Round-Trip Engineering for Maintaining Conceptual-Relational Mappings

Yuan An, Xiaohua Hu, and Il-Yeol Song

College of Info. Sci. and Tech., Drexel University, USA
{yan,thu,isong}@ischool.drexel.edu

Abstract. Conceptual-relational mappings between conceptual models and relational schemas have been used increasingly to achieve interoperability or overcome impedance mismatch in modern data-centric applications. However, both schemas and conceptual models evolve over time to accommodate new information needs. When the conceptual model (CM) or the schema associated with a mapping evolved, the mapping needs to be updated to reflect the new semantics in the CM/schema. In this paper, we propose a round-trip engineering solution which essentially synchronizes models by keeping them consistent for maintaining conceptual-relational mappings. First, we define the consistency of a conceptual-relational mapping through "semantically compatible" instances. Next, we carefully analyze the knowledge encoded in the standard database design process and develop round-trip algorithms for maintaining the consistency of conceptual-relational mappings under evolution. Finally, we conduct a comprehensive set of experiments. The results show that our solution is efficient and provides significant benefits in comparison to the mapping reconstructing approach.
[keywords: Round-trip Engineering, Mapping Maintenance]

1 Introduction

Modern data-centric applications increasingly rely on mappings between conceptual models and relational schemas, i.e., *conceptual-relational mappings (a.k.a., object-relational mappings)*, to achieve interoperability [5] or to overcome the well-known *impedance mismatch* problem [15]: the differences between the data model exposed by databases and the modeling capabilities and programmability needed by the application. Essentially, a conceptual-relational mapping specifies a *semantically consistent* relationship between a conceptual model (hereafter, CM) and a relational schema. For example, a many-to-one relationship from an entity E_1 to an entity E_2 in an Entity-Relationship (ER) diagram can be mapped using some mapping formalism to a relational table that uses the identifier of E_1 as the key and referring to the identifier of E_2 as a foreign key [15]. The key and foreign key constraints reflect the semantics encoded in the relationship.

However, conceptual models and schemas evolve over time to accommodate the changes in the information they represent. Such evolution causes the existing conceptual-relational mappings inconsistent. For example, if the database administrator (DBA) in charge of the aforementioned relational table has changed the key of the table from the identifier of E_1 to the combination of the identifiers of E_1 and E_2 due to new requirements of the application, then the many-to-one relationship from E_1 to E_2 in the ER diagram is *semantically inconsistent* with the new table because some instances of the table may violate the many-to-one relationship. When conceptual models and schemas changed, the conceptual-relational mappings between the conceptual models and schemas must be updated to reflect the evolution. This process is called *conceptual-relational mapping maintenance under evolution*, or *mapping maintenance* for short.

A typical solution to the mapping maintenance problem is to regenerate the conceptual-relational mapping. However, there are two major problems: first, regenerating the mapping alone sometimes cannot solve the inconsistency problem because the semantics of the conceptual model and the schema are out of synchronization, as shown by the previous example; second, the mapping generation process, even with the help of mapping generation tools [7, 6], can be costly in terms of human effort and expertise, especially for complex CMs and schemas that were developed independently. Moreover, there is no guarantee that the regenerated mappings preserve the semantics of the original mappings. A better solution is to design algorithms that synchronize the CMs and schemas and reuse the original mappings to (semi-)automatically update them into a set of new mappings that are consistent with respect to the new CMs and schemas.

The process for synchronizing models by keeping them consistent is called *Round-Trip Engineering* (RTE) [26, 20]. RTE offers a bi-directional exchange between two models. Changes to one model must at some point be reconciled with the other model. In this paper, we propose a round-trip engineering approach for maintaining the consistency of conceptual-relational mappings. Notice that round-trip engineering is **not** forward engineering, e.g., generating a relational schema from a CM, plus reverse engineering [19], e.g., generating a **new** CM from an existing schema. RTE focuses on synchronization.

1.1 Motivation

To motivate our work, we first consider a number of applications and environments in which conceptual-relational mappings are used extensively and a solution to the mapping maintenance problem will greatly benefit to the applications.

Database Design. A typical database design process begins with the development of a conceptual model such as an ER diagram and ends up with a logical database schema manipulated by a commercial database management system. Although the process of generating a logical schema from a CM is mostly automated, the translation mappings between CMs and logical schemas are not kept in automated tools, and the CMs and logical schemas may evolve independently causing the "legacy data" problem. Saving the mappings between CMs and logical schemas implied by the database design process and maintaining the mappings when CMs and schemas evolve will help reduce the "legacy data".

Data-Centric Applications. To increase the productivity of the developers of these applications, there are a number of middleware mapping technologies such as Hibernate [10], DB Visual Architect [1], Oracle TopLink [2], and Microsoft ADO.NET [4]. They provide an ease-to-use environment for generating conceptual-relational mappings. In these middleware mapping tools, when the object/conceptual models and the database schemas change, a solution is needed for maintaining the conceptual-relational mappings.

Data Integration. In data integration, a set of heterogeneous data sources are queried and accessed through a unified global and virtual view [22]. There are many ontology-based data integration applications [12, 23] which use ontologies as their global views. For these applications, the mappings between ontologies and local data sources are the main vehicle for data integration. Early studies have been focused on integration architectures [22], query answering capabilities [3], and global view integration [27]. What has been missing is a solution to maintaining the mappings between ontologies and local data sources when ontologies and database schemas evolve.

The Semantic Web. On the Semantic Web [28], data is annotated with ontologies having precise semantics. For the "deep web" where data is stored in backend databases, the semantic annotation of the data is achieved through the mappings between web ontologies and schemas of backend databases. However, maintaining mappings on the semantic web has not yet been considered.

Although mapping maintenance is important and necessary for many applications, solutions to the problem are rare. This is due to many challenges involved, including: how to define consistency of mapping and detect inconsistency of a mapping; what is a right mapping language; how to capture changes to CMs and database schemas; how to devise a plan for reconciling the CMs and schemas according to the intent and expectation of the user; and what are the principles for systematic reconciliation. In this paper, we address these challenges and offer a systematic study and comprehensive evaluation of how round-trip engineering can be applied to solve the mapping maintenance problem.

The rest of the paper presents our principled approach. In summary, We propose a declarative mapping formalism and define the consistency through "compatible" instances. Subsequently, we explore a novel approach of using correspondences for capturing changes and develop a novel round-trip engineering approach for mapping maintenance. We demonstrate the effectiveness and efficiency of our algorithm by conducting a comprehensive set of experiments.

The remaining content is organized as follows. Section 2 summarized studies on schema mapping adaptation, schema evolution for object-oriented databases, and other related work. Section 3 presents the formal notation used in later sections. Section 4 introduces our formalism for conceptual-relational mappings. Section 5 characterizes schema and CM evolution. Section 6 describes a solution to the problem of mapping maintenance. Section 7 presents our evaluation results. Finally, Section 8 concludes this paper.

2 Related Work

The directly related work is the study on schema mapping adaptation [29, 30]. The goal of schema mapping adaptation is to automatically update a schema mapping by reusing the semantics of the original mapping when the associated schemas change. Yu & Popa [30] explore the schema mapping composition approach. Schema evolutions are captured by formal and accurate schema mappings, and schema adaptation is achieved by composing the evolution mapping with the original mapping. On the other hand, the schema change approach in [29] proposed by Velegrakis et al. incrementally changes mappings each time a primitive change occurs in the source or target schemas. Both solutions focus on reusing the semantics encoded in existing mappings for merely adapting the mappings without considering the synchronization between schemas. This is due to the nature of their problems where schema mappings are primarily used for *data exchange* [16], i.e., translating a data instance under a source schema to a data instance under a target schema. If a schema mapping connecting two schemas which are semantically inconsistent, then the data exchange process simply does not always produce a target instance. Our approach is different from these solutions in that we aim to maintain the *semantic consistency* of conceptual-relational mappings through model synchronzation.

Other related work is schema evolution [25]. In object-oriented databases (OODB), the problem of schema evolution is to maintain the consistency of an OODB when its schema is modified. The challenges are to update the database efficiently and minimize information loss. A variety of solutions, e.g., [9, 13, 18] have been proposed in the literature. Our problem is different from the schema evolution problem in OODB in that we are concerned with the semantic consistency between a schema and a CM. In AutoMed [11, 17], schema evolution and integration are combined in one unified framework. Source schemas are integrated into a global schema by applying a sequence of primitive transformations to them. The same set of primitive transformations can be used to specify the evolution of a source schema into a new schema. In our approach, we do not ask users to specify a sequence of transformations. The EVE [21] investigates the view synchronization problem, which supports a limited set of changes. The work in [14] describes techniques for maintaining mapping in XML p2p databases which is different from our problem.

Another mapping maintenance problem studied in [24] mainly focuses on detecting inconsistency of simple correspondences between schema elements when schemas evolve. This problem is complementary to the problem we consider here.

3 Formal Preliminaries

A table or relation in a relational database consists of a set of tuples. The schema for a table specifies the name of the table, the name of each column (or attribute or field), and the type of each column. Furthermore, we can specify *integrity constraints*, which are conditions that the tuples in tables must satisfy. Here, we consider the *key* and *foreign key* (abbreviated as *f.k.* henceforth) constraints. A key in a table is a subset of the columns of the table that uniquely identifies a tuple. A f.k. in a table T is a set of columns F that *references* the key of another table T' and imposes a constraint that the projection of T on F is a subset of the projection of T' on the key of T'. A relational schema thus consists of a set of relational schemes (or tables for short). Formally, we use $\mathcal{R}$=(R, Σ_R) to denote a relational schema $\mathcal{R}$ with a set of tables R and a set Σ_R of key and f.k. constraints.

A conceptual model (CM) describes a subject matter in terms of concepts, relationships, and attributes. In this paper, we do not restrict ourselves to any particular language for describing CMs. Instead, we use a generic conceptual modeling language (CML), which has the following specifications. The language allows the representation of *classes/concepts/entities* (unary predicates over individuals), *object properties/ relationships* (binary predicates relating individuals), and *datatype properties/ attributes* (binary predicates relating individuals with values such as integers and strings); attributes are single valued in this paper. Concepts are organized in the familiar ISA hierarchy. Relationships and their inverses (which are always present) are subject to cardinality constraints, which allow 1 as lower bounds (called *total* relationships) and 1 as upper bounds (called *functional* relationships). In addition, a subset of attributes of a concept is specified as the identifier of the concept. As in the Entity-Relationship model, a strong entity has a global identifier, while a weak entity is identified by an identifying relationship plus a local identifier. We use $\mathcal{C}=(C, \Sigma_C)$ to denote a CM $\mathcal{C}$ with a set C of concepts, attributes, and relationships and a set Σ_C of identification and cardinality constraints.

We represent a given CM as a graph called an *CM graph*. We construct the CM graph from a CM by considering concepts and attributes as nodes and relationships as edges. There are also edges between a concept node and the attribute nodes belonging to the concept. A many-to-many relationship p between concepts C_1 and C_2 will be written in text as $\boxed{\mathtt{C_1}}$ `---p---` $\boxed{\mathtt{C_2}}$. For a functional relationship q – ones with upper bound cardinality of 1, from C_1 to C_2, we write $\boxed{\mathtt{C_1}}$ `---q->--` $\boxed{\mathtt{C_2}}$. We will treat an ISA relationship as functional in both directions.

4 Conceptual-Relational Mappings

A conceptual-relational mapping specifies a relationship between a CM and a relational schema. More specifically, a mapping consists of a set of statements each of which relates a query expression $\Phi(X, Y)$ in a language $\mathcal{L}_1$ over the CM with a query expression $\Psi(X, Z)$ in a language $\mathcal{L}_2$ over the relational schema, where the shared variables X give rise to the query results. In this paper, we consider conjunctive formulas over concepts, attributes, and relationships in a CM and conjunctive formulas over relational tables which can be translated into equivalent select, join, and project (SJP) query expressions over a relational schema. Queries are evaluated as the usual way.

In the sequel, we will use the terms "mapping" and "mapping statement" interchangeably when the context is clear. Generally, we represent a conceptual-relational mapping (or mapping statement) between a CM and a relational schema as an expression $\Phi(X, Y) = \Psi(X, Z)$, where $\Phi(X, Y)$ and $\Psi(X, Z)$ are conjunctive formulas. The following example illustrates the mapping formalism using a gene expression database and a conceptual model.

Example 1 A gene expression database contains a biosample table to record information about a biological sample which can be a tissue, cell, or RNA material that originates from a donor of a given species:

biosample($\underline{\text{sample\_ID}}$, species, organ, pathology,..., donor_ID), where the underlined column sample_ID is the key of the table and donor_ID is a foreign key to a table called donor.

Figure 1 shows a mapping between the biosample table and a CM containing two concepts Biosample and Person, and a relationship donation. The CM is described in the UML notation. The dashed arrows indicate the correspondences between columns of the relational table and attributes of concepts in the CM. We represent the conceptual-relational mapping between the relational table and the CM as the following expression:

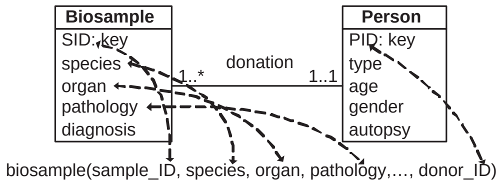


Fig. 1: A Conceptual-Relational Mapping

Biosample$(x_1)\wedge$SID$(x_1,\ sample_ID)\wedge$ species$(x_1,\ species)\wedge$ ...$\wedge$ Person$(x_2)\wedge$ donation$(x_1,\ x_2)\wedge$ PID$(x_2,\ donor_ID)$

= biosample$(sample_ID,\ species,...,\ donor_ID)$,

where the predicates Biosample and Person represent the concepts in the CM, the predicates SID, species,..., represent the attributes of the concepts and the relationship, and the shared variables *sample_ID*, *species*,..., give rise to query results on both sides. ■

Consistent Conceptual-Relational Mappings. We define a consistent conceptual -relational mapping between a CM and a relational schema in terms of *legal instances* of the CM and the relational schema. For a CM $\mathcal{C}=(C,\Sigma_C)$, a legal instance I is an instance of C which satisfies the constraints Σ_C. We use $\mathcal{I}$ to denote the set of all legal instances of $\mathcal{C}$, i.e., $\mathcal{I}=\{I \mid I$ is an instance of C and $I \models \Sigma_C\}$. Likewise, for a relational schema $\mathcal{R}=(R,\Sigma_R)$, we use $\mathcal{J}$ to denote the set of all legal instances of $\mathcal{R}$, i.e., $\mathcal{J}=\{J \mid J$ is an instance of R and $J \models \Sigma_R\}$.

For a query expression $\Phi(X,Y)$ over $\mathcal{C}$, we use I^{Φ} to denote the query results over the instance I. We use J^{Ψ} to denote the query results of the query expression $\Psi(X,Z)$ over the instance J of $\mathcal{R}$. We say that a pair of legal instances $\langle I,J\rangle$ satisfies a mapping statement $M{:}\Phi(X,Y)=\Psi(X,Z)$ between $\mathcal{C}$ and $\mathcal{R}$, if and only if $I^{\Phi}=J^{\Psi}$, denoted as $\langle I,J\rangle \models M$.

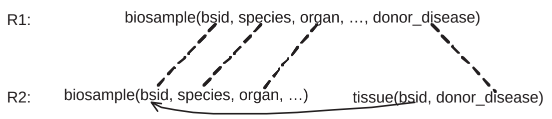
Definition 1 (Consistent Conceptual-Relational Mapping). *For a CM $\mathcal{C}=(C,\Sigma_C)$ and a relational schema $\mathcal{R}=(R,\Sigma_R)$, a mapping $M{:}\Phi(X,Y)=\Psi(X,Z)$ between $\mathcal{C}$ and $\mathcal{R}$ is consistent if and only if for every legal instance $I\in\mathcal{I}$, there is a legal instance $J\in\mathcal{J}$ such that $\langle I,J\rangle\models M$, and for every legal instance $J'\in\mathcal{J}$, there is a legel instance $I'\in\mathcal{I}$ such that $\langle I',J'\rangle\models M$.*

Essentially, the consistency of a mapping dictates the "compatibility" of the constraints in the CM and the schema.

5 Changes to Schemas and CMs

Changes to conceptual models and relational schemas can be classified along two orthogonal axes. First, on the *action* axis, changes can be classified into (1) adding/deleting elements; (2) merging/splitting elements; (3) moving/copying elements; (4) renaming elements; and (5) modifying constraints. Second, on the *effect* axis, for a conceptual-relational mapping, changes can be classified into (i) changes that cause modifications on the conceptual-relational mapping; (ii) changes that cause modifications on the related schema (or CM); and (iii) changes that cause modifications on both the mapping and the related schema (or CM).

A user can change a schema (or CM) in different ways: either through modifying the original schema (or CM) or by generating a new schema (or CM) directly. It is difficult to ask the user to provide a sequence of primitive actions for capturing the changes. It is probably easier to ask the user to draw a set of simple correspondences between the elements in the new schema (or CM) and the elements in the original schema (or CM). In this paper, we use a set of correspondences between columns in schemas (or attributes in CMs) to capture the commonality/differences between the new schema (or CM) and the original schema (or CM).


R1:
biosample(bsid, species, organ, ..., donor_disease)
R2:
biosample(bsid, species, organ, ...)
tissue(bsid, donor_disease)
Fig. 2: Capturing Changes to a Schema

Example 2 Figure 2 shows on the top an original schema $\mathcal{R}_1$ consisting of a single table biosample. On the bottom is a new schema $\mathcal{R}_2$ containing two tables biosample and tissue. $\mathcal{R}_2$ evolved from $\mathcal{R}_1$. The dashed lines between columns in $\mathcal{R}_1$ and the columns in $\mathcal{R}_2$ capture the commonality/differences between the original schema and the new schema. The open arrow indicates that the column tissue.bsid is a foreign key referring the key biosample.bsid. ■

6 Round-Trip Engineering for Conceptual-Relational Mappings

We now develop a round-trip engineering solution for maintaining conceptual-relational mappings under evolution. The primary goal of the maintenance is to keep the mapping consistent by synchronizing the schema and the CM. To fulfill the goal, the algorithm must *understand* the existing semantics in the original mapping and carry out necessary updates based on sound principles. We begin with the exploration on the knowledge encoded in the forward engineering process.

Knowledge about the Conceptual-Relational Mappings in Standard Database Design Process. In relational database design, a standard technique (we refer to this as er2rel schema design) which is widely covered in undergraduate database courses [15] derives a relational schema from an Entity-Relationship diagram. The er2rel design implies a set of conceptual-relational mappings in the form $\Phi(X, Y) = T(X)$, where $\Phi(X, Y)$ is a conjunctive formula encoding a tree structure called *semantic tree (or s-tree)* [8] in a CM, and $T(X)$ is a relational table with columns X. Such a conceptual-relational mapping is also used in the middleware mapping technologies.

We choose to design our solution for mapping maintenance in a systematic manner by considering the behavior of our algorithm on the conceptual-relational mappings implied by the er2rel design. In our previous work [8], we have carefully analyzed the knowledge encoded in the er2rel design. We summarize the knowledge related to our study in this paper as follows.

1. The er2rel design associates a relational table with a tree structure called semantic tree (s-tree) in a CM.
2. An s-tree can be decomposed into several subtrees called *skeleton trees*: a skeleton tree corresponding to the key of the table, skeleton trees corresponding to f.k.s of the table, and skeleton trees corresponding to the rest of the columns of the table.
3. Each skeleton tree has an anchor which is the root of the skeleton tree. An anchor also corresponds to the central object for deriving a table.
4. To satisfy the semantics of the key in a table, the s-tree is connected by functional paths from the anchor of the key skeleton tree to the anchors of f.k. skeleton trees and other skeleton trees.

Example 3 In Figure 1, the mapping associates the biosample table with the s-tree $\boxed{\texttt{Biosample}}$ `---donation-->-` $\boxed{\texttt{Person}}$. The s-tree is decomposed into two skeleton trees: $\boxed{\texttt{Biosample}}$ with anchor Biosample for the key <u>sample_ID</u> of the table and $\boxed{\texttt{Person}}$ with anchor Person for the foreign key donor_ID. (Skeleton trees for weak entities are more complex; see Example 5). The two anchors are connected by a functional edge `---donation->--`. ■

Sketch of the Maintenance Algorithm. We first outline the algorithm for maintaining mappings which are in the form of $\Phi(X,Y)=T(X)$. We develop the complete algorithm later. Given a relational schema $\mathcal{R}$, a CM $\mathcal{C}$, a set of existing consistent conceptual-relational mappings $M=\{\Phi(X,Y)=T(X)\}$ between $\mathcal{R}$ and $\mathcal{C}$, a new schema $\mathcal{R}'$ (or CM $\mathcal{C}'$), and a set of correspondences M' between $\mathcal{R}$ and $\mathcal{R}'$ (or between $\mathcal{C}$ and $\mathcal{C}'$), the algorithm works in several steps for fulfilling the goals of mapping maintenance:

1. Analyze the existing semantics in the original mapping in terms of skeleton trees and connections between anchors of skeleton trees.
2. Discover changes through the correspondences between the new schema/CM and the original schema/CM.
3. Synchronize the associated CM/schema and adapt the mapping accordingly.

Illustrative Examples. Before fleshing out the above steps, we illustrate the algorithms using several examples on *schema evolution*. Through these examples, we lay out our principles for mapping maintenance.

Example 4 [Adding a Column] Figure 3 (a) shows a mapping which is specified as following statement:
Sample(x_1) $\wedge$ sid(x_1, sid) $\wedge$ Person(x_2) $\wedge$ originates(x_1, x_2) $\wedge$ pid(x_2, $donor$)
= sample($sid, donor$).
Figure 3 (b) shows that a column species was added to the table sample(<u>sid</u>, donor). *For adding an element in the schema, our goal of mapping maintenance is to add a corresponding element in the CM to maximize the coverage of the schema elements.* Since the key column sid corresponds to the identifier attribute of the Sample class and the column donor is a foreign key referring to the key of a table donor(did) for the Person class, we synchronize the CM through adding an attribute species to the Sample class which is the anchor of the skeleton tree corresponding to the key <u>sid</u>. ■

The *first principle* for the mapping maintenance for schema evolution is to use the key and foreign key information in the original and new schemas through the correspondences to locate the appropriate elements in the CM for adding new attributes.

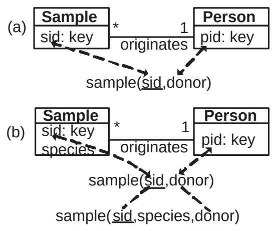


Fig. 3: Adding a Column to Schema

Example 5 Let us consider the case for adding a foreign key column. Figure 4 shows an original mapping enclosed whithin the rectangle:
M:Test(x_1) $\wedge$ tid$(x_1, test)$ $\wedge$ Sample(x_2) $\wedge$ sid$(x_2,\ sid)$ $\wedge$ screenedIn(x_1, x_2) $\wedge$ Person(x_3) $\wedge$ originates$(x_2,\ x_3)$ $\wedge$ pid$(x_3,$ $donor)$ = sample$(sid, test, donor)$.

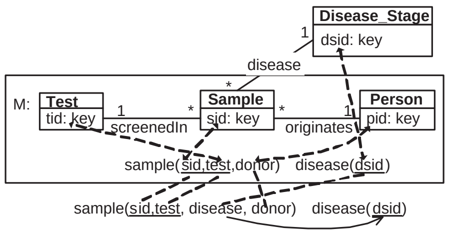


Fig. 4: Adding a Foreign Key Column to Schema

In the CM, Sample is modeled as a weak entity with an identifying functional relationship screenedIn connecting to the owner entity Test. Accordingly, the key of the table sample(<u>sid,test</u>,donor) is the combination of columns sid and test with test being a foreign key referring to a table test(tid) for the Test class (not showing in the figure.) On the bottom of Figure 4, the table sample(sid, test, donor) was changed to sample(sid,test,disease,donor) with the column disease being a foreign key referring to the key of the table disease(dsid) (shown as the open arrow.) To update the mapping between the new sample table and the CM, we analyze the key and foreign key structure of the table and recognize that Sample class is the *anchor* of the skeleton tree `Test` -<--screenIn--- `Sample` for the key. The newly added foreign key disease should indicate that there is a functional relationship from the Sample class to the Disease_Stage class rather than a functional relationship from the Test class to the Disease_Stage class. Therefore, we add/discover a functional relationship disease in the original CM and update the mapping between the sample(sid, test, disease, donor) and the new CM. ■

Our *second principle* is to use key and foreign key structure in the schemas through the correspondences to locate the anchors of the appropriate skeleton trees for discovering/adding relationships.

Example 6 [Changing Constraints] The following existing mapping associates a relational table treat(<u>tid</u>, <u>sgid</u>) with a CM `Treatment` ---appliesTo--- `Sample_Group`:
Treatment(x_1) $\wedge$ tid$(x_1,\ tid)$ $\wedge$ Sample_Group(x_2) $\wedge$ appliesTo$(x_1,\ x_2)$ $\wedge$ sgid$(x_3,$ $sgid)$ = treat$(tid,\ sgid)$, where the relationship appliesTo is many-to-many.

Later, the database administrator obtained a better understanding of the application by realizing that each treatment only applies to one sample group. Consequently, the DBA changed the key of the treat table from the combination of columns tid and sgid to the single column tid. Having the change on the schema, we update the appliesTo from a many-to-many relationship to a functional relationship $\boxed{\texttt{Treatment}}$ `---appliesTo->--` $\boxed{\texttt{Sample\_Group}}$ to keep the mapping consistent. ■

The *third principle* is to align the key and foreign key constraints in the (new) schema with the cardinality constraints in the (new) CM.

Maintenance Algorithm. In this paper, the maintenance algorithm requires that each original conceptual-relational mapping statement $\Phi(X,Y)=T(X)$ is consistent and associates a relational table $T(X)$ with a semantic tree $\Phi(X,Y)$ in a CM. For a general consistent conceptual-relational mapping associating a graph with a conjunctive formulas over a schema, we can first convert the graph into a tree by replicating nodes (see [5]). Then we either decompose the mapping into mappings between semantic trees and single tables or treat the entire conjunctive formula over the schema as a big table. The details for converting general mappings into mappings between semantic trees and tables are beyond the scope of this paper and will be realized in the future work.

The maintenance algorithm has two components. The first component deals with changes to schemas, and the second component deal with changes to CMs. We first focus on schema changes. The following Procedure 1 maintains the consistency of conceptual-relational mappings when schemas evolve.

Procedure 1 Maintain Mappings When Schemas Evolve
Input: A set of consistent conceptual-relational mappings $M=\{\Phi(X,Y)=T(X)\}$ between a CM $\mathcal{C}$ and a relational schema $\mathcal{R}$; a set of correspondences M' between columns in $\mathcal{R}$ and columns in a new schema $\mathcal{R}'$
Ouput: Synchronized CM $\mathcal{C}''$ and a updated set of mappings M'' between $\mathcal{C}''$ and $\mathcal{R}'$.
Steps:

1. Mark skeleton trees: for each mapping statement in M, decompose the semantic tree in the CM into several skeleton trees based on the key and foreign key structures of the table; mark the associations between keys/f.k.s and skeleton trees.
2. Apply the principles we have laid out above to each of the following cases for synchronizing the CM and update the mapping (we ignore the renaming change in our algorithm):
   – **Case 1:** A new table evolved from a single table by adding columns, deleting columns, or changing constraints.
   – **Case 2:** A new table evolved from several tables by adding columns, deleting columns, or changing constraints.
   – **Case 3:** Several tables evolved from a single table by adding columns, deleting columns, or changing constraints.

We now elaborate on each case.

Case 1: If a new table evolved from a single table, then columns which are not foreign key have been changed or a foreign key has been deleted. If a new column is added, then add a new attribute to the anchor of the key skeleton tree (see Example 4). If the column becomes part of the key, then the new attribute becomes part of the identifier of the anchor. If a column is deleted, we only update the mapping by removing the reference to the deleted column in the mapping. If the key constraint has been changed, then synchronize the identifier of the anchor of the key skeleton tree accordingly.

Case 2: If a new table T evolved from several tables $\{T_1, T_2, ..., T_n\}$, then we connect the semantic trees corresponding to the original tables $\{T_1, T_2, ..., T_n\}$ into a larger semantic tree as follows. Suppose the key of the table T come from the key of table T_1. Let the skeleton trees $\{S_1, S_2, ..., S_n\}$ correspond to the keys of $\{T_1, T_2, ..., T_n\}$. Connect the anchor of S_1 to the anchors of $\{S_2, ..., S_n\}$ by functional edges. The new table is mapped to the larger tree. Example 5 illustrates the case where a new table sample(sid, test, disease, donor) evolved from two original tables sample(sid, test,donor) and disease(dsid,diagnosis). The new table is mapped to a larger semantic tree by connecting the two anchors Sample and Disease_Stage using a functional edge `---disease-->-`.

Case 3: Several tables $\{T_1, T_2, ..., T_n\}$ evolved from a single table T. Without losing generality, suppose T_1 inherit the key of T. We create new concepts $\{C_2, ..., C_n\}$ in the CM for the new tables $\{T_2, ..., T_n\}$, respectively. Let C_i be the anchor of the skeleton tree corresponding to the key of T_i. For two tables T_i and T_j, if there is a foreign key constraint from the column T_i.f to the key of T_j, then we connect C_i to C_j by a functional edge in the CM. If the column T_i.f is also the key of the table T_i, then we connect C_i to C_j by an ISA relationship.

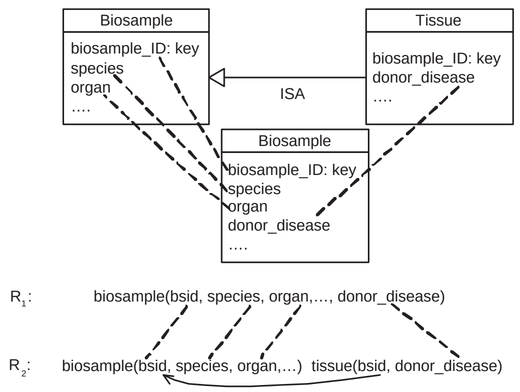


Fig. 5: New Tissue Concept for New tissue Table

Example 7 [Adding New Tables] In Figure 5, a new schema $\mathcal{R}_2$ containing two tables biosample and tissue evolved from the original schema $\mathcal{R}_1$ with a single table biosample. The original mapping associates $\mathcal{R}_1$ with the concept Biosample. On the top of the figure is a new CM, where a new concept Tissue is added and connected to Biosample by an ISA relationship according to the f.k. constraint between the keys of tissue and biosample tables in the new schema $\mathcal{R}_2$. ■

We now turn to the procedure dealing with changes to CMs. Intuitively, synchronizing schemas when associated CMs change is more costly than synchronizing CMs when schemas change because synchronizing schema often results in data translation. Two strategies can be considered for maintaining mappings when CMs change. The first strategy is to design a procedure in the similar fashion as for the Procedure 1. The second is to adapt mappings to maintain consistency without automatic synchronization. We take the second approach in this paper and leave the first approach in the future work. The following Procedure 2 updates conceptual-relational mappings when the CMs evolve.

Procedure 2 Maintain Mappings When CM Evolve
Input: A set of consistent conceptual-relational mappings $M{=}\{\Phi(X,Y){=}T(X)\}$ between a CM $\mathcal{C}$ and a relational schema $\mathcal{R}$; a set of correspondences M' between attributes in $\mathcal{C}$ and attributes in a new CM $\mathcal{C}'$
Ouput: Update M to a new set of mappings M'' between $\mathcal{R}$ and $\mathcal{C}'$.
Steps:

1. Mark skeleton trees: the same as in the first step of Procedure 1.
2. For a mapping statement in M associating a semantic tree S with a table T
   (a) If the skeleton tree corresponding to the key of T has changed such that identifier attributes of the anchor were added/deleted or a cardinality constraint in the skeleton tree has changed from one to many, then drop the mapping. */*changes to the identifier information of either a strong or a weak entity will result in inconsistent mapping to the original table.*/*
   (b) Else if a cardinality constraint imposed on a relationship p in S has changed from many to one or from one to many, then remove from S the relationship edge p and the rest part connecting to the anchor through p. Update the mapping so that T is mapped to the new smaller tree.
   (c) Else compose the correspondences M' with the original mapping M to generate a new mapping M'' between $\mathcal{R}$ and $\mathcal{C}'$. */*see [30] for composition algorithm.*/*

The following states the desired property of the maintenance algorithm consisting of the steps in Procedure 1 and Procedure 2.

Proposition 1. *Let $M{=}\{\Phi(X,Y){=}T(X)\}$ be a set of consistent conceptual-relational mappings between a CM $\mathcal{C}$ and a relational schema $\mathcal{R}$. Let $\mathcal{R}'$ (or $\mathcal{C}'$) be a new schema (or a new CM) that evolved from $\mathcal{R}$ (or $\mathcal{C}$). Let M' be a set of identity mappings between columns in $\mathcal{R}$ and columns in $\mathcal{R}'$ (or attributes in $\mathcal{C}$ and attributes in $\mathcal{C}'$.) Each mapping in the set of conceptual-relational mappings returned by the* Procedure 1 *(or* Procedure 2*) is consistent.*

7 Experience

To evaluate the performance of our round-trip solution for maintaining conceptual-relational mappings, we applied the algorithm to a set of conceptual-relational mappings drawn from a variety of domains. The purpose of our evaluation is two-folds: (1) to test the efficiency of the algorithm and (2) to measure the benefits of mapping maintenance over reconstructing consistent mappings using mapping discovery tools.

Data Sets. We selected our test data from a variety of domains. Our previous work [8] on the development of the MAONTO mapping tools generated conceptual-relational mappings for many of the test data. Subsequently, our other previous work [5] used the conceptual-relational mappings for improving traditional tools on constructing direct mappings between database schemas. It follows naturally to continue on this set of data for measuring the benefits of mapping maintenance. Table 1 summarizes the characteristics of the test data. The size of a mapping is measured by the size of the semantic tree - the number of nodes including attribute nodes.

Methodology. Our experiments focused on maintaing the consistency of tested mappings under *schema evolution*. For each mapping, we applied different types

Schema	**#Tables**	**Avg. # Cols Per Table**	**CM**	**#Nodes in CM**	**Avg. Mapping Size**
DBLP	22	9	Bibliographic	75	9
Mondial	28	6	Factbook	52	7
Amalgam	15	12	Amalgam ER	26	10
3Sdb	9	14	3Sdb ER	9	6
CS Dept.	8	6	KA onto.	105	7
Hotel	6	5	Hotel Onto.	7	7
Network	18	4	Network onto.	28	6

Table 1: Characteristics of Test Data

of changes to the relational table. For each type of change, we ran the maintenance algorithm for measuring (1) execution time and (2) benefits in comparison to the mapping reconstructing approach. The types of changes to a table include: (a) adding/deleting ordinary columns; (b)adding/deleting key columns; (c) splitting a table; (d) merging two tables; (e) add/deleting f.k. columns; (f) moving columns from one table to another table; and (g) changing existing key and f.k. constraints.

To measure the benefits of mapping maintenance, we adopt the approach for measuring how much user effort can be saved when schemas evolved and a new *consistent* mapping has to be established. Both Velegrakis et al. in [29] and Yu & Popa in [30] applied the similar approach for measuring the benefits of mapping adaptation. Essentially, the user effort for obtaining a consistent mapping through mapping maintenance after the schema evolved is compared to the same type of user effort spent for reconstructing the mapping. In our study, we compared the mapping maintenance approach with the MAPONTO [8] tool for discovering mappings.

For a mapping $\Phi(X,Y)=T(X)$ associating a semmantic tree with a relational table, let T' be the new table that evolved from T. For mapping maintenance, the user specifies a set of simple correspondence bewteen T' and T. Then the maintenance algorithm generates a new mapping between T' and, probably, an updated semantic tree. On the other hand, to reconstruct a mapping using the MAPONTO tool, the user also needs to specify a set of correspondences between T' and the CM. However, MAPONTO tool may be unable to generated the expected mappings because the CM is out of synchronization. If the expected mapping is generated by the maintenance algorithm while it is missing from the results of MAPONTO, then we assign 100% to the benefit of maintenance. Otherwise, we use the following quantity to measure the benefit:

$$1 - \frac{\#mapping_{maintenace}}{\#mappping_{MAPONTO}+\#correspondences}$$

Because specifying correspondences between a schema and CM is much more costly than specifying correspondences between an evolved schema and the original schama, we omit the effort for specifying evolution correspondences from the above quantity.

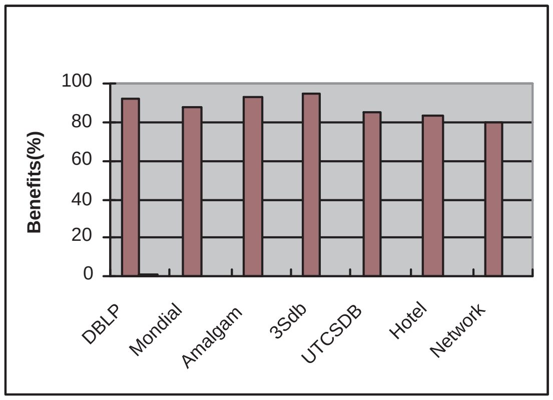


Fig. 6: Benefits of Mapping Maintenance

Results. First of all, the times used by the maintenance algorithm for synchronizing CMs and updating mappings are insignificant. For all the tested mappings, the maintenance algorithm took less than one second to generate expected results. This is comparable with the MAPONTO tool for discovering mappings between schemas and CMs. Next, in terms of benefits, Figure 6 presents the average benefits for the tested cases. The results show that the round-trip engineering solution provides significant benefits in terms of maintaining the consistency of conceptual-relational mappings under evolution.

8 Conclusions

In this paper, we studied the problem of maintaining the consistency of conceptual-relational mappings with evolving schemas and CMs. We motivated the need for synchronizing the CM and relational schema associated by a conceptual-relational mapping. We presented a novel round-trip engineering framework and developed algorithms that automatically maintain conceptual-relational mappings as schemas/CMs evolve. Our solution is unique in that we carefully compile the knowledge encoded in the widely covered methodology for database design into our approach. Experimental analysis showed that the solution is efficient and provides significant benefits for maintaining conceptual-relational mappings in dynamic environments.

References

1. *Database Visual Architect*. http://www.visual-paradigm.com/product/dbva.
2. *Oracle TopLink*. http://www.oracle.com/technology/product/ias/toplink.
3. S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 254–263, 1998.
4. A. Adya, J. Blakeley, S. Melnik, and S. Muralidhar. Anatomy of the ado.net entity framework. In *SIGMOD'07*, 2007.
5. Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos. A Semantic Approach to Discovering Schema Mapping Expression. In *ICDE'07*, 2007.

6. Y. An, A. Borgida, and J. Mylopoulos. Constructing Complex Semantic Mappings between XML Data and Ontologies. In *ISWC'05*, 2005.
7. Y. An, A. Borgida, and J. Mylopoulos. Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In *ODBASE'05*, 2005.
8. Y. An, A. Borgida, and J. Mylopoulos. Discovering the Semantics of Relational Tables through Mappings. *JoDS VII*, pages 1–32, 2006.
9. J. Banerjee et al. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *SIGMOD'87*, 1987.
10. C. Bauer and G. King. *Java Persistence with Hibernate.* Manning Publications, November, 2006.
11. P. M. Brien and A. Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *CAiSE'02*, 2002.
12. D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R.Rosati. Data Integration in Data Warehousing. *J. of Coop. Info. Sys.*, 10(3):237–271, 2001.
13. K. T. Claypool, J. Jin, and E. Rundensteiner. SERF: Schema Evolution through an Extensible, Re-usable, and Flexible Framework. In *CIKM'98*, 1998.
14. D. Colazzo and C. Sartiani. Mapping maintenance in xml p2p databases. In *DBPL'05*, 2005.
15. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems (5th Edition).* Addison Wesley, 2006.
16. R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *Proceedings of International Conference on Database Theory (ICDT)*, 2003.
17. H. Fan and A. Poulovassilis. Schema evolution in data warehousing environments — a schema transformation-based approach. In *ER'04*, 2004.
18. F. Ferrandina, G. Ferran, T. Meyer, J. Madec, and R. Zicari. Schema and Database Evolution in the O2 Object Database System. In *VLDB'95*, 1995.
19. J.-L. Hainaut. *Database reverse engineering.* http://citeseer.ist.psu.edu/article/hainaut98database.html, 1998.
20. H. Knublauch and T. Rose. Round-trip engineering of ontologies for knowledge-based systems. In *SEKE'00*, 2000.
21. A. Lee, A. Nica, and E. Rundensteiner. The eve approach: View synchronization in dynamic distributed environment. *TKDE*, 14(5):931–954, 2002.
22. M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.
23. A. Y. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *J. of Intelligent Info. Sys.*, 5(2):121–143, 1996.
24. R. McCann et al. Maveric: Mapping Maintenance for Data Integration Systems. In *VLDB'05*, 2005.
25. E. Rahm and P. Bernstein. An on-line bibliography on schema evolution. *SIGMOD Record*, 35(4):30–31, 2006.
26. S. Sendall and J. Kuster. Taming model round-trip engineering. In *Proceedings of Workshop on Best Practices for Model-Driven Software Development*, 2004.
27. O. Udrea, L. Getoor, and R. J. Miller. Leveraging data and structure in ontology integration. In *SIGMOD'07*, 2007.
28. F. van Harmelen. *A Semantic Web Primer.* MIT Press, 2004.
29. Y. Velegrakis, R. J. Miller, and L. Popa. Mapping Adapdation under Evolving Schemas. In *VLDB'03*, 2003.
30. C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schema Evolve. In *VLDB'05*, 2005.