_0\ \underline{\text{in}}\ P_0, x_1\ \underline{\text{in}}\ P_1, \ldots\ x_n\ \underline{\text{in}}\ P_n$ is used in constraint F: $\underline{\text{foreach}}\ X\ \underline{\text{exists}}\ Y\ \underline{\text{with}}\ C$ if there is a renaming function f from the variables of e to the variables of F and expression $f(e_{n+1})$ is used in the condition C of F. A similar condition applies for an element to be used in an association. When atomic element e is removed each constraint F in which e is used is removed by following the procedure described in Section 5.1. Similarly, an atomic element e participates in a correspondence V: $\underline{\text{foreach}}\ P^S\ \underline{\text{exists}}\ P^T\ \underline{\text{with}}\ C$ if there is a renaming function g from the variables of e to the variables of V and $g(e_{n+1})$ is used in the condition C. 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 condition 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 5.3 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 5.3. A complex type element can be removed if all its attributes (children) have been removed.

```
Algorithm 5.3  - Atomic Element Deletion

Input:   Mapping System $<\mathcal{S}, \mathcal{T}, \mathcal{M}>$
         Atomic element $e$
Body:    While exists constraint $F$ that uses $e$
             remove $F$
         $\forall\ m \leftarrow (\underline{\text{foreach}}\ A^S\ \underline{\text{exists}}\ A^T\ \underline{\text{with}}\ D) \in M$
             $D \leftarrow \{q \mid c$ 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 $M$
```

Example 5.4 *In the mapping system of Figure 1 removing element* `topic` *from* `project` *will not affect any mappings since it is neither used in a constraint, nor in a correspondence. On the other hand, removing* `code` *will invalidate mapping m_1 that 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 condition 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

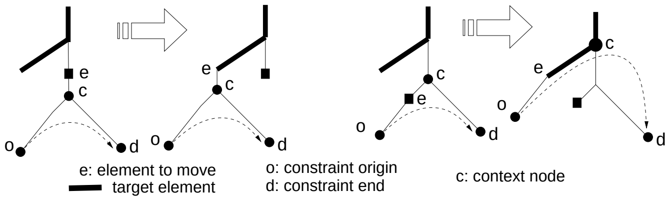


Figure 4: Updating constraint when element moves

not be considered seperately since it can be shown that this is equivalent to removing element e and then adding one of type t and with the same name as 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, copy, and move. 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 is carrying 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 one.

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 become invalid and must be adapted. A constraint F uses element e if there is a renaming from the variables of the path query P_e that identifies e to the variables of F. To realize how F is affected by the change, we have to consider the relative position of e with respect to the context element of F. (In more technical terms, we have to consider where the image of the last variable of the path P_e, under the above mentioned renaming, is within F.) Recall that F has the form foreach P_0 [foreach P_1 exists P_2 with C] where the path queries P_1 and P_2 start from the last variable of the path query P_0 which represents the context node. Moreover, C is of the form $e_1{=}e_2$ where e_1 and e_2 are expressions depending on the last variable of path P_1, respectively, P_2. Figure 4 provides a graphical explanation of how F has to be adapted to the move of the element e. In the figure, for constraint F, we use c, o, and d to denote (both before and after the move) the context element, the element identified by the path select e_1 from P_0, P_1 (also called the origin element) and, respectively, the element identified by the path 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 the path to the new location of the context node c. The path 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 either used in P_1 or in P_2. Assume that it is used in P_1 (the other case is symmetric). This case is shown in the second part of the figure. Then the 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' is the path to the new context node and P_1' and P_2' are the relative paths from P_0' to (the new location of) o and d. The condition C' is the result of changing C so that it uses the end points of paths P_1' and P_2'

Example 5.5 *Assume that the schema owner of schema S in the mapping system of Figure 1 has decided to store the grants nested within each company so that each company keeps its own grants. This translates to a move of the element* `grants` *under the element* `company`. *Consider the constraint f_2 of Example 3.5 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 Figure 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 *a* in S.companies

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 path in the* exists *clause need not be changed, since the new context element continues to be the root.)*

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

Adapting mappings. When an element is moved to a new location, some of the old logical associations that were using it become invalid and new ones have to be generated. To avoid redundant recomputations by regenerating every association, we exploit information given by existing mappings and computations that have already been performed. In particular, we first identify the mappings that need to adapt 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 x_0 in e_0, x_1 in e_1, ..., x_n in e_n

t = select t_{m+1} from y_0 in t_0, y_1 in t_1, ..., y_m in t_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 is 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 in use by the original mapping and that are still well-formed are still 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.

Example 5.6 *Assume that in the mapping system of Figure 1* grants *are moved under* company *as in Example 5.5. This change affects neither mapping m_3, nor mapping m_2. (Recall from Section 5.2 that just the addition of 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 (see Figure 3), is affected. First, schema constraints are adapted as described in Example 5.5. Then we run the mapping adaption algorithm described above, for e:* select S.grants from _ *and t:* select o.company.grants from o in S.companies *(we denote here by _ the empty* from *clause). The clause* o in S.companies *is added in 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 o in S.companies, p in S.projects,
               g in o.company.grants, n in g.grant.sponsors
               r in n.sponsor→private,
               c in S.contacts, e in S.persons,
         where p.project.source=g.grant.gid and
               r=c.contact.cid and
               o.company.CEO=e.person.SSN
       exists  j in T.privProjects, m in T.companies
         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 a 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 that the element is copied instead of being moved, the same reasoning takes place 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.

6 Mapping adaptation experience

To evaluate the effectiveness and usefulness of our approach, we have implemented a prototype tool called ToMAS[2] and we have applied it to a variety of real application scenarios. The experiments were conducted on a number of publicly available schemas that vary in terms of size and complexity. Their characteristics are summarized in Table 1. The size is shown in terms of 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. The different versions of each schema were either 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 the set 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.

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 realize how the mappings were affected by those changes and how they should adapt. 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 re-infer 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 meaningful mappings and the user had to browse through all of them to find those that were describing the initial semantics. The second alternative was to perform the schema changes and let ToMAS handle the maintenance of the mappings. ToMAS returns only a small number of mappings since it utilizes knowledge about choices that were embedded in the initial set of mappings. At the end, the user would have to go through only the small number of adapted mappings and verify their correctness. We performed and compared both techniques experimentally. In terms of performance, ToMAS made the computations in time that is very close to the time of Clio as reported in [21], even though it uses none of the auxiliary structures that Clio does (which means that every computation had to be made on demand every time it was needed). We also compared the user effort required in the two approaches. In the first approach where

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

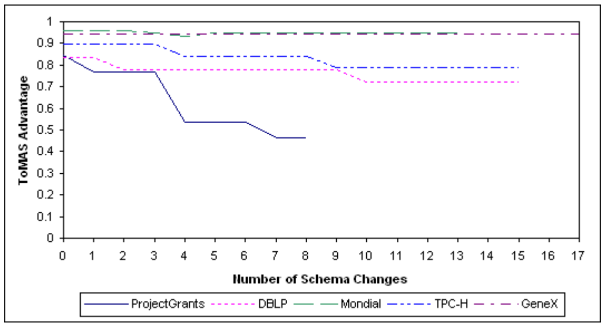


Figure 5: Benefit of ToMAS use

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

Table 1: Test schemas characteristics

mappings have to be regenerated from scratch, the effort of the user was measured as the number of correspondences that have to be re-specified, 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 that specifies the advantage of ToMAS against the "from-scratch" approach. A value 0.5, for example, means that ToMAS requires half of the effort required with the other alternative.

$$1 - \frac{mappings\ generated\ by\ ToMAS}{mappings\ generated\ by\ Clio + correspondences}$$

Figure 5 provides a graphical representation of how the above quantity changes for the various schemas during our experiments as a function of the number of changes. It can be noticed that as the number of changes becomes larger, and the modified schemas become much different than their original version, the advantage of ToMAS is reduced. Furthermore, we have noticed that as the number of mappings that are to be maintained becomes closer to the number of all the semantically meaningful mappings that exist, ToMAS also becomes less preferable. However, the rate of reduction is small and in practice schemas do not change radically. The new evolved schemas are not dramatically different from their original version and the number of mappings that are to be maintained is relatively small. In these cases, ToMAS would be the right tool to use.

7 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 [13] 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 invlove multiple schema elements (e.g., moving an attribute or subtree from one type to another).

References

[1] F. Bancilhon, S. Cluet, and C. Delobel. *A Query language for O_2*, chapter 11. Morgan Kaufman, 1992.

[2] J. Banerjee, W. Kim, H. Kim, and H. F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *SIGMOD*, pages 311–322, May 1987.

[3] E. Bertino, L. M. Haas, and B. G. Lindsay. View management in distributed data base systems. In *VLDB*, pages 376–378, 1983.

[4] P. Bohannon, J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Siméon. LegoDB: Customizing Relational Storage for XML Documents. In *VLDB*, pages 1091–1094, 2002.

[5] S. Ceri and J. Widom. Deriving Production Rules for Incremental View Maintenance. In *VLDB*, pages 277–289, September 1991.

[6] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, pages 207–224, 2003.

[7] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: Getting to the core. In *PODS*, pages 90–101, 2003.

[8] Gösta Grahne and Alberto O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. In *ICDT*, pages 332–347, 1999.

[9] A. Gupta, I. Mumick, and K. Ross. Adapting Materialized Views After Redefinition. In *SIGMOD*, pages 211–222, 1995.

[10] M. Gyssens, L. Lakshmanam, and I. N. Subramanian. Tables as a Paradigm for Querying and Restructuring. In *PODS*, pages 93–103, 1995.

[11] Y. Kotidis and N. Roussopoulos. A case for dynamic view management. *ACM TODS*, 26(4):388–423, 2001.

[12] A. J. Lee, A. Nica, and E. A. Rundensteiner. The EVE Approach: View Synchronization in Dynamic Distributed Environments. *TKDE*, 14(5):931–954, 2002.

[13] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[14] B. S. Lerner. A Model for Compound Type Changes Encountered in Schema Evolution. *ACM TODS*, 25(1):83–127, March 2000.

[15] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, pages 251–262, 1996.

[16] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM TODS*, 4(4):455–469, 1979.

[17] P. McBrien and A. Poulovassilis. Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. In *CAiSE*, pages 484–499, 2002.

[18] M. K. Mohania and G. Dong. Algorithms for Adapting Materialised Views in Data Warehouses. In *CODAS*, pages 309–316, December 1996.

[19] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of Data Cubes and Summary Tables in a Warehouse. In *SIGMOD*, pages 100–111, May 13–15 1997.

[20] L. Popa and V. Tannen. An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In *ICDT*, pages 39–57, 1999.

[21] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, August 2002.

[22] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

[23] S. Spaccapietra and C. Parent. View Integration : A Step Forward in Solving Structural Conflicts. *TKDE*, 6(2):258–274, 1994.

[24] V. Vassalos and Y. Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *VLDB*, pages 256–265, 1997.

[25] Y. Velegrakis, R. J. Miller, and L. Popa. Adapting mappings in frequently changing environments. Technical Report CSRG-468, Univ. of Toronto, Dep. of Comp. Sc., February 2003. ftp://ftp.cs.toronto.edu/cs/ftp/pub/reports/csri/468.

[26] J. Widom. Research Problems in Data Warehousing. In *CIKM*, pages 25–30, Baltimore, Maryland, 1995.