A Vision for Management of Complex Models

Philip A. Bernstein
Microsoft Research
Redmond, WA 98052-6399
philbe@microsoft.com

Alon Y. Halevy
University of Washington
Seattle, WA, 98195
alon@cs.washington.edu

Rachel A. Pottinger
University of Washington
Seattle, WA, 98195
rap@cs.washington.edu

Abstract

Many problems encountered when building applications of database systems involve the manipulation of models. By "model," we mean a complex structure that represents a design artifact, such as a relational schema, object-oriented interface, UML model, XML DTD, web-site schema, semantic network, complex document, or software configuration. Many uses of models involve managing changes in models and transformations of data from one model into another. These uses require an explicit representation of "mappings" between models. We propose to make database systems easier to use for these applications by making "model" and "model mapping" first-class objects with special operations that simplify their use. We call this capability model management.

In addition to making the case for model management, our main contribution is a sketch of a proposed data model. The data model consists of formal, object-oriented structures for representing models and model mappings, and of high-level algebraic operations on those structures, such as matching, differencing, merging, selection, inversion and instantiation. We focus on structure and semantics, not implementation.

1 Introduction

Many of the problems encountered when building applications of database systems (DBMSs) involve the manipulation of models. By "model," we mean a complex discrete structure that represents a design artifact, such as an XML DTD, web-site schema, interface definition, relational schema, database transformation script, workflow definition, semantic network, software configuration or complex document. Many uses of models involve managing the change in models and the transformation of data from one model into another. These uses require an explicit representation of *mappings* between models. We believe there is an opportunity to make DBMSs easier to use for these applications by making "model" and "mapping" first-class objects with high-level operations that simplify their use. We call this capability *model management*.

This paper makes two main contributions. First, it argues that general-purpose model management functions are needed to reduce the amount of programming required to manipulate models. Second, it proposes a data model that captures model management functions. The data model consists of formal structures for representing models and mappings between models, and of algebraic operations on those structures. We present the overall shape of the data model in just enough detail to justify our thesis that general-purpose model management is a worthwhile and achievable goal. We expect the details of the data model will take years to work out.

Despite the functionality advances of relational and OO DBMSs, today's model management applications still include a lot of complex code for navigating graph-like structures. Producing, understanding, tuning, and maintaining navigational code is a serious drag on programmer productivity, making model management applications expensive to build.

To address these problems, we propose raising the level of abstraction beyond current DBMSs, by introducing high-level operations on models and model mappings, such as matching, merging, selection, and composition. These operations are not especially novel. There is research literature on each of them, in the DB field and elsewhere, much of which is relevant to the design and implementation of model management functions. We believe they can and should be generalized and integrated, to support a generic model management interface.

To illustrate the pervasiveness and scope of model management, we offer some examples of models and model mappings that arise in various applications:

- mapping an XML schema of one application to that of another in order to guide the exchange of XML instances between the applications;
- mapping a web site's content to its page layout in order to drive the generation of web pages;
- mapping data sources into data warehouse tables in order to generate programs that transform production data and load it into a data warehouse;

- mapping the DB schema of one software release into that of the next release, to guide the migration of DBs;
- mapping source makefiles into target makefiles in order to drive the transformation of make scripts and thereby help port complex applications from one programming environment to another; and
- mapping the components of a complex application to the components of a system where it will be deployed in order to drive the generation of installation, upgrade, and de-installation programs.

By building generic functions to create models and mappings and manipulate them as single objects, we can provide a better environment for the above tasks. At least initially, we expect that a model management system would primarily manipulate models whose home is on other platforms. Such a target platform could be another DBMS, a web site, an XML environment, a programming environment, etc. The glue between the systems is provided by simple adapters that (1) import or export a model in the model management system from or to a schema in the target platform, or (2) interpret a mapping in the model management system to transform instances of one target model to those of another. One of the many challenges in this area is to find appropriate architectures for coupling these systems.

The leverage of building model management functionality is that it is highly generic and therefore widely applicable. Still, to be competitive with more customized approaches, it must be specializable so it can exploit the semantics of a particular data type (e.g., SQL). Making it both generic and easily specializable is another challenge.

Model management applications are usually considered examples of "metadata management," where most of the effort in building the application is in manipulating *descriptions* of a thing of interest, rather than the thing itself. We intentionally avoid using the term metadata because it is so overloaded. One person's metadata is another person's data. Are keywords data or metadata? Model management takes a different cut at the problem. It focuses attention on a particular kind of metadata, structure and mathematical semantics of descriptive information. We see much leverage to be gained looking at this kind of metadata in isolation.

In describing the data model, we focus on structure and semantics. Although we have little to say here about its implementation, we emphasize that we see model management being implemented on top of advanced DB systems and logical inferencing engines. It is not a replacement for these technologies.

We begin with a scenario in Section 2, which both motivates the need for a coherent system and describes some of the models and operators that such a system should provide. In Sections 3, 4, and 5 respectively, we discuss the formal structure of models, mappings, and operations. Finally, in Section 6 we discuss how previous work relates to a model management system and how it can be used to tackle some of the many open questions.

An extended version of this paper appears in [BLP00].

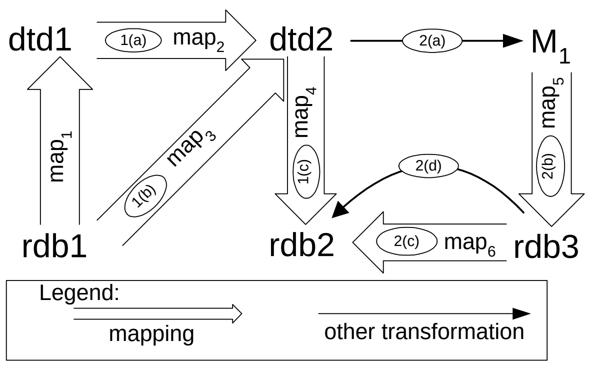
2 A Motivating Scenario

We begin by describing a scenario in which a model management system would play a central role and considerably reduce the coding burden of the application. We refer here to several model management operations whose semantics will be elaborated later.

Consider an online merchant selling books. The data about its titles, customers, and orders are stored in a relational DB, whose schema is represented as a model, rdb1. The relational data is mapped into an XML DTD, dtd1, to serve the data onto their web site and to share data in a marketplace of online merchants. The DTD conforms to the standard recommended by the marketplace participants. Denote the mapping from rdb1 into dtd1 by map_1.

In a model management system, rdb1, dtd1, and map_1 would be represented as first class objects. One scenario is that the company starts a partnership in a slightly different domain (e.g., CDs) with another online merchant, which exports its data in dtd2. In this case, the DTDs of the two companies differ; while they both talk about customers and orders, one has DTD elements concerning books while the other concerns CDs. Suppose our task is to create a relational schema rdb2 for the data from the CD merchant. We will do this by using model management operators to create the model for rdb2. In this case, we could proceed as follows (see Figure 1):

1. For the parts of the DTDs that match exactly, use the inverse of map_1 to create rdb2. We do this in three steps, using the operations on models and mappings:
   (a) Use the generic model matching function Match to create map_2 = Match(dtd1,dtd2), which is a mapping from dtd1 to dtd2 that identifies the maximal subsets of the two DTDs that exactly match.
   (b) Create $map_3 = map_1 \circ map_2$, the composition of map_1 and map_2, from rdb1 to dtd2. Since map_2 includes only exact matches, map_3 maps only to the subset of dtd2 that exactly matches dtd1.
   (c) Set $<map_4$, rdb2$>$ = DeepCopy(map_3^{-1}), which creates a new copy of map_3^{-1} and rdb1, called map_4 and rdb2 respectively. It is "deep" in the sense that it copies not only objects in map_3^{-1} but also the objects that map_3^{-1} connects to in rdb1.
2. For the parts of dtd2 that don't match dtd1, use a default mapping from DTDs to relational schemas. Using our operations, this can be done as follows:



1(a). map_2 = Match(dtd1, dtd2)
1(b). $map_3 = map_1 \circ map_2$
1(c). <map_4, rdb2> = DeepCopy(map_3^{-1})
2(a). M_1 = dtd2 − Range(Match(dtd1,dtd2))
2(b). map_5 = Default(M_1, rdb3)
2(c). map_6 = Match(rdb2, rdb3)
2(d). Merge(rdb2, rdb3, map_6)

Figure 1: An example usage of model management operations.

(a) Use the set-difference operator to create a model M_1 representing the part of dtd2 that doesn't match with dtd1: M_1 = dtd2 − Range(Match(dtd1,dtd2)).

(b) Instantiate the default DTD-to-RDB transformation by calling map_5 = Default(M_1, rdb3), which creates a mapping map_5 and a relational schema rdb3 with new names. The domain of map_5 is M_1 and its range is rdb3.

(c) Create map_6 = Match(rdb2, rdb3) to align the overlapping tables and keys of the two schemas, in preparation for merging them.

(d) Call Merge(rdb2, rdb3, map_6) to merge rdb3 into rdb2 based on map_6.

Other model management operations can be useful in complications of the above situation:

- If there are differences even in the common parts of the DTDs, we would like the match function to propose a set of possible matches between the two DTDs and rank them. The engineer can then choose the appropriate one and, possibly, modify it manually. The system should also be able to take input from the engineer that constrains the possible matches between the two DTDs.
- After creating rdb2, we have two separate relational DBs for the two companies. Suppose that later on, one company acquires the other, and therefore wants to merge their DBs. In this case, we would like a Merge(rdb1,rdb2) operation to propose a relational schema that merges the original ones. This is more complex than 2(c,d) above, because the DB schemas are probably not disjoint and may have different internal structure. Furthermore, we would like the system to automatically generate a mapping from the old schema to the merged one, to facilitate the migration of applications to the new merged company.
- There may be several possible mappings from XML DTDs to relational schemas. In this case, instead of using Default(dtd,rdb), we may try several mappings and pose *what-if* queries on their results. In particular, we would want to estimate the cost of processing certain queries on the resulting schemas, using a generic function Estimate(queries, schema).

Companies are facing an increasing need to share data with others in various contexts, especially with the growth of business-to-business online applications. Hence, these operations are becoming more important all the time.

The above scenario might sound like pie-in-the-sky that is hopeless to achieve. For example, it sounds extremely hard to develop a generic algorithm that finds the *best* match of two distinct models or that inverts an *arbitrary* mapping. But optimal and complete algorithms are not essential ingredients for success. Success would be realized by a platform for manipulating models and mappings using high level operators that users can further customize manually. Since there are many published, mostly-heuristic algorithms for all the above operations (cf. Section 6), this goal seems well within reach.

3 Models

This section describes a first attempt at a data model for model management. We begin by discussing models. In the next section we discuss mappings between models.

At an abstract level, we think of models as labeled directed graphs. An object-oriented data model is therefore the natural platform on which to define model management functions. Almost any object model will do. In this paper, we use the following simple one: Each object has a set of scalar-valued properties whose values describe the state of the object. Each object also has relationship properties, each of which contains a set of binary relationships. Each relationship is a pair of object references directed from its *origin* object to its *destination* object, but it can be traversed in either direction. As in ODMG [CCB$^+$00], relationships are not first-class objects.

We assume each object is an instance of a class. Each class definition describes the set of properties and relationship properties that are defined for instances of that class. Since relationships are binary, each relationship property definition has an inverse relationship property definition on the class to which it connects and is labelled as either the origin or destination side.

We assume that class definitions are objects. This makes the data model self-describing. Thus, a model can be a mixture of class definitions and ordinary objects.

We use *database* to refer to a set of objects of interest. It may or may not be persistent.

Model contents: A model is a composite object containing objects and relationships. Which objects and relationships? We could make this explicit by representing a model by a particular object that has a relationship to every object in the model. But this is inconvenient, since every time a submodel is added to a model, a relationship would have to be added from the model object to every object in the submodel. To avoid this, we use ordinary relationships in a model to define the contents of the model.

We could say that a model is simply the set of objects reachable from its root. Thus, when adding a relationship from an object in a model to some submodel, all of the objects in the submodel immediately become part of the model. Unfortunately, this doesn't quite work, because some relationships are between pairs of objects in different models. Our solution is to distinguish between relationships that imply containment within the model from those that do not. This is similar to the representation of complex objects in some OO DBMSs. A model, then, is the transitive closure of containment relationships emanating from the model's root. Formally, we assume that each relationship property definition in the schema includes a $containment$ flag, which is set to True for containment relationships, and we define a model as follows:

Definition 1 A $model$ is a set of objects O that is identified by a $root\ object$, r in O, such that $O - \{r\}$ is the set of objects that are reachable by following containment relationships from r. □

In our experience, it is worth restricting the containment flag to be settable only on the origin side of a relationship property, and to require that containment relationships in a model constitute a directed acyclic graph [BBC$^+$99]. This makes containment correspond to the intuitive notion of set containment. It simplifies the maintenance of a materialized closure, which enables the content of a model to be identified efficiently [DS00]. We expect it also simplifies algorithms for propagating delete, copy and other operations on models (cf. Section 5.1).

Challenge 1 Develop a mechanism for representing models and for storing these representations. One key issue is how much of a model's semantics is expressed in its representation. For example, an integrity constraint for a relational schema can be represented as a string-valued property, or as a logical formula whose structure and interpretation is known to the model management system. Another issue is how much of the semantics of the model to describe. Storing and indexing models also raises challenges. Different storage schemes can be devised for building a model management system over an OO, object-relational or other DBMS. □

4 Mappings Between Models

A key goal of model management is to provide support for managing change in models and for mapping data between different models. Hence, we believe it is crucial that model mappings be manipulated as first-class citizens. Before describing our representation of model mappings, we outline the key elements underlying our approach to modeling mappings.

- We need to manipulate model mappings much like we manipulate models: copy a mapping, delete a mapping, select from a mapping, etc. To avoid defining separate elementary operations on mappings, we require that a mapping be a model. This also allows us to have mappings between mappings.
- A mapping consists of connections between instances of two models, often of different types (e.g., a mapping between a relational schema and an XML DTD). While we could allow a mapping to connect more than two models, this adds complexity to the data model and is unnecessary for the applications we know of.
- There may be more than one mapping between a given pair of models. For example, two different mappings between a relational schema and an XML DTD can represent two different ways to encode instances of the relational schema as instances of the DTD.
- A mapping may relate a *set* of objects in one model to a set of objects in another via a language for building complex expressions. For example, a mapping between relational schemas M_1 and M_2 may specify that view V_1 over M_1 corresponds to view V_2 over M_2, where the view definitions are part of the mapping. Moreover, a model may have an associated language for building expressions over elements in the model, such as a query language or arithmetic expressions.
- Mappings must be able to nest. This enables the reuse of mappings: a mapping on a model M to be used as a component of a mapping on models that contain M.

Given these points, we define model mappings as follows:

Definition 2 A model mapping map from a model M_1 to a model M_2 is a model where:

1. map has a distinguished root element that has two single-valued relationship properties, domainRoot and rangeRoot, which point to the root objects of M_1 and M_2, respectively. These properties identify the models being related by the mapping.

2. Each object in map (called a *mapping object*) has a property Expr, which is an expression over the objects of M_1 and M_2.
3. Each mapping object in map has two relationship properties, domain and range, which include the objects of M_1 and M_2 (resp.) referred to in Expr. □

Although a mapping must be a model, the definition of mapping says nothing about the mapping's containment relationships. This degree of freedom is appropriate because different applications will want to structure mappings in different ways. For example, a mapping between two very similar XML DTDs could mimic the structure of the DTDs being related. By contrast, a mapping between two highly dissimilar DTDs might be flat, where all objects in the mapping are children of the root, since the mapping is not preserving much of the DTDs' structure.

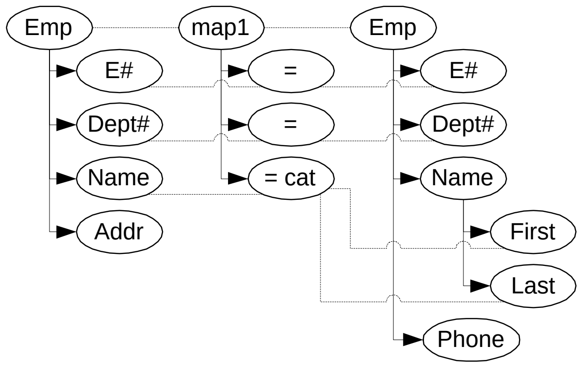


Figure 2: A mapping between a relational schema and DTD

Example 1 Figure 2 shows a mapping between a relational schema and DTD. The root of map1 connects the root of the relational schema and XML DTD, as required by Definition 2. The mapping objects' Expr properties are abbreviated by the operation that relates the domain and range. □

An immediate advantage of the explicit representation of models and mappings is that it allows us to pose queries such as "Find all attributes in the XML DTD that map to members of keys in the relational model." Since we are formulating our model as a graph, we can answer essentially any queries that follow regular expressions. This is similar to OQL and many XML query languages.

The property Expr is a placeholder for specifying semantic details. We do not require a specific representation of the mapping's semantics. It could be a string, or it could be the root object of a complex structure that represents the expression. This raises the following challenge:

Challenge 2 Find appropriate representations of model mappings that trade off expressiveness of semantics with the goal of keeping operations generic across a wide set of model types. □

Directionality of mappings: A mapping is directional if it specifies how to transform data from its *domain* to its *range*. Some operations require mappings to be directional, such as some kinds of composition. Others do not, such as differencing. We therefore defined mappings so that they can represent directionality but do not require it. To help manage directionality when it is required, we envision an orientation operator that takes as input a mapping and produces a directional mapping from its domain to its range whenever possible (and returns the best approximation otherwise). The issue of directionality raises another model management challenge, which can be instantiated for different model and mapping types:

Challenge 3 Develop an algorithm that given a mapping m between two models M_1 and M_2 produces the best directional mapping m' from M_1 to M_2. □

5 Operations on Models

Recall that our main goal in creating a data model for model management is to reduce the amount of programming required to manipulate persistent models. We do this by defining a set of high-level algebraic operations on the two main structures of interest, models and mappings. We have two requirements for these operations. First, each operation should return a model, so that operations can be composed. Second, each operation should be generic, so it works for any type of model or model management application. The overall challenge is:

Challenge 4 Design an algebra of useful, composable operations on models. Consider combinations of operations as well as efficient implementations. □

5.1 Elementary Operations

Standard operations are needed to manipulate models, such as create, update, delete, select, project, setDifference, applyFunction and copy. Their definitions are fairly standard, but see [BLP00, BR00] for some additional wrinkles introduced in the context of model management.

A particularly interesting operation is model enumeration. Although our goal is to capture as much model management functionality as possible in set-at-a-time operations, there will undoubtedly be times when the objects of a model need to be navigated one-by-one. Since a model is a set of objects, one could navigate it using the data manipulation language of the underlying DBMS. However, we can offer a simpler interface by defining an Enumerate operation that performs the traversal, thereby bridging the programming gap between models and objects within models. Enumerate takes a model, M, as parameter plus directives regarding the order in which objects should be traversed. It returns a cursor object, which can be used by get-next operations. There are many useful directives that Enumerate could offer, such as depth-first vs. breadth-first and pre-order vs. post-order traversal. Enumerate should use the containment relationships to guide the traversal, to avoid straying outside the model. It is conceivable to use non-containment relationships instead, but this introduces a potentially expensive model membership test as each object is returned by a get-next.

5.2 Matching and Differencing

Since many applications of model management involve tracking changes in models, a key operation to consider is one that accepts as input two models, and returns the mapping that describes the *best* match between them. Unlike the operations described in the previous section, the output of a match operation is often just an educated guess made by the system, based on examining the schema and integrity constraints of the models, or by inspecting data instances of the two models. Such a guess gives an engineer a starting point for designing a best match.

There are different flavors of the problem of finding matches between models, depending on whether they focus on the commonalities or on the differences.

1. Find the *best mapping* between two models. In Section 2, we used such an operator to find the parts of two DTDs that match.
2. Find the *difference* between two models. This is basically the same as matching, except that the answer needs to highlight the differences.
3. The engineer may wish to find a best match that is consistent with a priori knowledge about the mapping. In some cases, the knowledge may concern which objects in one model *should not* be mapped to the other model. For example, a designer may know that "phone number" in one schema means home phone while in the other it means business phone.
4. We may already have a partial mapping m between models M_1 and M_2 and would like the system to find the best complete mapping that extends m.

To summarize, the following is a key model management challenge, predicated on the specific type of model.

Challenge 5 Develop algorithms for finding the best matches between two models. Specifically, given two models M_1 and M_2, and a partial mapping m that specifies a priori knowledge about the mapping from M_1 to M_2, find an extension m' of m that is the best complete mapping from M_1 to M_2. □

Matching: A Match operation should be generic, so it can apply to any type of model. One approach is to obtain a generic Match by encapsulating object mapping criteria in a *similarity relation*, $\cong$, over pairs of objects. A simple semantics for $\cong$ is type and value equality.

We can define a Match operation that takes as input two models and a $\cong$ relation and produces a mapping that is consistent with $\cong$. Each object in the mapping returned by Match should include an expression that describes how the domain and range objects are related. Ideally, we would like a generic Match algorithm that treats the mathematical system as a black box that it calls to generate a formula (e.g., a view definition) whenever it identifies a combination of domain and range objects that match. Or, we may need a repertoire of Match algorithms, each customized to a particular formal system.

The result of Match could be flat where all objects in each mapping are children of the root. More likely, a user would want a mapping to mimic the structure of the models being matched. By mimicking the model structure in the mapping, an application can more easily navigate the mapping systematically.

Challenge 6 What are good structures for the result of matching models that have different structure? □

Differencing: The differencing operation is essentially a Full OuterMatch, since the latter identifies objects that appear in one model but not the other, i.e., were inserted or deleted. The expression in each object of the map returned by Full OuterMatch is still some form of equivalence that explains how the domain and range objects are related. For example, the expression may say that domain and range objects represent the same conceptual object, but some of their property values or relationships differ. Such an expression has the same meaning as an ordinary Match, where sameness rather than difference is being emphasized. However, it may be desirable to phrase the expression in a way that emphasizes difference rather than sameness. If a mapping object has an empty domain or range, then presumably its associated expression is null.

Challenge 7 Should Diff a separate operation or a specialization of OuterMatch? Is there a useful way to express a transformation sequence as a model? Should script generation be encapsulated as a generic operation? □

5.3 Merge

Merging is the activity of moving the content of a target model into a source model. If the source and target models are disjoint, this amounts to taking the union of the source and target models and assigning it to the source model. In this case, if the roots of the source and target are identical or if they are merely placeholders under which to hang the model, then merging can be achieved by connecting the root of the target model with the children contained by the root of the source model. That is, the effect of Merge(M_1, M_2) is to make a copy of all of the outgoing containment relationships from the root of M_2 and connect them to the root of M_1. For example, this semantics would satisfy the needs of steps 2(b) and 2(c) in Section 2.

When the source and target are not disjoint, Merge faces several issues:

1. Avoiding the creation of duplicate copies of source objects that are already present in the target, and choosing property values for such objects when the source and target state differ.
2. Inserting source objects and relationships that are not present in the target.
3. Possibly deleting target objects and relationships that are not in the source.

Addressing these issues in a generic Merge operation is hard, because many variations are possible, depending on assumptions about the structures being merged. One way to address them is to have Merge take a third parameter which is a delta of the target and source, and use it to drive the merge activity, following the interpretation presented for the definition of potential delta.

Challenge 8 Propose a semantics for Merge that addresses the above issues and is sufficiently generic to subsume most of the known semantic variations. □

5.4 Mapping Composition

Composition of mappings is relatively easy to define for mappings that are single-valued functions, since ordinary function composition semantics works well. However, we want the result to be a model, so we need containment relationships that connect the objects in the resulting mapping. One approach is to use the containment relationships of one of the two mappings involved in the composition. For example, consider single-valued functions $map_1{:}M_1 \rightarrow M_2$ and $map_2{:}M_2 \rightarrow M_3$. The result of composing map_1 and map_2, denoted $map_1 \circ map_2$, is equivalent to the following:

1. Create a shallow copy map_3 of map_1 (i.e., copy the map and its relationships, but not the objects it connects to).
2. For each object $o \in map_3$, if $\exists$ object $q \in map_2$ with $q.domain = o.range$, then set $o.range = q.range$ (i.e., replace $o.range$ by $q.range$). Otherwise set $o.range = \phi$.

Notice that map_3 includes a mapping object for every object in map_1. However, it only includes the range of an object o_2 of map_2 if an object in map_1 composes with o_2. An alternative, equally useful semantics is to guide the above procedure by map_2 instead of map_1. Then the resulting mapping will include every object in map_2 but not map_1. The latter is what is needed for step 1(b) of Section 2, where we do not want the result of the composition to contain any object o for which $o.range = \phi$ (i.e., any object in dtd2 that does not match dtd1).

This definition works even if each object o in map_1 maps a set of objects in M_1 to an object in M_2 (e.g., maps a set of web pages to a relation that they reference). Allowing $o.range$ to be set-valued is more problematic. There are (at least) two ways to interpret $o.range$:

1. Treat map_1 as a single-valued function whose output is a set. The above definition of composition works in this case.
2. Treat map_1 as a multi-valued function whose output is a set of individual objects. So in step (2) of the above definition, we can compose o with a set of objects in map_2 the union of whose domains are covered by $o.range$.

Both semantics appear to be useful, so variations of the composition operation are needed for each of them (see [BR00] for details). Finally, many interesting questions arise when mappings relate complex expressions over the two models, and when we consider composition of non-directional mappings.

6 Related Work

Model management is not an isolated area of research. The DB literature already deals with many aspects of the problem. Model management offers opportunities to extend these works in new directions. This section describes how some of that work fits into the overall vision.

Platforms for model management systems: Many of today's advanced DB architectures and features are relevant to developing an appropriate platform for model management. A model management system should be implemented on a platform that includes OO and object-relational functionality. This will enable it to exploit the usual OO features (inheritance, encapsulation, polymorphism), recursive queries, an extensible set of algebraic operations, an extensible query optimizer, etc. Some model management functions are likely to run faster in client cache than on a server, which is more conducive to today's OO DBs than object-relational ones. Models are usually versioned, making techniques from temporal DBs of interest. Other architectures that combine OO and deductive capabilities in sophisticated ways can also provide significant benefits to model management such as, Telos [MBJK90], ConceptBase [JJ89], F-Logic [KLW95], and Description Logics [Bor95].

Inferencing in model management: Several key operations in model management involve various forms of inference, such as inverting a mapping, completing a mapping, and determining equivalence of models. For example, a mapping can be thought of as a view of one model in terms of another. Therefore, inverting a mapping resembles the problem of inverting views, which raises the relevance of work on answering queries using views [Hal01]

Efficient operations on models: A model is the transitive closure of a graph. In many scenarios, it will be important to compute this closure and/or maintain it as a materialized view [ADJ90, AJ87, DP97, DR94, DS95, Jag90, IRW93]. Probably, some structure in models can be exploited to improve upon published techniques. Transitive closure is also closely related to recursive query processing [BMSU86, BR86, DR94]. It will be important to learn how best to map model management functions on top of recursion, which is now a part of SQL3.

Differencing models: In many cases, differencing models is a lot like differencing graph structures. As shown in [CRGMW96, CGM97], its computational complexity is sensitive to assumptions about the kind of structure that the graph can represent and the available mapping operations. This suggests it will be hard to develop generic algorithms for differencing that are parameterized by the kinds of structures of interest. Luckily, there is a substantial research literature on differencing that can be leveraged to understand the variations that need to be covered by a generic solution, or even to understand if a generic solution is possible [Mye86, SZ90, WSC$^+$97, ZWS95].

Modeling change in models: One of most longstanding topics of DB research is data translation [BKKK87, LCC94, SHT$^+$77]. Recent techniques, such as those of [CJR98, MZ98], are excellent test cases for a generic model management system.

Schema integration: There are many approaches to schema integration which are candidate algorithms for Match [BCV99, JMN$^+$99, MHH00, MWK00, PSU98, MMP95, DDL00]. Information capacity of models [MIR93] may also be key to comparing among models.

7 Final Remarks

In this paper, we presented an outline of a data model for model management, which has two main abstractions

- *model*, which captures the structure of engineered information artifacts, such as database schemas, interface definitions, XML DTDs, web site designs, semantic networks, complex documents, and software configurations, and
- *mapping*, which captures relationships between models such as transformations and matchings.

The data model includes high-level set-oriented operations that manipulate models and mappings as first class objects, such as copy, select, delete, apply-function, enumerate, compose, match, and merge. These operations should greatly reduce the amount of code required for applications that manipulate models and mappings. Moreover, they should enable people to write model manipulation applications that today seem too daunting.

A modern database system supporting object-oriented or object-relational functions should be a very suitable platform on which to implement a model management data model. However, producing such an implementation is not a cake walk. As pointed out in this paper, there are many technical challenges in developing a generic, customizable and efficient implementation. Long term, we expect that a solution to these challenges will result in a substantial layer of software that we can properly think of as a new kind of database system.

Applications that manipulate models are complicated and hard to build. By implementing generic model management functionality along the lines presented in this paper, the database field stands a good chance of improving programmer productivity for these applications by an order of magnitude. It is an exciting prospect.

8 Acknowledgements

We are grateful for many useful suggestions from Thomas Bergstraesser, Zack Ives, Sergey Melnick, John Mylopoulos, Arnie Rosenthal, and especially Erhard Rahm.

References

[ADJ90] Rakesh Agrawal, Shaul Dar, and H. V. Jagadish. Direct transitive closure algorithms: Design and performance evaluation. *TODS*, 15(3):427–458, 1990.

[AJ87] Rakesh Agrawal and H. V. Jagadish. Direct algorithms for computing the transitive closure of database relations. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *Proc. of VLDB*, pages 255–266. Morgan Kaufmann, 1987.

[BBC$^+$99] P. A. Bernstein, T. Bergstraesser, J. Carlson, P. Sanders S. Pal, and D. Shutt. Microsoft repository version 2 and the open information model. *Information Systems*, 24(2):71–98, 1999.

[BCV99] Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, 1999.

[BKKK87] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proc. of SIGMOD*, pages 311–322, 1987.

[BLP00] Philip A. Bernstein, Alon Y. Levy, and Rachel A. Pottinger. A vision for management of complex models. Technical Report MSR-TR-2000-53, Microsoft Research, Available from http://research.microsoft.com/pubs/., 2000.

[BMSU86] François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proc. of PODS*, pages 1–16, 1986.

[Bor95] Alex Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[BR86] Franois Bancilhon and Raghu Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *Proc. of SIGMOD*, pages 16–52, 1986.

[BR00] Philip A. Bernstein and Erhard Rahm. Data warehouse scenarios for model management. In *Proceedings of the Entity Relationship Conference*, pages 1–15. Springer Verlag, 2000.

[CCB$^+$00] R.G.G. Cattell, Rick Catell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez, editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers, 2000.

[CGM97] Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful change detection in structured data. In *Proc. of SIGMOD*, 1997.

[CJR98] Kajal T. Claypool, Jing Jin, and Elke A. Rundensteiner. Serf: Schema evalution through an extensible, reusable and flexible framework. In *Proc. of CIKM*, pages 314–321, 1998.

[CRGMW96] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proc. of SIGMOD*, 1996.

[DDL00] Anhai Doan, Pedro Domings, and Alon Y. Levy. Learning source descriptions for data integration. In *In Proc. of WebDB*, 2000.

[DP97] Guozhu Dong and Chaoyi Pang. Maintaining transitive closure in first order after node-set and edge-set deletions. *Information Proc. Letters*, 62(4):193–199, 1997.

[DR94] Shaul Dar and Raghu Ramakrishnan. A performance study of transitive closure algorithms. In *Proc. of SIGMOD*, pages 454–465, 1994.

[DS95] Guozhu Dong and Jianwen Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Information and Computation*, 120(1):101–106, 1995.

[DS00] G. Dong and J. Su. Incremental maintenance of recursive views using relational calculus / SQL. *SIGMOD Record*, pages 44–51, 2000.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *To appear in the VLDB Journal*, 2001.

[IRW93] Yannis E. Ioannidis, Raghu Ramakrishnan, and Linda Winger. Transitive closure algorithms based on graph traversal. *TODS*, 18(3):512–576, 1993.

[Jag90] H. V. Jagadish. A compression technique to materialize transitive closure. *TODS*, 15(4):558–598, 1990.

[JJ89] Matthias Jarke and Manfred A Jeusfeld. Rule representation and management in ConceptBase. *SIGMOD Record*, 18(3):46–51, 1989.

[JMN$^+$99] Jan Jannink, Prasenjit Mitra, Erich Neuhold, Srinivasan Pichai, Rudi Studer, and Gio Wiederhold. An algebra for semantic interoperation of semistructured data. In *IEEE Knowledge and Data Engineering Exchange Workshop (KDEX)*, 1999.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.

[LCC94] Chien-Tsai Liu, Shi-Kuo Chang, and Panos K. Chrysanthis. Database schema evolution using EVER diagrams. In *Proceedings of the Workshop on Advanced Visual Interfaces*, pages 123–132, New York, New York, USA, 1994.

[MBJK90] John Mylopoulos, Alexander Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: Representing knowledge about information systems. *ACM TOIS*, 8(4):325–362, 1990.

[MHH00] Renee J. Miller, Laura Haas, and Mauricio Hernandez. Schema mapping as query discovery. In *Proc. of VLDB*, 2000.

[MIR93] Renne J. Miller, Yannis E. Ioannidis, and Raghu Ramakrishnan. The use of information capacity in schema integration and translation. In *Proc. of VLDB*, pages 120–133, 1993.

[MMP95] John Mylopoulos and Renate Motschnig-Pitrik. Partitioning information bases with contexts. In *Proceedings of 3rd CoopIS*, pages 44–54, 1995.

[MWK00] Prasenjit Mitra, Gio Wiederhold, and Martin L. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proc. of EDBT*, pages 86–100, 2000.

[Mye86] E. Myers. An o(nd) difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986.

[MZ98] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. of VLDB*, New York City, USA, 1998.

[PSU98] L. Palopoli, D. Saccà, and D. Ursino. Semi-automatic, semantic discovery of properties from database schemes. In *Proceedingsof IDEAS'98*, pages 244–253. IEEE Press, Cardiff, United Kingdom, 1998.

[SHT$^+$77] N.C. Shu, B.C. Housel, R.W. Taylor, S.P. Ghosh, and V.Y. Lum. Express: A data extraction, processing and restructuring system. *ACM Transactions on Database Systems*, 2(2):134–174, 1977.

[SZ90] Dennis Shasha and Kaizhong Zhang. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4):581–621, 1990.

[WSC$^+$97] Jason Tsong-Li Wang, Dennis Shasha, George Jyh-Shian Chang, Liam Relihan, Kaizhong Zhang, and Girish Patel. Structural matching and discovery in document databases. In *Proc. of SIGMOD*, pages 560–563, 1997.

[ZWS95] Kaizhong Zhang, Jason Tsong-Li Wang, and Dennis Shasha. On the editing distance between undirected acyclic graphs and related problems. In *Proceedings of CPM*, pages 395–407, 1995.