Preserving mapping consistency under schema changes

Yannis Velegrakis¹, Renée J. Miller¹, Lucian Popa²

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)

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 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