Maintaining Mappings between Conceptual Models and Relational Schemas

Yuan An, Drexel University, USA
Xiaohua Hu, Drexel University, USA
Il-Yeol Song, Drexel University, USA

ABSTRACT

This paper describes a round-trip engineering approach for incrementally maintaining mappings between conceptual models and relational schemas. When either schema or conceptual model evolves to accommodate new information needs, the existing mapping must be maintained accordingly to continuously provide valid services. In this paper, the authors examine the mappings specifying “consistent” relationships between models. First, they define the consistency of a conceptual-relational mapping through “semantically compatible” instances. Next, the authors analyze the knowledge encoded in the standard database design process and develop round-trip algorithms for incrementally maintaining the consistency of conceptual-relational mappings under evolution. Finally, they conduct a set of comprehensive experiments. The results show that the proposed solution is efficient and provides significant benefits in comparison to the mapping reconstructing approach.

Keywords: Conceptual Modeling, Data Integration, Database Design, Database Reverse Engineering, Mapping Maintenance, Round-Trip Engineering, Semantic Mapping

INTRODUCTION

There are many data-centric applications relying on some kinds of mappings between conceptual models and relational schemas—conceptual-relational mappings. The mappings are used to achieve interoperability (An, Borgida, Miller, & Mylopoulos, 2007) or to overcome the well-known impedance mismatch problem (Elmasri & Navathe, 2006), that is, the differences between the data model used by databases and the modeling capabilities and programmability needed by the application. Essentially, a conceptual-relational mapping specifies a particularly meaningful relationship between a conceptual model (hereafter, CM) and a relational schema. Most often, a mapping specifies a semantically consistent relationship. Informally, a semantically consistent relationship between a CM and a relational schema specifies that, despite the differences between the constructs and abstraction levels of the modeling languages, both the CM and relational schema will describe the same “semantics” of...
an application. A CM describes an application in terms of entities, relationships, and attributes, while a relational schema describes information in terms of relational tables; each table has one or more columns with a primary key, and zero or more foreign key constraints. A semantically consistent relationship that associates relationships/entities in a CM with relational tables in a relational schema satisfies the following condition: The constraints imposed on the relationships/entities, such as cardinality/participation constraints, encode the same semantic requirements as that described by the key and foreign key constraints in the relational tables. For instance, a many-to-one relationship from an entity \( E_1 \) to an entity \( E_2 \) in an Entity-Relationship diagram will 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 (Elmasri & Navathe, 2006). The key and foreign key constraints reflect the semantics encoded in the many-to-one relationship.

However, conceptual models and relational schemas evolve over time to accommodate changes in the information they represent. Such evolution may cause existing conceptual-relational mappings to become inconsistent. For example, the database administrator (DBA) in charge of the aforementioned relational table might change the key of the table from the identifier of \( E_1 \) to the combination of the identifiers of \( E_1 \) and \( E_2 \). Consequently, the many-to-one relationship from \( E_1 \) to \( E_2 \) in the ER diagram would be semantically inconsistent with the new table. The reason is that some instances of the table might violate the many-to-one relationship. When conceptual models and schemas change, the conceptual-relational mappings between conceptual models and schemas must be updated. 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. Second, the mapping generation process, even with the help of mapping generation tools (An, Borgida, & Mylopoulos, 2005a, 2005b), can be costly in terms of human effort and expertise. Especially, complex CMs and schemas that were developed independently require a great deal of effort for reconciliation. A better solution would be to design algorithms that synchronize CMs and schemas, and reuse the original semantics. The algorithms should be able to incrementally update the mappings into a set of new mappings. The new mappings should be consistent with respect to the new CMs and schemas.

The process of synchronizing models by keeping them consistent is called Round-Trip Engineering (RTE) (Knublauch & Rose, 2000; Sendall & Kuster, 2004). 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 process for maintaining the consistency of conceptual-relational mappings. Notice that round-trip engineering is not forward engineering, for example, generating a relational schema from a CM, plus reverse engineering (Hainaut, 1998), for example, generating a new CM from an existing schema. RTE focuses on synchronization.

**Motivation**

We begin with 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 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. 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. The CMs and logical schemas may evolve independently causing the “legacy data” problem. Saving and maintaining the mappings between CMs and logical schemas implied by the database design process will help reduce “legacy data.”

**Data-Centric Applications.** To increase the productivity of the developers of data-centric applications, there are a number of middleware mapping technologies such as Hibernate (Bauer & King, 2006), DB Visual Architect, Oracle TopLink, and Microsoft ADO.NET (Adya, Blakeley, Melnik, & Muralidhar, 2007). They provide an easy-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 (Lenzerini, 2002; Zhao & Siau, 2007). There are many ontology-based data integration applications which use ontologies as their global views. For these applications, the mappings between ontologies and local data sources are the main vehicles for data integration. Early studies have been focused on integration architectures, query answering capabilities, and global view integration. 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, 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 conceptual-relational mapping maintenance is important and necessary for many applications, solutions to the problem are rare. The scarcity of solutions is due to many challenges 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 a comprehensive evaluation of how round-trip engineering can be applied to solve the mapping maintenance problem.

The rest of the paper presents our approach. In summary, we explore the approach of using correspondences for capturing changes. We develop a novel round-trip engineering approach for maintaining consistent conceptual-relational mappings. We demonstrate the effectiveness and efficiency of our algorithms by conducting a set of experiments.

The remaining content is organized as follows. The “Literature Review” section reviews studies on related work. The “Formal Preliminaries” section presents the formal notation used in later sections. The section entitled “Mappings between Conceptual Models and Relational Schemas” introduces our formalism for conceptual-relational mappings. The section entitled “Changes to Schemas and CMs” characterizes schema and CM evolution. The section entitled “Maintaining Conceptual-Relational Mappings” describes a solution to the problem of mapping maintenance and provides formal results of the proposed algorithms. The “Experience” section presents our evaluation results. The “Discussion” section highlights the contributions and discusses limitations. Finally, the last section concludes the paper.

**LITERATURE REVIEW**

We conduct a comprehensive literature review in this section. The areas we reviewed are directly
related to the work in this paper. These areas include mappings, view maintenance, schema mapping adaptation, schema evolution for object-oriented databases, conceptual models, reverse engineering, round-trip engineering, data integration, and the Semantic Web.

Mappings. Mapping is the problem of finding a “meaningful” relationship from one data model/representation to another data model/representation (Evermann, 2008; Madhavan, Bernstein, Domingos, & Halevy, 2002). The relationship is expressed in terms of logical formulae. A mapping is often used for data integration, exchange, and translation. Mappings are fundamental to many applications. To study them in a general, application independent way, Madhavan et al. (2002) proposed a general framework for mappings between domain models. A domain model denotes a representation of a domain in a formal language, such as a relational schema in the relational formalism. Given two domain models $T_1$ (in a language $L_1$) and $T_2$ (in $L_2$), a mapping between $T_1$ and $T_2$ may include a helper domain model $T_3$ (in language $L_3$), and consists of a set of formulae each of which is over $(T_1, T_2)$, $(T_1, T_3)$, or $(T_2, T_3)$. A mapping formula over a pair of domain models $(T_1, T_2)$ is of the form $e_1 \text{op} e_2$, where $e_1$ and $e_2$ are expressions over $T_1$ and $T_2$, respectively, and $\text{op}$ is a well-defined operator. For example, both $e_1$ and $e_2$ can be query expressions over two relational databases $T_1$ and $T_2$. The query results of $e_1$ and $e_2$ should be compatible, and $\text{op}$ can be a subset relationship between the two queries. The semantics of the mapping is given by the interpretations of all domain models involved such that these interpretations together satisfy all the mapping formulae.

Given two data models/representations—either relational schemas or conceptual models such as Entity-Relationship diagrams, finding the mapping between the two representations is a difficult problem. Nevertheless, people have striven to develop tools for helping users in deriving mappings. Example tools are TranSem (Milo & Zohar, 1998), Clio (Miller, Haas, & Hernandez, 2000; Popa, Velgrakis, Miller, Hernández, & Fagin, 2002), HePToX (Bonifati, Chang, Ho, Lakshmanan, & Pottinger, 2005), MQG (Kedad & Bouzeghoub, 1999), the XML data integration system presented in (Kedad & Xue, 2005), and MAPONTO (An, Borgida, & Mylopoulos, 2006). Such a tool usually adopts a two-step paradigm: First, specify some simple correspondences between schema elements; there are a number of tools supporting this task (Melnik, Garcia-Molina, & Rahm, 2002; Rahm & Bernstein, 2001). Then derive plausible declarative mapping expressions. Users need to select the final mappings from the list of candidates. The selection process may be assisted using the actual data stored in the database (Yan, Miller, Haas, & Fagin, 2001). In addition to tools for schema mapping, a considerable body of work exists for discovering mappings between ontologies (Kalfoglou & Scholemmer, 2003; Shvaiko & Euzenat, 2005).

In this paper, we assume that mappings have been established between CMs and relational schemas. We are concerned with maintaining the properties of the original mappings when changes occur to CMs or schemas. Our assumption is that an incremental maintenance approach would outperform the mapping reconstructing approach that uses a mapping discovery tool. We verify the assumption at the end of this paper in the section about experimental results.

View Maintenance. In a database, a view is a derived relation defined in terms of base (stored) relations. A view, thus, defines a mapping from a set of base tables to a derived table. A view can be materialized by storing the tuples of the view in the database. Index structures can be built on the materialized view. Consequently, database accesses to the materialized view can be much faster than re-computing the view. Incremental view maintenance
(Ceri & J.Widom, 1991; Huang, Yen, & Hsueh, 2007; Mumick, Quass, & Mumick, 1997) is the problem of dealing with the methods for efficiently updating materialized views when the base schema data are updated. This problem is closely related to the mapping maintenance problem. View adaptation (Gupta, Mumick, & Ross, 1995; Mohania & Dong, 1996) 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. However, view adaptation may be required after mapping maintenance; hence, we view this work as complementary to mapping maintenance.

**Schema Mapping Adaptation.** The directly related work is the study on schema mapping adaptation (Velegrakis, Miller, & Popa, 2004; Yu & Popa, 2005). 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 (2005) 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 proposed by Velegrakis et al. (2004) 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. Adapta-
tion without synchronization is due to the nature of the adaptation problems where schema mappings are primarily used for data exchange, that is, translating a data instance under a source schema to a data instance under a target schema. If a schema mapping connects 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 synchronization.

**Schema Evolution.** In object-oriented databases (OODB), the problem of schema evolution (Rahm & Bernstein, 2006) 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., (Banerjee, Kim, Kim, & Korth, 1987; Claypool, Jin, & Rundensteiner, 1998; Ferrandina, Ferran, Meyer, Madec, & Zicari, 1995) 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 (Brien & Poulovassilis, 2002; Fan & Poulovassilis, 2004), 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 (Lee, Nica, & Rundensteiner, 2002) investigates the view synchronization problem, which supports a limited set of changes. The work in (Colazzo & Sartiani, 2005) describes techniques for maintaining mapping in XML peer-to-peer databases which is different from our problem. Another mapping maintenance problem studied in (McCann et al., 2005) mainly focuses on detecting inconsistency of simple correspondences between schema elements when schemas evolve. This problem is complementary to the problem we consider here.

**Conceptual Models.** Conceptual modeling is concerned with the construction of computer-based symbol structures which
model some part of the real world directly and naturally (Mylopoulos, 1998). In Database, conceptual modeling produces semantic data models which are used to directly and naturally model an application before proceeding to a logical and physical database design. For data management, semantic data models offer more semantic terms and abstraction mechanisms for modeling an application than logical data models.

The fundamental concern for all conceptual models is the abstraction mechanisms. In the following, we summarize some common abstraction mechanisms used in conceptual modeling.

- **Classification**, sometimes called instanceOf, classifies instances under one or more generic classes. Instances of a class share common properties.
- **Generalization**, referred to as ISA, organizes all classes in terms of a partial order relation determined by their generality/specificity. Inheritance is a functional inference rule of generalization mechanism.
- **Aggregation**, also called partOf, views objects as aggregates of their components or parts. A strong form of aggregation states that a component can be a part of only one aggregate.

**Round-Trip Engineering.** Round-trip engineering (Henriksson & Larsson, 2003) has been applied to and studied in software engineering and knowledge management applications (Demeyer, Ducasse, Tichelaar, & Tichelaar, 1999; Knublauch & Rose, 2000). There are commercial software engineering tools for implementing round-trip engineering between Unified Modeling Language models and source code. In software engineering, round-trip engineering is a process for converting a piece of program code to a design and generating the original code from the design.

While reverse engineering can be defined as the process of reconstructing the design of a program from the program itself, round-trip engineering can recover the original program from the reverse-engineered design. In particular, assume that there is a reverse engineering procedure that is always able to give the design of a given program. Now assume that there is a procedure that will always generate the program from a given design. If a design is reverse engineered from a program, used to generate a program, and the generated program is identical to the original program, then this is a round-trip engineering process.

In maintaining mappings between CMs and relational schemas, a round-trip engineering process is employed to synchronize CMs/schemas when changes occur to them. The process implies that the synchronization will propagate changes of the model on one side to the model on the other side. The propagated changes can be converted back to the original changes.

**Data Integration.** Data integration is the problem of combining data from disparate sources and provides users with a unified view of these data (Lenzerini, 2002). The typical architecture of a data integration system consists of a global schema, a set of data sources (local schemas), and mappings between the sources and the global schema. The sources provide the real data,
while the global schema is an integrated and reconciled view of the real data. Sometimes, a middleware infrastructure (Bouguettaya, Malik, Rezgui, & Korff, 2006) is needed to support data integration.

As ontologies have gained growing attention over the past decade, many people have used an ontology as the global schema for a data integration system, building so-called ontology-based information integration systems, for example, Carnot (Collet, Huhns, & Shen, 1991), SIMS (Arens, Knoblock, & Shen, 1996), OBSERVER (Mena, Illarramendi, Kashyap, & Sheth, 1996), Information Manifold (Levy, Srivastava, & Kirk, 1996), InfoSleuth (Bayardo et al., 1997), PICSEL (Goasdoue, Lattes, & Rousset, 1999), and DWQ (Calvanese, Giacomo, Lenzerini, Nardi, & Rosati, 2001). In these systems, mappings are specified from data sources to an ontology acting as a global schema. One of the important problems associated with ontology-based information integration systems is to maintain the mappings between ontologies and source databases when ontology or database schemas change. In this paper, we study a solution to the problem.

Semantic Web. Recently, the Semantic Web was proposed for improving information gathering and integration on the Web (Berners-Lee, Hendler, & Lassila, 2001). Data on the Semantic Web promises to be machine-understandable by being attached through semantic annotations. These annotations can be based on formal ontologies with, hopefully, widely accepted vocabulary and definitions. Data on the Semantic Web could be database-resident data or HTML Web pages. The annotations to database-resident data can be viewed as mappings between the data and ontologies. For the purpose of describing and publishing ontologies on the Semantic Web, the W3C consortium has proposed the RDF (Klyne & Carroll, January 2003) data model and the OWL ontology language (McGuinness & v. Harmelen, 2004). In this paper, we use a general term conceptual model to refer to a wide range of data representations including Web ontologies.

FORMAL PRELIMINARIES

In this section, we describe some formal notation about the relational schema and CM studied in this paper.

Relational Schema. A relational database consists of a set of relational tables (or relations) each of which contains 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. An 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). Formally, we use $R=(R, \Sigma_R)$ to denote a relational schema $R$ with a set of tables $R$ and a set $\Sigma_R$ of key and f.k. constraints.

Figure 1 shows two relational tables: biosample and person. The biosample table has 4 columns: biosample.sampleID, biosample.species, biosample.donorID*, and biosample.do-norID*; the person table has 3 columns: person.pID, person.age, and person.address. We use a dot notation such as tableName.columnName to indicate a column “columnName” of the table “tableName.” Each table contains a number of tuples consisting of values drawn from the domains of the columns. An underlined column such
as biosample.sampleID is the key of a table. The column annotated with "*" represents a foreign key (f.k.). For instance, the column biosample.donorID* of the table biosample is a foreign key referencing the key person.pID of the table person. A relational table T with columns \( c_1, c_2, \ldots, c_n \) is written as \( T(c_1, c_2, \ldots, c_n) \), for example, biosample (sampleID, species, organ, donorID*).

**Conceptual Model.** 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 ISA hierarchy, that is, generalization/specialization hierarchy. Relationships and their inverses (which are always present) are subject to cardinality constraints each of which consists of two numbers: a lower bound cardinality and an upper bound cardinality. If the lower bound cardinality for a relationship is 1, then the relationship is called a total relationship. If the upper bound cardinality for a relationship is 1, then the relationship is called a functional relationship. If the upper bound cardinalities of a relationship and its inverse are greater than 1, then the relationship is called a many-to-many relationship. 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 \( C = (C, \Sigma_C) \) to denote a CM \( C \) with a set \( C \) of concepts, attributes, and relationships and a set \( \Sigma_C \) of identification and cardinality constraints.

Figure 2 shows a conceptual model having 4 concepts and several relationships. In the figure, a relationship and its inverse are represented as a single line. For instance, the line between the concepts Biosample and Person represents the relationship donor from Biosample to Person and its inverse donation from Person to Biosample.

The concept Person is a specialized concept of the more general concept Donor. These two concepts are organized in an ISA hierarchy. The lower bound cardinality of the donation relationship is 1. This indicates that the donation relationship is total, that is, each donor donates at least one Biosample instance. The upper bound cardinality of the donor relationship is 1. This indicates that the donor relationship is a functional relationship, that is, a given Biosample instance has a unique donor. Both upper bound cardinalities of the relationship testLab and its inverse testSample are greater than 1. This indicates that the relationship is a many-to-many relationship. The keyword “key” indicates the identifier of a concept.
Figure 3 illustrates a conceptual model with a weak entity, Sample concept. A weak entity in a CM diagram is represented as a double-lined rectangle. The identifying/ownership relationship is represented by a dashed line in the CM, that is, the screenedIn relationship. The ownership relationship indicates that a sample cannot be globally identified by its local identifier sid. To globally identify a sample, the combination of the identifier, tid, of the owner entity, Test, and the local identifier, sid, of the weak entity, Sample, must be used together. An identifying/ownership relationship from a weak entity to its owner entity is always functional, that is, each weak entity has a unique owner entity.

We can represent a given CM as a graph called a **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. For the sake of succinctness, we sometimes use UML-like notations, as in Figures 2 and 3, to represent the CM graph. Note that in such a diagram, instead of drawing separate attribute nodes, we place the attributes inside the rectangle concept nodes; and relationships and their inverses are represented by a single undirected edge. A many-to-many relationship p between concepts $C_1$ and $C_2$ will be written in text as $C_1 \--p---C_2$. For a functional relationship q, ones with upper bound cardinality of 1, from $C_1$ to $C_2$, we write $C_1 \--q->--C_2$. In a CM graph, we will treat an ISA relationship as a 1:1 functional edge.

### MAPPINGS BETWEEN CONCEPTUAL MODELS AND RELATIONAL SCHEMAS

In this section, we introduce conceptual-relational mappings and define a consistent conceptual-relational mapping. A conceptual-relational mapping specifies a particularly meaningful relationship between a CM and a relational schema. Formally, a mapping consists of a set of statements each of which relates a query expression $\Phi(X, Y)$ in a language $L_1$ over...
the CM with a query expression \( \Psi(X,Y) \) in a language \( L_1 \) over the relational schema. In such a statement, the shared variables \( X \) give rise to the query results. In this paper, we consider conjunctive formulae over concepts, attributes, and relationships in a CM and conjunctive formulae over relational tables. A conjunctive formula over relational tables can be translated into equivalent select, join, and project (SJP) query expressions. Queries are evaluated in 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 \( \Phi(X,Y) \leftrightarrow \Psi(X,Y) \), where \( \Phi(X,Y) \) and \( \Psi(X,Y) \) are conjunctive formulae. 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. The biosample table is specified as follows:

```plaintext
biosample(sampleID, species, organ, pathology,..., donorID*),
```

where the underlined column biosample. sampleID is the key of the table, and biosample. donorID* is a foreign key to a table called donor. The table represents a BioSample concept and a donation relationship between the BioSample concept and a Person concept in the application domain of a gene expression database. Figure 4 shows a CM, the relational table biosample, and value correspondences between columns of the relational table and attributes of concepts in the CM. The CM contains two concepts Biosample and Person, and a relationship donation. The dashed arrows indicate the value correspondences. A value correspondence relates a column in a relational table to an attribute of a concept in a CM. A set of value correspondences gives rise to the mapping between the CM and the relationship table.

Formally, a mapping between a CM and a relational table is represented as an expression in the form of \( \Phi(X,Y) \leftrightarrow \Psi(X,Y) \). In this example, the conceptual-relational mapping between the relational table biosample and the CM shown in Figure 4 is represented in the following expression:

**Figure 4. A conceptual model, a relational schema, and value correspondences**
where, the prefix $M:\text{Biosample}(x_1) \land \text{SID}(x_1, \text{sampleID}) \land \text{species}(x_1, \text{species}) \land \ldots \land \text{Person}(x_2) \land \text{donation}(x_1, x_2) \land \text{PID}(x_2, \text{donorID}) \leftrightarrow \text{biosample}(\text{sampleID}, \text{species}, \ldots, \text{donorID})$,

Consistent Conceptual-Relational Mappings. The semantics of a mapping expression $\Phi(X, Y) \leftrightarrow \Psi(X, Y)$ is given through data instances. Three possibilities have been considered in the literature (Lenzerini, 2002), sound, complete, and exact. A sound or complete mapping $\Phi(X, Y) \leftrightarrow \Psi(X, Y)$ means that the answers provided by one query, for example, $\Phi(X, Y)$, is contained in the answers provided by another query, for example, $\Psi(X, Y)$. An exact mapping means that these two sets of answers are equivalent, that is, both sound and complete. We define a consistent mapping as an exact mapping over all legal instances of the two related models. We write such a consistent mapping as $\Phi(X, Y) = \Psi(X, Y)$.

Specifically, we define a consistent conceptual-relational mapping $\Phi(X, Y) = \Psi(X, Y)$ between a CM and a relational schema in terms of legal instances of the CM and the relational schema. An instance of a model contains values drawn from the element domains specified by the model definition. An element domain contains a set of permissible values. Generally, for a given model we can construct multiple instances. However, in defining a consistent conceptual-relational mapping, we are more interested in legal instances of models. For-
(or CM) by modifying the original schema (or CM), a possible way to capture the changes is to classify them into a set of primitive actions. An example set of primitive actions (Velegarakis et al., 2004) includes (1) adding/deleting elements, (2) merging/splitting elements, (3) moving/copying elements, (4) renaming elements, and (5) modifying constraints. On the other hand, if a user changed a schema (or CM) through generating a new schema (or CM), it would be difficult to ask the user to provide a sequence of primitive actions for capturing the changes. In this case, 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 a new schema (or CM) and an original schema (or CM).

Simple Correspondences. A simple correspondence is a link/relationship between a single element e₁ in a model M₁ and a single element e₂ in another model M₂. We represent such a correspondence as M₁: e₁ --- M₂: e₂ or a simple line in a graph, where the prefixes M₁ and M₂ distinguishes elements in the two models. For example, R₁: T₁: f --- R₂: T₂: e represents the simple correspondence between the column f of the table T₁ in the relational schema R₁ and the column e of the table T₂ in the relational schema R₂. The following example illustrates that changes to a schema are captured by a set of simple correspondences specified as dashed lines.

Example 2. Figure 5 shows on the top an original schema R₁ consisting of a single table biosample. At the bottom of the figure is a new schema R₂ containing two tables biosample and tissue. R₂ evolved from R₁. The dashed lines between columns in R₁ and the columns in R₂ 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 to the key biosample.bsid.

MAINTAINING CONCEPTUAL-RELATIONAL MAPPINGS

As introduced earlier, conceptual models and schemas evolve over time to accommodate changes in the information they represent. Such evolution may cause existing conceptual-relational mappings to become inconsistent. The focus of this paper is to deal with the problem of maintaining conceptual-relational mappings under evolution. The solution we develop is a round-trip engineering approach that synchronizes the schema and CM, and keeps the mapping consistent. To achieve the goal of synchronization, the algorithm in the round-trip engineering approach must first “understand” the existing semantics in the original mapping and then systematically update the associated schema and CM.

We begin with exploring the knowledge encoded in the standard forward engineering process. The forward engineering process designs relational schemas from CMs. We then develop the algorithms in the round-trip engineering approach by utilizing that knowledge.

Figure 5. Capturing schema change
Knowledge about the Conceptual-Relational Mappings in Standard Database Design Process

In relational database design, a standard technique (we refer to this as ER-to-Relational schema design) which is widely covered in undergraduate database courses (Elmasri & Navathe, 2006) derives a relational schema from an Entity-Relationship diagram. The ER-to-Relational 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 (An et al., 2006) 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.

In this paper, we choose to design our solution for synchronizing models and maintaining mappings in a systematic manner by considering the behavior of our algorithm based on the conceptual-relational mappings encoded in the ER-to-Relational design process. In our previous work (An et al., 2006), we have carefully analyzed the knowledge encoded in the ER-to-Relational design. We summarize the knowledge related to our study in this paper as follows.

The ER-To-Relational design methodology is defined by a function \( t(O) \) returning a relational table scheme for every CM component \( O \), where \( O \) is either a concept/entity or a relationship. We use \( \text{key}(T) \) to refer to the key of a relational table \( T \); therefore, \( \text{key}(t(O)) \) refers to the key of the table generated from the object \( O \) in a CM. In listing the results of applying the function \( t(O) \) to an Entity-Relationship diagram, we use several functions for referring to the components of the ER diagram. Specifically, for an entity set \( E \) (called just “entity” here),

<table>
<thead>
<tr>
<th>Table 1. ER-to-relational design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ER Diagram Object O</strong></td>
</tr>
<tr>
<td><strong>Strong Entity S</strong></td>
</tr>
<tr>
<td>Let ( X ) = attrs(S)</td>
</tr>
<tr>
<td>Let ( K = \text{unique}(S) )</td>
</tr>
<tr>
<td>Let ( y ) be the variable used in the mapping for the entity ( S )</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Weak Entity W</strong></td>
</tr>
<tr>
<td>Let ( E = \text{idOwn}(W) ), ( P = \text{idRel}(W) )</td>
</tr>
<tr>
<td>( Z = \text{attrs}(W) ), ( X = \text{key}(t(E)) )</td>
</tr>
<tr>
<td>( U = \text{localUnique}(W) ), ( V = Z – U )</td>
</tr>
<tr>
<td>Let ( y_j ) and ( y_2 ) be the variables used in the mapping for ( W ) and ( E ), respectively.</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
| | \( ^\text{hasAttribs}(y_j, Z)^\text{E}(y_j) \) ^\text{P}(y_j, y) \)
| | \( ^\text{identify}(y_j, X) \) |
| **Functional Relationship F from E1 to E2** | | |
| Let \( X_1 = t(E_1) \) for \( i = 1, 2 \) | columns: \( X_1 \) |
| Let \( y_1 \) and \( y_2 \) be the variables used in the mapping for \( E_1 \) and \( E_2 \), respectively. | primary key: \( X_1 \) |
| | foreign key: \( X_2 \) |
| | anchor: \( ^\text{E}(y_1) \) referring \( t(E_j) \) |
| | mapping: \( T(X_1, X_2) = E_1(y_1) \) ^\text{identify}(E_1(y_1), X_2) \)
| | \( ^\text{E}(y_1) \) ^\text{P}(y_1, y_2) \) ^\text{identify}(E_1(y_1), y_2) \)
| | \( ^\text{E}(y_1) \) ^\text{identify}(E_1(y_1), X_2) \)
| **Many-to-Many Relationship M between E1 and E2** | | |
| Let \( X_i = t(E_i) \) for \( i = 1, 2 \) | columns: \( X_1 \) |
| Let \( y_1 \) and \( y_2 \) be the variables used in the mapping for \( E_1 \) and \( E_2 \), respectively. | primary key: \( X_1 \) |
| | foreign key: \( X_2 \) |
| | anchor: \( X_1 \) referencing \( t(E_i) \) for \( i = 1, 2 \) |
| | mapping: \( T(X_1, X_2) = E_1(y_1) \) ^\text{identify}(E_1(y_1), X_2) \)
| | \( ^\text{E}(y_1) \) ^\text{P}(y_1, y_2) \) ^\text{identify}(E_1(y_1), y_2) \)
| | \( ^\text{E}(y_1) \) ^\text{identify}(E_1(y_1), X_2) \)

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
function attribs(E) returns the attributes of the entity. A strong entity S has some attributes that act as identifier. We shall refer to these using unique(S) when describing the rules of schema design. A weak entity W has instead localUnique(W) attributes, plus a functional total binary relationship p denoted as idRel(W) to an identifying owner entity denoted as idOwn(W). Table 1 illustrates the application of the function t(O) to the Entity-Relational model that only contains binary relationships.

In addition to the schema (columns, key, f.k.s), Table 1 also associates with a relational table T(X) a number of additional notions:

- An anchor, which is the central object in the CM from which T is derived, and which is useful in explaining our algorithm (it will be the root of the semantic tree);
- A formula for the semantic mapping for the table, expressed as a formula in the form of \( T(X) = \Phi(X,Y) \) (this is what our algorithm should be maintaining); in the body of the formula, the function hasAttribs(x, Y) returns conjuncts attr(x, Y[j]) for the individual columns Y[j].
- The predicate \( \text{identify}_C(x, Y) \) which can be expanded into a conjunctive formula, showing how object x in (strong or weak) entity C can be identified by values in Y.

Another important step of the ER-to-Relational schema design methodology suggests that the schema generated using t(O) can be modified by (repeatedly) merging into the table \( T_0 \) of an entity E the table \( T_i \) of some functional relationship involving the same entity E (which has a foreign key reference to \( T_0 \)). Consequently, tables for multiple functional relationships can be merged into the table for the single entity involved in these functional relationships. Graphically, the merged table represents the information from a tree structure connecting all participating entities. In fact, as illustrated in Table 1, all of the objects that the ER-to-Relational design function t applies to are trees: either a single node tree, for example, a strong entity, or a tree connecting multiple entities, for example, a many-to-many relationship. Trees also correspond to the tables derived from ER models containing n-ary relationships (An et al., 2006). We call the trees that are associated with the relational tables generated by the ER-to-Relational design semantic tree (s-tree) in a CM.

If a relational table T contains foreign keys, then the table also represents information about some relationships. To further capture the fine-grained knowledge about the ER-to-Relational design process, we can decompose an s-tree into several sub-trees: a sub-tree corresponding to the key of the table, sub-trees corresponding to f.k.s of the table, and sub-trees corresponding to the rest of the columns of the table. This decomposition is not a simple decomposition of the entire s-tree into its nodes and edges because some sub-trees are not trivial (i.e., not single nodes). For example, if a table represents a many-to-many relationship between two weak entities, then the key of the table is the combination of the keys of the tables for the two weak entities. The keys of the tables for the two weak entities correspond to trees representing the identifying/owner relationships between the weak entities and their owner entities. We call the sub-trees corresponding to key and foreign keys skeleton trees of an s-tree. The semantic and skeleton trees have the following more characteristics:

1. Each skeleton tree has a root which is the anchor entity illustrated in Table 1.
2. To satisfy the semantics of the key in a table, the s-tree is connected by functional paths from the anchor of the skeleton tree corresponding to the key to the anchors of the skeleton trees corresponding to the foreign keys as well as anchors of other skeleton trees.

Example 3. Figure 6 shows a relational table sample(sid, test*, donor*) and the associated s-tree: Test <-screenIn--Sample --originates-> Person. As the
diagram indicates, the Sample concept is a weak entity connecting to the owner entity Test by the identifying relationships =<--screenIn-->. The anchor of the s-tree is the Sample concept. The sample.donor* column is a foreign key referencing the key donor.did of the table donor(did) which is associated with the s-tree consisting of a single Person node.

The s-tree associated with the table sample(sid, test*, donor*) is decomposed into two skeleton trees: The skeleton tree Test =<--screenIn-- Sample corresponds to the key {sample.sid, sample.test} of the table. The second skeleton tree Person corresponds to the foreign key sample.donor* of table. The anchor of the first skeleton tree is the Sample node and the anchor of the second skeleton tree is the Person node.+

**Sketch of the Maintenance Algorithm**

We now turn to the problem of maintaining a conceptual-relational mapping under evolution. We first outline the maintaining algorithm for round-trip engineering through the mappings in the form of $\Phi(X, Y) = T(X)$. We develop the complete algorithm later. Given a relational schema R, a CM C, a set of existing consistent conceptual-relational mappings $M = \{\Phi(X, Y) = T(X)\}$ between R and C, a new schema R’ (or CM C’), and a set of correspondences $M’$ between R and R’ (or between C and C’), the algorithm works in several steps for fulfilling the goals of model synchronization and mapping maintenance:

1. Analyze the existing semantics in the original mapping in terms of skeleton trees and connections between roots (i.e., 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 and model synchronization.

**Example 4 [Adding a Column].** Figure 7 (a) shows a mapping which is specified in the following statement:

$$M : Sample(x_1) \land sid(x_1, sid)$$
$$\land Person(x_2) \land originates(x_1, x_2)$$
$$\land pid(x_2, donor) = sample(sid, donor)$$

Figure 7 (b) shows that a column species was added to the table sample(sid, 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. Because the key column sid corresponds to the identifier attribute of the

---

*Figure 6. S-tree and skeleton trees*
Sample concept and the column sample.donor* is a foreign key referring to the key donor.did of a table donor(did) (not shown in the figure) for the Person concept, we synchronize the CM through adding an attribute sample.species to the Sample concept which is the anchor of the skeleton tree corresponding to the key sample.sid.

The first principle for mapping maintenance under schema evolution is to locate, through correspondences, the appropriate elements in the CM for adding new attributes. The location process is guided by analyzing the key and foreign key information in the original and new schemas.

Adding a new column to the schema associated with a mapping may have different effects on the mapping. As we illustrated in Example 4, if the new column is not a foreign key, a new attribute is added to the anchor of the skeleton tree in the CM, where the skeleton tree corresponds to the key of the schema. However, if a newly added column is a foreign key, a functional relationship needs to be discovered or added for achieving the goal of mapping maintenance. Because an f.k. represents a functional relationship between two elements in the CM, adding foreign key columns in the schema means that the semantics of the schema covers a new functional relationship. The original mapping needs to accommodate the new semantics. The following example illustrates the situation when a foreign key column is added to the schema associated with a mapping.

Example 5. Figure 8 shows an original mapping between a table sample(sid,test,donor*) and an s-tree:

$$M : \text{Test}(x_1) \land \text{tid}(x_1, test) \land \text{Sample}(x_2) \land \text{sid}(x_2, sid) \land \text{screenedIn}(x_1, x_2) \land \text{Person}(x_3) \land \text{origina}$$

$$\text{tes}\text{sin}(x_2, x_3) \land \text{pid}(x_3, donor) = \text{saple}(sid, test, donor).$$

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(sid,test*,donor*) is the combination of columns sample.sid and sample.test* with sample.test* being a foreign key referring to a table test(tid) for the Test concept (not shown in the figure.)

In Figure 9, the table sample(sid, test*, donor*) is changed to sample(sid,test*,disease*,donor*) with the new column sample.disease* being a foreign key referring to the key disease.dsid of the table disease(dsid). The foreign key constraint is 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 S: Test --> screenIn --> Sample for the key of the original table sample(sid,test*,donor*). Sample is a weak entity which must be identified by the owner entity Test through the identifying relationship screenIn. But Sample is the anchor of the skeleton tree corresponding to the key of the table sample(sid,test*,donor*).

The newly added foreign key sample.disease* represents a functional relationship from the anchor of the s-tree S to the anchor of the s-tree corresponding to the key of the table disease(dsid). The anchor of the s-tree S is the Sample class, while the anchor of the s-tree corresponding to the key of the table disease(dsid) is the Disease_Stage class. Therefore, we update the original mapping between sample(sid,test*,donor*) and the CM to a new mapping between sample(sid,test*,disease*,donor*) and the CM where the s-tree which is associate with sample(sid,test*,disease*,donor*) covers a new functional relationship from Sample to Disease_Stage. The new mapping expression is as follows:

\[
M : \text{Test}(x_1) \land \text{tid}(x_1, \text{test}) \land \\
\text{Sample}(x_2) \land \text{sid}(x_2, \text{sid}) \land \\
\text{screenedIn}(x_1, x_2) \land \text{Person}(x_1) \land \\
\text{originates}(x_2, x_3) \land \text{pid}(x_3, \text{donor}) \land \\
\text{Disease_Stage}(x_4) \land \text{dsid}(x_4, \text{disease}) \land \\
\text{disease}(x_4, x_5) = \text{sample}(\text{sid}, \text{test}, \text{disease}, \text{donor}).
\]

Note that with the key and foreign key structures, we correctly identified a functional relationship from the weak entity Sample to the Disease_Stage concept rather than some
other relationship from other concepts, for example, the owner entity Test concept, to the Disease_Stage concept.

Our second principle is to locate, through the correspondences, the anchors of the appropriate skeleton trees for discovering/adding relationships. The location process is guided by using key and foreign key structure in the schemas.

Primitive changes to schemas include adding columns, deleting columns, and changing constraints. We have illustrated the situations about adding different types of columns. For deleting columns, it is easier to update the mapping by deleting any references to the deleted columns in the mapping expression. The next example illustrates the case when a constraint is changed in the schema associated with a mapping.

Example 6 [Changing Constraints]. The following existing mapping associates a relational table treat(tid*, sgid*) with a CM Treatment--appliesTo--Sample_Group:

\[
M : \text{Treatment}(x_1) \land \text{tid}(x_1, \text{tid}) \land \text{Sample}_{-}\text{Group}(x_2) \\
\text{appliesTo}(x_1, x_2) \land \text{sgid}(x_2, \text{sid}) = \text{treat}(\text{tid}, \text{sgid})
\]

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(tid*, sgid*) table from the combination of columns \{treat.tid*, treat.sgid\} to the single column treat.tid*, so the table becomes treat(tid*, sgid*). Having the change on the schema, we update the appliesTo from a many-to-many relationship to a functional relationship Treatment--\(\Rightarrow\)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.

## Round-Trip Engineering Algorithm for Mapping Maintenance

In this paper, we develop a round-trip engineering algorithm for maintaining consistent mappings. In particular, the input to the maintenance algorithm is a set of conceptual-relational mapping statements \{ \(\Phi(X, Y) = T(X)\) \}. Each statement is consistent and associates a relational table \(T(X)\) with a semantic tree in a CM. The semantic tree is encoded in the formula \(\Phi(X, Y)\). In general, conceptual-relational mappings may associate graphs with conjunctive formulae over schema. For a mapping associating a graph with a conjunctive formulae over a schema, we can first convert the graph into a tree by replicating nodes (An et al., 2007). Then we either decompose the mapping into mappings between semantic trees (s-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 future work.

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

### Procedure 1 Maintaining Mappings When Schemas Evolve

**Input:** A set of consistent conceptual-relational mappings \(M = \{ \Phi(X, Y) = T(X) \}\) between a CM C and a relational schema R; a set of correspondences \(M'\) between columns in R and columns in a new schema R'.

**Output:** Synchronized CM C" and a set of updated mappings \(M''\) between C" and 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 updating the mapping (we ignore the renaming change in our algorithm):**

   a. **Case 1**: A new table is obtained by adding columns, deleting columns, or changing constraints to a single original table.

   b. **Case 2**: A new table is obtained by adding columns, deleting columns, or changing constraints from several original tables.

   c. **Case 3**: Multiple new tables are obtained by adding columns, deleting columns, or changing constraints from a single original table.

We now elaborate on each case.

**Case 1**: If a new table is obtained from a single table, then columns which are not foreign key components have been changed or a foreign key has been deleted. Otherwise, the change would involve multiple original tables, which is covered in Case 2. If a new column is added, then the algorithm adds 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’s skeleton tree accordingly. Finally, if a foreign key is deleted, then update the mapping by removing the corresponding functional relationship referenced in the mapping.

**Case 2**: If a new table $T$ is obtained from several tables $\{T_1, T_2, ..., T_n\}$, then we focus on the following two situations: (1) foreign keys are modified in the table $T$ and (2) multiple tables are merged. When new foreign keys are added to the new table $T$, we consider merging the semantic trees which correspond to the original tables $\{T_1, T_2, ..., T_n\}$ to obtain a larger tree. Suppose that the key of the table $T$ comes from the key of table $T_i$. 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 $S_i$ 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). The new table is mapped to a larger semantic tree by connecting the two anchors Sample and Disease_Stage using a functional edge ->. We now continue the above illustration. If multiple tables are merged into the new table $T$ through the same key, we merge the corresponding s-trees by connecting them using ISA relationships from the anchor of $S_i$ to the anchors of $\{S_2, ..., S_n\}$. If the s-trees are the same because the original tables store the split information about the same s-tree, we update the mapping by merging the original mapping expressions into a larger one.

If existing foreign keys are updated in the table $T$, we proceed as follows. If a foreign key is deleted, then it is handled by Case 1. If a foreign key becomes part of the key, we identify the corresponding functional relationship in the CM. Whether we modify the functional relationship depends on the semantics of the key structure. If the original key is also a foreign key, update the relationship from functional to non-functional. Otherwise, the functional relationship remains unchanged because it indicates an identifying relationship for a weak entity.
**Case 3:** Multiple tables \{\(T_1, T_2, \ldots, T_n\}\} are obtained from a single table \(T\). There are two sub-cases: (1) creating new tables connecting to the original table through foreign keys and (2) splitting an original table. For case (1), if new tables are created and connect to the original table through foreign keys, we create new concepts and functional relationships in the CM. For case (2), splitting an original table into multiple tables primarily means two things: (i) storing the information about a structure of a CM in different relational tables and (ii) storing the information about an ISA (i.e., generalization/specialization) hierarchy into separate tables, each table corresponds to a single concept in the hierarchy. In both case (i) and case (ii), the set of new tables should have the same key. Among the set of tables, the key of one table is not a foreign key, while the keys of the rest of the tables are foreign keys referencing the key of the first table. We focus on the case of splitting the information about an ISA hierarchy here.

The ability to represent ISA hierarchies, such as the one in Figure 2, is a hallmark of modern conceptual modeling languages. If two classes are involved in an ISA relationship, we usually call the more general class `superclass` and the specialized class `subclass`. Almost all textbooks (e.g., Elmasri & Navathe, 2006) describe several techniques for designing relational schemas in the presence of ISA hierarchies:

1. Map each class/concept into a separate table following the standard ER-to-Relational rules. Each table for a subclass must have the same key as the table for the single superclass. The key of a subclass table is also a foreign key referencing the key of the superclass table.
2. Expand inheritance, so that all attributes and relations involving a class \(C\) appear on all its subclasses. Then generate tables as usual for the subclasses, though not for \(C\) itself. This approach is used only when the subclasses cover the superclass.
3. Some researchers also suggest a third possibility: “Collapse up” the information about subclasses into the table for the superclass. The “collapse-up” can be viewed as the result of merging the subclass tables generated at case 1.

When an original table is split into several tables, each of which has the same key as the original table and the key is also a foreign key, then we can consider it as the result of the following process: Splitting the information about an ISA hierarchy into individual tables. The original table might be generated by design (3). In the CM, it could be imagined that the entire ISA hierarchy was “collapsed up.” Then the “collapsed up” table gets split into tables as generated by design (1). Accordingly, we should create separate subclasses in the CM corresponding to the separate tables.

For multiple tables \{\(T_1, T_2, \ldots, T_n\}\] that are obtained from a single table \(T\), we assume that one of the tables inherits the key of \(T\). Without losing generality, suppose \(T_i\) inherits the key of \(T\). We then create new concepts \{\(C_1, \ldots, C_n\}\} in the CM for the new tables \{\(T_1, \ldots, T_n\}\].

Let \(S_i\) be the original s-tree associated with \(T_i\).

Some attributes of \(S_i\) are moved to concepts \{\(C_1, \ldots, C_n\)\}, if the attributes are associated with those columns in \(T\) such that the columns are split into tables \{\(T_1, \ldots, T_n\)\]. Let \(C_i\) be the anchor of the skeleton tree in \(S_i\) corresponding to the key of \(T_i\). For tables \(T_i\) (\(i=2\ldots n\)], if there is a foreign key constraint from the column \(T_p\) to the key of \(T_i\), then we connect \(C_i\) to \(C_p\) by a functional edge in the CM. If the column \(T_p\) is also the key of the table \(T_p\), then we connect \(C_i\) to \(C_p\) by an ISA relationship.

**Example 7 [Adding New Tables].** In Figure 10, a new schema \(R_j\) containing two tables biosample and tissue evolved from the original schema \(R_1\) with a single table biosample.

The original mapping associates \(R_1\) 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 tissue.bsid* and biosample.bsid of the tissue and biosample tables in the new schema $R_2$.

The following proposition states the desired property of the Procedure 1 for maintaining a conceptual-relational mapping through synchronizing the CM and schema when the associated schema evolves.

**Proposition 1.** Let the input of the Procedure 1 be (1) $M=\{\Phi(X,Y)=T(X)\}$ which is a set of consistent conceptual-relational mappings between a CM $C=(C, S_C)$ and a relational schema $R=(R, S_R)$ is consistent, if and only if for every legal instance $I$ of $C$, there is a legal instance $J$ of $R$ such that $<I, J> \models M$, and vice versa. To prove Proposition 1, we consider the set of mappings $M'=\{\Phi'(X,Y)=T'(X)\}$ between a new CM $C'$ and a new schema $R'$ returned by the Procedure 1. We need to prove that for any mapping statement $\Phi'(X,Y)=T'(X)$ in $M'$, for a legal instance of $C'$, we can find a legal instance of $R'$ such that the two instances together satisfy the statement, and vice versa.

We conduct the proof for each case in Procedure 1 as follows.

**Case 1:** A new table is obtained from a single table. Let $T$ be the original table with mapping $\Phi(X,Y)=T(X)$. Let $T'$ be the new table with mapping $\Phi'(X,Y)=\Phi T'(X)$. If $T'$ is obtained from $T$ by adding a new column $c$ which is not a foreign key, then Procedure 1 adds a new attribute $f$ to the
anchor of the s-tree corresponding to \( \Phi(X, Y) \). In particular, \( \Phi'(X, Y) = \Phi(X, Y)'f(\text{anchor}, c) \) and \( T'(X) = T'(\text{columns}(T(X), c)) \), where the variable \( \text{anchor} \) denotes the anchor of the s-tree corresponding to \( \Phi(X, Y) \), and \( \text{columns}(T(X)) \) denotes the columns of the table \( T(X) \). For a legal instance \( I' \) of \( T' \), \( I' \) contains new values for the new column \( c \) and values corresponding to a legal instance \( I \) of \( T \). We can create an instance \( J' \) of \( \Phi'(X, Y) \) through the mapping \( \Phi(X, Y)'f(\text{anchor}, c) = T'(\text{columns}(T(X), c)) \) as follows. First, we map the legal instance \( I \) of \( T(X) \) to the legal instance \( J \) of \( \Phi(X, Y) \). Second, the values for the column \( c \) are mapped to the values of the attribute \( f \). Because the constraints \( \Sigma_c \) remain unchanged, \( J' \) is also a legal instance of \( \Phi'(X, Y) \). Conversely, we can can create an instance \( I' \) of \( T'(X) \) from a legal instance of \( \Phi'(X, Y) \) through the mapping \( \Phi(X, Y)'f(\text{anchor}, c) = T'(\text{columns}(T(X), c)) \). Because the constraints \( \Sigma_f \) remain unchanged, the instance \( I' \) is a legal instance.

If the new column \( c \) is part of the key of the new table, the new attribute \( f \) becomes part of the identifier of the anchor concept of the s-tree corresponding to \( \Phi(X, Y) \). Let \( \text{key}(T(X)) \rightarrow \text{columns}(T(X)) \) be the key constraint in \( K_f \). Let \( \text{identifier}(\text{anchor}) \rightarrow \text{attributes}(\text{s-tree S corresponding to } \Phi(X, Y)) \) be the identifier constraint in \( K_c \). The new constraints become \([\text{key}(T(X)), c] \rightarrow \text{columns}(T'(X)) \) and \([\text{identifier}(\text{anchor}), f] \rightarrow \text{attributes}(\text{s-tree S' corresponding to } \Phi'(X, Y)) \). By the consistency of the mapping \( \Phi(X, Y)'f(\text{anchor}, c) = T'(\text{columns}(T(X), c)) \), \( \text{columns}(T'(X)) \) corresponds to attributes(\text{s-tree S'}) and the values of \( f \) correspond to the value of \( c \). Therefore, the two constraints are \([\text{key}(T(X)), c] \rightarrow \{\text{columns}(T(X), c) \) and \([\text{identifier}(\text{anchor}), f] \rightarrow \{\text{attributes}(\text{s-tree S' corresponding to } \Phi(X, Y)), f \} \). It is easy to show that for a legal instance of \( T'(X) \), we can create a legal instance of \( \Phi'(X, Y) \), the new semantic tree, and vice versa.

If the column \( c \) is deleted from the table \( T(X) \), the mapping becomes \( \Phi(X, Y)'Y(X', Y') = T'(\text{columns}(T(X))-c) \), where \( Y(X', Y') \) contains the atoms referencing the attribute \( f \) corresponding to the column \( c \). If \( c \) is not a key or foreign key column, we can create a legal instance of \( T''(X) \) for each legal instance of \( \Phi'(X, Y) \) and vice versa. If \( c \) is a key column, the corresponding identifier constraint corresponding to the key constraint referencing \( c \) is deleted from the CM as well. Therefore, the mapping \( \Phi(X, Y)'Y(X', Y') = T'(\text{columns}(T(X))-c) \) is consistent. Similarly, we have consistent mapping if \( c \) is a foreign key.

**Case 2:** A new table \( T' \) is obtained by changes from multiple tables \( \{T_1, T_2, ..., T_n\} \). When new foreign keys are added to the new table \( T' \), Procedure 1 generates a larger s-tree \( S' \) for the new table \( T' \) by connecting the anchors of the skeleton trees \( \{S_1, S_2, ..., S_n\} \) corresponding to the keys of \( \{T_1, T_2, ..., T_n\} \) using functional edges. To prove that the resultant mapping between the new table \( T'' \) and the new s-tree \( S'' \) is consistent, we consider the set of legal instances of \( T' \) and the set of legal instances of \( S' \). Suppose that the functional relationships are added from the anchor of \( S_1 \) to the anchors of \( \{S_2, ..., S_n\} \). For any legal instance \( I' \) of \( T' \), the newly added foreign keys are accounted for by the functional relationships from the anchor of \( S_1 \) to the anchors of \( \{S_2, ..., S_n\} \). Therefore, we can create a legal instance of \( S' \) through the new mapping. Conversely, the functional relationships in the s-tree \( S' \) are accounted for by the foreign key constraints in \( T' \). Consequently, for any legal instance of \( S' \) we can create a legal instance of \( T' \) through the new mapping. Putting them together, we prove that the new mapping is consistent.

If multiple tables are merged into the new table \( T \) through the same key, Procedure 1
generates a new s-tree $S'$ by merging the corresponding s-trees by connecting them using ISA relationships. To prove that the resultant mapping between the new table $T'$ and the new s-tree $S'$ is consistent, we consider the set of legal instances of $T'$ and the set of legal instances of $S'$. Suppose that the ISA relationships are added from the anchor of $S_1$ to the anchors of $S', \ldots, S_j$. For any legal instance $I'$ of $T'$, the same key and merged columns are accounted for by the ISA relationships from the anchor of $S_i$ to the anchors of $S', \ldots, S_j$. Conversely, the ISA relationships in the s-tree $S'$ are accounted for by the key and merged columns in $T'$. Therefore, we can create legal instances for both cases. Putting them together, we prove that the new mapping is consistent.

If a foreign key becomes part of the key, Procedure 1 generates a new s-tree $S'$ by identifying the corresponding functional relationship in the CM, and updating the relationship from functional to non-functional if the original key is also a foreign key. To prove that the resultant mapping between the new table $T'$ and the new s-tree $S'$ is consistent, we consider the set of legal instances of $T'$ and the set of legal instances of $S'$. Suppose that the key of table $T$ corresponds to the skeleton tree $S$, and the foreign key corresponds to the skeleton $S'$, and the functional relationship is from the anchor of $S$, to the anchor of $S'$. The key of $T'$ is the combination of two foreign keys. The combination is accounted for by the non-functional relationship which is newly updated. Conversely, the non-functional relationship is accounted for by the key of the table $T'$ in which the key is the combination of two foreign keys. Therefore, for any legal instance $I'$ of $T'$, we can create a legal instance of $S'$, and vice versa. Putting them together, we prove that the new mapping is consistent.

**Case 3:** Multiple tables $\{T_p, T_q, \ldots, T_j\}$ are obtained from a single table $T$. There are two sub-cases: splitting a table and creating new tables connecting to the original table through foreign keys. Without losing generality, suppose $T_i$ inherits the key of $T$. Procedure 1 creates new concepts $\{C_p, \ldots, C_j\}$ in the CM for the new tables $\{T_p, T_q, \ldots, T_j\}$, respectively; and connects $C_i$ to $C_j$ by a functional edge in the CM if there is a foreign key constraint from the column $T_i.f$ to the key of $T_j$, or connects $C_i$ to $C_j$ by an ISA relationship if the column $T_i.f$ is also the key of the table $T_j$. To prove that the resultant mappings between the new tables $\{T_p, T_q, \ldots, T_j\}$ and the new s-trees are consistent, we consider the set of legal instances of each table $T_i$ and the set of legal instances of each s-tree $S_i$. The foreign key of $T_i$ referencing the key of $T_j$ is accounted for by the functional relationship from the anchor of $S_i$ to the anchor of $S_j$. If the key of $T_i$ is the same as the key of $T_j$, then the ISA relationships between the anchor of $S_i$ and the anchor of $S_j$ accounts for the semantics. Conversely, the foreign key and key constraints in the table $T_i$ account for the functional/ISA relationship in the skeleton tree. Therefore, for any legal instance $I'$ of $T_i$, we can create a legal instance of $S_i$, and vice versa. Putting them together, we prove that the new mappings are consistent.

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. The reason is that 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 same fashion as in 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 to future work.

Specifically, for a mapping $M=\{\phi(X, Y) = T(X)\}$ between a CM $C=(C, \Sigma_C)$ and a relational schema $R=(R, \Sigma_R)$, when the CM evolves, we update those mapping statements $\phi(X, Y) = T(X)$ that are directly affected by the changes in the CM. For a mapping $\phi(X, Y) = T(X)$, let $S$ be the s-tree corresponding to the
formula $\Phi(X, Y)$. $S$ may change due to several actions: deleting some elements from $S$, changing the identifier of the anchor, changing the cardinality constraints of some relationships of $S$, and restructuring $S$. If an element is deleted from $S$, we update the mapping expression by deleting the atoms referencing the element. If $S$ is restructured, we update the mapping expression by generating a new conjunctive formula from the new s-tree (An et al., 2006). If the cardinality constraints of a specific relationship of $S$ are changed, for example, from one to many or from many to one, then we drop the atoms in the mapping expression which reference the relationship. Finally, if the identifier of the anchor of $S$ is changed, we have to drop the mapping or mark the mapping as inconsistent.

The following Procedure 2 updates conceptual-relational mappings when the CMs evolve.

**Procedure 2 Maintain Mappings When CMs Evolve**

**Input:** A set of consistent conceptual-relational mappings $M = \{ \Phi(X, Y) = T(X) \}$ between a CM $C$ and a relational schema $R$; a set of correspondences $M'$ between attributes in $C$ and attributes in a new CM $C'$

**Output:** Update $M$ to a new set of mappings $M''$ between $R$ and $C'$.

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 which connects to the anchor through $p$. Update the mapping so that $T$ is mapped to the new smaller tree.
   c. **Else if** an element is deleted from the s-tree, then delete the atoms from $M$ which reference the element.
   d. **Else if** $S$ is restructured, generate a new conjunctive formula from the s-tree with the existing algorithm. /* see (An et al., 2006) for the algorithm.*/

**Example 8 [Change Cardinality Constraint].**

The following original mapping $M$ is a consistent mapping between a relational schema biosample(bsid, disease*, donor*) and an s-tree consisting of two functional relationships as shown in Figure 11.

In the table, the columns biosample.disease* and biosample.donor* are two foreign keys referencing the keys of the tables corresponding to the concepts Person and Disease. The foreign key constraints are consistent with the functional relationships between Biosample and the other two concepts, that is, Person and Disease.

If the upper bound cardinality imposed on the relationship disease from Biosample to Disease gets changed from 1 to many, then the mapping $M$ is updated, according to **step 2.b** of Procedure 2, to the following expression by dropping the disease relationship in order to maintain the property of consistency:

$$M : \text{Biosample}(x_1) \land \text{bsid}(x_1, \text{bsid}) \land \text{Person}(x_2) \land \text{originates}(x_1, x_2) \land \text{pid}(x_3, \text{donor}) \land \text{Disease}(x_3) \land \text{dsid}(x_3, \text{disease}) \land \text{disease}(x_1, x_3) = \text{biosample}(\text{bsid}, \text{disease}, \text{donor}).$$
The following proposition states the desired property of the Procedure 2 for maintaining a conceptual-relational mapping through updating the mapping when the associated CM evolves.

**Proposition 2.** Let the input of the Procedure 2 be (1) \( M = \{ \Phi(x, y) = T(x) \} \) which is a set of consistent conceptual-relational mappings between a CM \( C \) and a relational schema \( R \), and (2) \( M' \) which is a set of identify correspondences between attributes in \( C \) and attributes in a new CM \( C' \) evolved from \( C \). Each mapping in the set of conceptual-relational mappings returned by the Procedure 2 is consistent.

**Proof (Sketch).** We prove the proposition in a similar fashion as we did for Proposition 1.

By the definition of mapping consistency, a mapping \( M : \Phi(x, y) = T(x) \) between a CM \( C = (C, \Sigma_C) \) and a relational schema \( R = (R, \Sigma_R) \) is consistent, if and only if for every legal instance \( I \) of \( C \), there is a legal instance \( J \) of \( R \) such that \( \langle I, J \rangle \models M \), and vice versa. To prove the Proposition 2, we consider the set of mappings \( M' = \{ \Phi'(x, y) = T(x) \} \) between the new CM \( C' \) and the original schema \( R \) returned by the Procedure 2.

First of all, if \( M' = \{ \Phi'(x, y) = T(x) \} \) is a subset of the original mapping by dropping the inconsistent mapping statements due to changes to the identifier in the s-tree, then \( M' = \{ \Phi'(x, y) = T(x) \} \) is still consistent.

Let the mapping \( \Phi'(x, y) = T(x) \) be a mapping updated from an original mapping \( \Phi(x, y) = T(x) \). If \( \Phi'(x, y) \) is a sub formula of \( \Phi(x, y) \) by deleting some atoms, then \( \Phi'(x, y) = T(x) \) is still consistent.

Suppose \( \Phi'(x, y) \) is a formula generated from the new s-tree \( S' \) which is restructured from the original s-tree \( S' \). For a legal instance of \( S' \), we can create an instance of the table \( T(x) \) through the new mapping \( \Phi'(x, y) = T(x) \). Conversely, for a legal instance of \( T(x) \), we can create an instance of the s-tree \( S' \) through the new mapping \( \Phi'(x, y) = T(x) \). Because the constraints for both \( T(x) \) and the s-tree remain unchanged, the instance of \( S' \) created from a legal instance of \( T(x) \) still satisfies the constraints in \( S' \). Conversely, the instance of \( T(x) \) created from a legal instance of \( S' \) still satisfies the constraints in \( T(x) \). Therefore, \( \Phi'(x, y) = T(x) \) is consistent.

Overall, we prove that each mapping returned by Procedure 2 is consistent.

**EXPERIENCE**

To evaluate the performance of our round-trip engineering approach 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-fold: (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 (An et al., 2006) on the development of the MAONTO mapping tools generated conceptual-relational mappings for many of the test data. Subsequently, our other previous work (An et al., 2007) used the conceptual-relational mappings for improving traditional tools on constructing direct mappings between database schemas. It follows naturally to continue using 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 maintaining the consistency of tested mappings under *schema evolution*. For each mapping, we applied different types of changes to the relational table (Table 2). For each type of change, we ran the maintenance algorithm for measuring (1) execution time and (2) benefits of the round-trip engineering 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.

For measuring benefits, we compared the round-trip engineering process with the mapping reconstructing approach which discovers a new mapping from scratch. In our experiment, we used the MAPONTO (An et al., 2006) tool for the comparison. We adopted 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. (2004) and Yu & Popa (2005) applied a similar approach for measuring the benefits of mapping adaptation. Specifically, 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. Intuitively, user effort is related to the difficulty of a task. The more difficult a task is, the more effort a user has to put into performing the task. For the task of mapping maintenance/discovery, two quantities would be related to “difficulty.” The first is the number of correspondences a user has to specify before

---

**Table 2. Characteristics of test data**

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

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
starting the maintenance/discovery algorithm, and the second is the number of candidate mappings that the maintenance/discovery algorithm generates for a user to select the final ones. In our study, we used these two quantities for measuring “difficulty” or “user effort.” For a mapping \( \Phi(X,Y)=T(X) \) associating a semantic 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 correspondences between \( T' \) 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, the MAPONTO tool may be unable to generate 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 expression to measure the benefit:

\[
1 - \frac{\# \text{ mapping} \_\text{maintenance}}{\# \text{ mapping} \_\text{MAPONTO} + \# \text{ correspondences}}
\]

where the quantity \( \# \text{ mapping} \_\text{maintenance} \) is the number of candidate mappings generated by the round-trip engineering approach, the quantity \( \# \text{ mapping} \_\text{MAPONTO} \) is the number of candidate mappings generated by the MAPONTO mapping discovery tool, and the quantity \( \# \text{ correspondences} \_\text{MAPONTO} \) is the number of correspondences a user has to specify before starting the MAPONTO mapping tool. Because specifying correspondences between a schema and CM is much more costly than specifying correspondences between an evolved schema and the original schema, we omit the quantity for specifying evolution correspondences, that is, \( \# \text{ correspondences}_{\text{maintenance}} \), from the above expression.

**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 round-trip maintenance algorithm took less than one second to generate the results. This is comparable with the MAPONTO tool for discovering mappings between schemas and CMs. Next, in terms of benefits, Figure 12 presents the average benefits for the tested cases. The results show that the round-trip engineering process provides significant benefits in terms of maintaining the consistency of conceptual-relational mappings under schema evolution.

**DISCUSSION**

Database schemas play a critical role in data management. In practice, changes almost always occur to database schemas that have been populated with data instances. Previous study has been focused on the problem of data transformation (Haas et al., 1999), that is, migrating the data instances structured under an old schema to new data instances that conform to a new schema. Little attention has been paid to the problem of adopting the conceptual model associated with a database schema when the schema changes. In this paper, we have identified a number of practical applications where adopting conceptual models associated with schemas under schema evolution is important and has deep implications.

A common feature of these applications is that database schemas are associated with conceptual models through mappings. Mappings are expressed in terms of logical formulae. Such a mapping specifies a particularly
meaningful relationship between a database schema and a conceptual model, and is a key component in these applications. For example, object-relational mappings play a key role in many modern data-centric applications. When schemas or conceptual models evolve in order to accommodate new information needs, previously existing mappings often lose their “meanings” and need to be regenerated. In this paper, we proposed a novel approach for incrementally maintaining mappings under schemas and conceptual models evolution. The goal is to keep the mappings remain “meaningful” after the changes. Our hypothesis was that incrementally maintaining mappings would be more efficient and have “better” results than regenerating mappings from scratch.

We focused on a type of relationship which we define as consistent mapping. A consistent mapping permits a two-way translation between the instances of a conceptual model and a relational schema. When there are changes to either the schema or the CM, mapping maintenance becomes the problem of keeping the mapping remain consistent. A consistent mapping has a wide range of applications in practice. To achieve the maintenance goal, we proposed a round-trip engineering approach that can synchronize the CM/schema associated with a consistent mapping when either one evolves. We developed synchronization algorithms based on a careful study on the knowledge encoded in standard database design methodology.

The contributions of this research are manifold. Theoretically, we explicated the hidden knowledge encoded in the process of standard database design. The knowledge was largely ignored in previous studies and applications. We demonstrated in our round-trip algorithms that the explication of the knowledge provided opportunities of developing systematic approaches for dealing with mapping maintenance. We believe that the explicit knowledge can be leveraged in many problems involving database schemas and conceptual models. Practically, we studied an important problem to many applications. We proposed an effective and useful solution to the problem of maintaining object-relational mappings under schemas/CMs evolution. As opposed to current mapping regeneration practice, our experiments showed that the incremental maintenance approach provided significant benefits.

As the first study on maintaining mappings between conceptual models and database schemas, this research bears some limitations

---

Figure 12. Benefits of mapping maintenance approach in comparison to mapping reconstruction approach

![Benefits of mapping maintenance approach](image_url)
and points to several interesting future research directions.

The main limitation is that the paper addresses only one side of the problem caused by changes to database schemas, that is, the problem of synchronizing conceptual models and database schemas. We do not address the problem of data transformation when schemas change. If changes first occur to schemas, we will assume that underlying data has already been migrated and we focus on adopting the associated conceptual model and mapping. However, if changes occur to conceptual models, we only modify the mapping for maintaining consistency, leaving the schema unchanged to avoid data transformation. This limitation indicates that there is more work to do for synchronizing schemas when CMs evolve. It involves not only schema synchronization but also data transformation.

The second limitation of the current work is that it considers the common and relatively simple constraints such as primary and foreign key constraints and their corresponding conceptual counterparts. More complex constraints can be found in the literature such as in ORM conceptual schemas or in advanced ER schemas (Halpin & Morgan, 2008). It is also recognized in the literature that data transformation is, in general, nontrivial (Halpin & Morgan, 2008). Extending the current work to consider more complex constraints is listed in the future work directions. Moreover, as discussed above, combining CM/schema synchronization and data transformation is an interesting problem and will be investigated in the future as well.

CONCLUSION

A conceptual-relational mapping specifies a particularly meaningful relationship between a conceptual model (CM) and a relational schema. In this paper, we studied an important problem which is to maintain a consistent conceptual-relational mapping when the associated CM and schema evolve. We considered 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 and incrementally 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. Theoretically, our results showed that the process achieved desired properties for maintaining mappings. Experimental analysis demonstrated that the solution was efficient and provided significant benefits for maintaining conceptual-relational mappings in dynamic environments.

Future research directions include extending the round-trip engineering approach to more general conceptual-relational mappings such that a mapping statement may involve a sub-graph (not necessarily a tree) in a conceptual model and multiple tables in a relational schema. In addition, we also plan to extend the round-trip engineering approach by taking the following two situations into consideration: (1) relational schemas with other types of constraints such as general inclusion constraint, in addition to key and foreign key constraints; and (2) synchronizing relational schemas when the associated CMs change.

ACKNOWLEDGMENT

Yuan An is partially supported by NSF CCF-0905291. Xiaohua (Tony) Hu is supported by NSF CCF-0905291.

REFERENCES


Yuan An received a PhD degree from the University of Toronto in 2007 in Computer Science. He has been an assistant professor in the College of Information Science and Technology at Drexel University since that year. He has research interests in semantic technologies for information integration, information modeling including ontology design, schema/ontology mapping, and the Semantic Web. Dr. An designed and developed the MAPONTO tool for creating semantic mappings between ontologies and database schemas. Dr. An also has 10 years working experience in the Information Technology industry. As a principal developer and team leader, he designed and led the development of various management information systems. Dr. An has a Master’s degree in Computer Science from Dalhousie University, Canada. He also earned a Master’s degree in Electrical Engineering from Tsinghua University, China, in 1989 and a Bachelor’s degree in Electrical Engineering from the same Chinese university in 1987.

Xiaohua (Tony) Hu is currently an associate professor and the founding director of the data mining and bioinformatics lab at the College of Information Science and Technology at Drexel University. His current research interests are in biomedical literature data mining, bioinformatics, text mining, semantic Web mining and reasoning, rough set theory and application, information extraction, and information retrieval. He has published more than 160 peer-reviewed research papers in various journals, conferences and books such as various IEEE/ACM Transactions. He has received a few prestigious awards including the 2005 National Science Foundation (NSF) Career award, the best paper award at the 2007 International Conference on Artificial Intelligence, the best paper award at the 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology. He serves as the IEEE Computer Society Bioinformatics and Biomedicine Steering Committee Chair, and the IEEE Computational Intelligence Society Granular Computing Technical Committee Chair (2007-2008).
Il-Yeol Song is a professor of the College of Information Science and Technology at Drexel University. He received the M.S. and Ph.D. degrees from the Department of Computer Science, Louisiana State University, in 1984 and 1988, respectively. His research interests include conceptual modeling, object-oriented analysis & design, data warehousing, and bioinformatics. He has published over 160 peer-reviewed papers. He has won three teaching awards from Drexel University. He is a Co-Editor-in-Chief of Journal of Computing Science and Engineering (JCSE). He is also associate editors for many prestigious journals including International Journal of E-Business Research as well as on the Editorial Board of Data & Knowledge Engineering (DKE). Dr. Song is a steering committee member of several conferences including International Conference on Conceptual Modeling. He served as a program/general chair of 18 international conferences/workshops. He is also currently serving as the General co-Chair for International Conference on Information and Knowledge Management 2009.