Merging Models Based on Given Correspondences

Rachel A. Pottinger
University of Washington
Seattle, WA 98195-2350 USA
rap@cs.washington.edu

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

Abstract

A model is a formal description of a complex application artifact, such as a database schema, an application interface, a UML model, an ontology, or a message format. The problem of merging such models lies at the core of many meta data applications, such as view integration, mediated schema creation for data integration, and ontology merging. This paper examines the problem of merging two models given correspondences between them. It presents requirements for conducting a merge and a specific algorithm that subsumes previous work.

1 Introduction

A *model* is a formal description of a complex application artifact, such as a database schema, an application interface, a UML model, an ontology, or a message format. The problem of merging models lies at the core of many meta data applications, such as view integration, mediated schema creation for data integration, and ontology merging. In each case, two given models need to be combined into one. Because there are many different kinds of models and applications, this problem has been tackled independently in specific domains many times. Our goal is to provide a generic framework that can be used to merge models in all these contexts.

Combining two models requires first determining correspondences between the two models and then merging the models based on those correspondences. Finding correspondences is called schema matching; it is a major topic of ongoing research and is not covered here [8-11]. Rather, we focus on the problem of combining the models after correspondences are established. We encapsulate the problem in an operator, Merge, which takes as input two models, A and B, and a mapping Map_{AB} between them that embodies the given correspondences. It returns a third model that is the "duplicate-free union" of A and B with respect to Map_{AB}. This is not as simple as set union because the models have structure, so the semantics of "duplicates" and duplicate removal may be complex. In addition, the result of the union can manifest constraint violations, called *conflicts*, that Merge must repair.

An example of the problems addressed by Merge can be seen in Figure 1. It shows two representations of Actor, each of which could be a class, concept, table, etc. Models A and B are to be merged. Map_{AB} is the mapping between the two; relationships relating the models are shown by dashed lines. In this case, it seems clear that Merge is meant to collapse A.Actor and B.Actor into a single element, and similarly for Bio. Clearly, A.ActID should be merged with B.ActorID, but what should the resulting element be called? What about the actor's name? Should the merged model represent the actor's name as one element (ActorName), two elements (FirstName and LastName), three elements (ActorName with FirstName and LastName as children), or some other way?

These cases of differing representations between input models are called *conflicts*. For the most part, conflict resolution is independent of the representation of A and B. Yet most work on merging schemas is data-model-specific, revisiting the same problems for ER variations [19], XML [3], data warehouses [7], semi-structured data [4], and relational and object-oriented databases [6]. Note that these works, like ours, consider merging only the models, not the instances of the models. Some models, such as ontologies and ER diagrams, have no instance data.

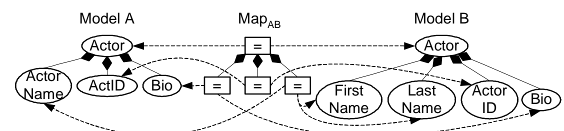


Figure 1: Examples of models to be merged

The similarities among these solutions offer an opportunity for abstraction. One important step in this

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment

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

direction was an algorithm for schema merging and conflict resolution of models by Buneman, Davidson, and Kosky (hereafter *BDK*) [6]. Given a set of pair-wise correspondences between two models that have Is-a and Has-a relationships, BDK give a formal definition of merge and show how to resolve a certain kind of conflict to produce a unique result. We use their theoretical algorithm as a base, and expand the range of correspondences, model representations, conflict categories, and applications, yielding a robust and practical solution.

Merge is one of the operators proposed in [5] as part of *model management*, a framework that consists of operators for manipulating models and mappings. Other model management operators include: Match, which returns a mapping between two given models; Apply, which applies a given function to all the elements of a model; and Diff, which, given two models and a mapping, returns a model consisting of all items in the first model that are not in the second model [5].

The main contribution of this paper is the design of a practical generic merge operator. It includes the following specific contributions:

- Technical requirements for a generic merge operator.
- The use of an input mapping that is a first-class model, enabling us to express richer correspondences than previous approaches.
- A characterization of when Merge can be automatic.
- A taxonomy of the conflicts that can occur and a definition of conflict resolution strategies using the mapping's richer correspondences.
- Experimental evaluation showing that our approach scales to a large real world application.
- An analysis that shows our approach subsumes previous merge work.

The paper is structured as follows: Section 2 gives a precise definition of Merge. Section 3 describes our categorization of conflicts that arise from combining two models. Section 4 describes how to resolve conflicts in Merge, often automatically. Section 5 defines our merge algorithm. Section 6 discusses an alternate merge definition and how to simulate it using Merge and other model management operators. Section 7 evaluates Merge experimentally by merging two large anatomy databases and conceptually by showing how our approach subsumes previous work. Section 8 is the conclusion.

2 Problem Definition

2.1 Representation of Models

Defining a representation for models requires (at least) three meta-levels. Using conventional meta data terminology, we can have: a *model*, such as the database schema for a billing application; a *meta-model*, which consists of the type definitions for the objects of models, such as a meta-model that says a relational database schema consists of table definitions, column definitions, etc.; and a *meta-meta-model*, which is the representation language in which models and meta-models are expressed.

The goal of our merge operator, Merge, is to merge two models based on a mapping between them. For now, we discuss Merge using a small meta-meta-model (which we extend in Section 4.1). It consists of the following:

- *Elements* with semi-structured properties. Elements are the first class objects in a model. Three properties are required: Name, ID, and History. Name is self-explanatory. ID is the element's unique identifier, used only by the model management system. History describes the last operator that acted on the element.
- Binary, directed, kinded *relationships* with cardinality constraints. A relationship is a connection between two elements. Relationship kinds define semantics, such as Is-a, Has-a, and Type-Of. Relationships can be either explicitly present in the model or *implied* by a meta-meta-model's rule, such as "a is a b" and "b is a c" implies that "a is a c." Relationship cardinalities are omitted from the figures for ease of exposition.

In Figure 1 elements are shown as nodes, the value of the Name property is the node's label, mapping relationships are edges with arrowheads, and sub-element relationships are diamond-headed edges.

2.2 Merge Inputs

The inputs to Merge are the following:

- Two models: A and B.
- A mapping, Map_{AB}, which is a model that defines how A and B are related.
- An optional designation that one of A or B is the *preferred model*. When Merge faces a choice that is unspecified in the mapping, it chooses the option from the preferred model, if there is one.
- Optional overrides for default Merge behavior (explained further below).

The input mapping is more expressive than just simple correspondences; it is a first-class model consisting of elements and relationships. Some of its elements are *mapping elements*. A mapping element is like any other element except it also is the origin of one or more *mapping relationships*, M(x, y), each of which specifies that the origin element, x, *represents* the destination element, y. So a given mapping element, x, represents all elements y such that M(x, y). All elements of Map_{AB} in Figure 1 are mapping elements. In Map_{AB} in Figure 2 AllBios is not a mapping element.

There are two kinds of mapping elements: equality and similarity. An *equality mapping element* x asserts that for all y1, y2 ∈ Y such that M(x, y1) and M(x, y2), y1=y2. All elements represented by the same equality mapping element are said to *correspond* to one another. A *similarity mapping element* x asserts that the set of all y1, y2 ∈ Y such that M(x, y1) and M(x, y2) are related through a complex expression that is not interpreted by Merge. This expression is the value of x's Expression

property, which is a property of all similarity mapping elements. Each mapping element also has a property HowRelated, with value "Equality" or "Similarity," to distinguish the two kinds of mapping elements.

Given this rich mapping structure, complex relationships can be defined between elements in A and B, not just simple correspondences. For example, the mapping in Figure 2 (which is between the same models in Figure 1) shows that the FirstName and LastName of model B should be sub-elements of the ActorName element of model A; this is expressed by element m_4, which represents ActorName in A and contains elements m_5 and m_6 which represent FirstName and LastName respectively.

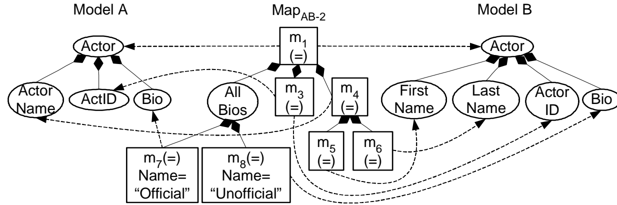


Figure 2: A more complicated mapping

A mapping can also contain non-mapping elements that do not represent elements in either A or B but help describe how elements in A and B are related, such as AllBios in Figure 2. The mapping Map_{AB} in Figure 2 indicates that A.Bio should be renamed "Official," B.Bio should be renamed "Unofficial," and both are contained in a new element, AllBios, that appears only in Map_{AB}.

A mapping can express *similarity* between elements in A and B. For example, if A.Bio is a French translation of B.Bio and this needs to be reflected explicitly in the merged model, they could be connected by a similarity mapping element with an Expression property "A.Bio = English2French(B.Bio)" not shown in Figure 2.

Prior algorithms, whose mappings are not first-class models, cannot express these relationships. Often they require user intervention during Merge to incorporate relationships that are more complicated than simply equating two elements. Merge can encode simple correspondences in a mapping, so it can function even if a first-class mapping is unavailable.

2.3 Merge Semantics

The output of Merge is a model that retains all non-duplicated information in A, B, and Map_{AB}; it collapses information that Map_{AB} declares redundant. If we consider the mapping to be a third model, this definition corresponds to the least-upper-bound defined in BDK [6], "a schema that presents all the information of the schemas being merged, but no additional information." We require Merge to be generic in the sense that it does not require its inputs or outputs to adhere to any given meta-model. We consider another merge definition in Section 6.

We now define the semantics of Merge more precisely. The function "Merge(A, Map_{AB}, B) → G" merges two models A and B based on a mapping Map_{AB}, which describes how A and B are related. The function produces a new model G that satisfies the following Generic Merge Requirements (GMRs):

1. **Element preservation:** Each element in the input has a corresponding element in G. Formally: each element $e \in A \cup B \cup Map_{AB}$ corresponds to exactly one element $e' \in G$. We define this correspondence as $\chi(e, e')$.
2. **Equality preservation:** Input elements are mapped to the same element in G if and only if they are equal in the mapping, where equality in the mapping is transitive. Formally: two elements $s, t \in A \cup B$ are said to be *equal* in Map_{AB} if there is an element $v \in A \cup B$ and an equality mapping element x such that M(x, s) and M(x, v), where either v = t or v is equal to t in Map_{AB}. If two elements $s, t \in A \cup B$ are equal in Map_{AB}, then there exists a unique element $e \in G$ such that $\chi(s, e)$ and $\chi(t, e)$. If s and t are not equal in Map_{AB}, then there is no such e, so s and t correspond to different elements in G.
3. **Relationship preservation:** Each input relationship is explicitly in or implied by G. Formally: for each relationship $R(s, t) \in A \cup B \cup Map_{AB}$ where $s, t \in A \cup B \cup Map_{AB}$ and R is not a mapping relationship M(s, t) with $s \in Map_{AB}$, if $\chi(s, s')$ and $\chi(t, t')$, then either $s' = t'$, $R(s', t') \in G$, or $R(s', t')$ is implied in G.
4. **Similarity preservation:** Elements that are declared to be similar (but not equal) to one another in Map_{AB} retain their separate identity in G and are related to each other by some relationship. More formally, for each pair of elements $s, t \in A \cup B$, where s and t are connected to a similarity mapping element, x, in Map_{AB} and s and t are not equal, there exist elements $e, s', t' \in G$ and a meta-model specific non-mapping relationship R such that $\chi(s, s')$, $\chi(t, t')$, $R(e, s')$, $R(e, t')$, $\chi(x, e)$, and e includes an expression relating s and t.
5. **Meta-meta-model constraint satisfaction:** G satisfies all constraints of the meta-meta-model. G may include elements and relationships in addition to those specified above that help it satisfy these constraints. Note that we do not require G to conform to any meta-model.
6. **Extraneous item prohibition:** Other than the elements and relationships specified above, no additional elements or relationships exist in G.
7. **Property preservation:** For each element $e \in G$, e has property p if and only if $\exists\, t \in A \cup B \cup Map_{AB}$ s.t. $\chi(t, e)$ and t has property p.
8. **Value preference:** The value, v, of a property p, for an element e is denoted p(e) = v. For each $e \in G$, p(e) is chosen from mapping elements corresponding to e if possible, else from the preferred model if possible, else from any element that corresponds to e. More formally:
   - $T = \{t \mid \chi(t, e)\}$
   - $J = \{j \in (T \cap Map_{AB}) \mid p(j) \text{ is defined}\}$
   - $K = \{k \in (T \cap \text{the preferred model}) \mid p(k) \text{ is defined}\}$
   - $N = \{n \in T \mid p(n) \text{ is defined}\}$
     - If $J \neq \emptyset$, then p(e) = p(j) for some $j \in J$

- Else if $K \neq \emptyset$, then $p(e) = p(k)$ for some $k \in K$
- Else $p(e) = p(n)$ for some $n \in N$

GMR 8 illustrates our overall conflict resolution strategy: give preference first to the option specified in the mapping (i.e., the explicit user input), then to the preferred model, else choose a value from one of the input elements. The ID, History, and HowRelated properties are determined differently as discussed in Section 5.

For example, the result of merging the models in Figure 2 is shown in Figure 3. Note that the relationships Actor-FirstName and Actor-LastName in model B and the Actor-Bio relationships in both models are implied by transitivity in Figure 3, so GMR 3 is satisfied.

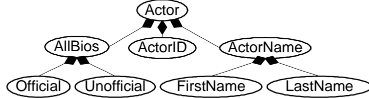


Figure 3: The result of performing the merge in Figure 2

The GMRs are not always satisfiable. For example, if there are constraints on the cardinality of relationships that are incident to an element, then there may be no way to preserve all relationships. Depending on the relationships and meta-meta-model constraints, there may be an automatic resolution, manual resolution or no possible resolution that adheres to the GMRs. In Section 4 we present conflict resolutions for a set of common constraints and discuss when such resolution can be automatic. We also specify default resolution strategies for each category of constraint and note when resolution can be made to adhere to the GMRs outlined above.

3 Conflict Resolution

Determining the merged model requires resolving conflicts in the input. We categorize conflicts based on the meta-level at which they occur:

- **Representation conflicts** (Section 3.1) are caused by conflicting representations of the same real world concept – a conflict at the *model level*. Resolving these conflicts requires manual user intervention. Such conflict resolution is necessary for many uses of mappings – not just Merge. Hence we isolate it from Merge by requiring it to be captured in the input mapping.
- **Meta-model conflicts** (Section 3.2) are caused by the constraints in the *meta-model* (e.g., SQL DDL). Enforcing such constraints is inherently non-generic, so we resolve them using a separate operator after Merge.
- **Fundamental conflicts** (Section 3.3) are caused by violations of constraints in the meta-meta-model. Unlike representation conflicts, fundamental conflicts must be resolved by Merge since subsequent operators count on the fact that the Merge result is a well-formed model.

3.1 Representation Conflicts

A representation conflict arises when two models describe the same concept in different ways. For example, in Figure 1 model A represents Name by one element, ActorName, while model B represents it by two elements, FirstName and LastName. After merging the two models, should Name be represented by one, two or three elements? The decision is application dependent.

Merge resolves representation conflicts using the input mapping. Having a mapping that is a model allows us to specify that elements in models A and B are either:

- The same, by connecting them to the same equality mapping element. Merge can collapse these elements into one element that includes all relationships incident to the elements in the conflicting representations.
- Related by relationships and elements in our meta-meta-model. E.g., we can model FirstName and LastName in B as sub-elements of ActorName in A by the mapping shown in Figure 2.
- Related in some more complex fashion that we cannot represent using our meta-meta-model's relationship kinds. E.g., we can represent that ActorName equals the concatenation of FirstName and LastName by a similarity mapping element that has mapping relationships incident to all three and an Expression property describing the concatenation. Resolution can be done by a later operator that understands the semantics of Expression.

The mapping can also specify property values. For example, in Figure 2 Map_{AB} specifies that one of the elements contained by AllBios is named Official and the other is named Unofficial.

Solving representation conflicts has been a focus of the ontology merging literature [14, 15] and of database schema merging [2, 19].

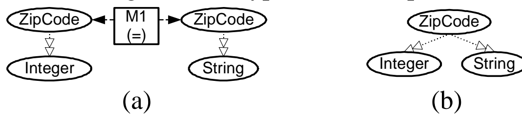
3.2 Meta-model Conflicts

A meta-model conflict occurs when the merge result violates a meta-model-specific (e.g., SQL DDL) constraint. For example, suppose that in Figure 2 Actor is a SQL table in model A, an XML database in model B, and a SQL table in the merged model. If the mapping in Figure 2 is used, there will be a meta-model conflict in the merge result because SQL DDL has no concept of sub-column. This does not violate any principle about the *generic* merged outcome. Rather, it is meta-model-specific. Traditionally, merge results are required to conform to a given meta-model during the merge. However, since Merge is meta-model independent, we do not resolve this category of conflict in Merge. Instead, we break out coercion as a separate step, so that Merge remains generic and the coercion step can be used independently of Merge. We therefore introduce an operator, EnforceContraints, that coerces a model to obey a set of constraints. This operator is necessarily meta-model specific. However, it may be possible to implement it in a generic way, driven by a declarative specification of each meta-model's constraints. EnforceContraints would enforce other constraints, such as integrity constraints, as well. We leave this as future work.

3.3 Fundamental Conflicts

The third and final category of conflict is called a fundamental conflict. It occurs above the meta-model level at the meta-meta-model level, the representation that all models must adhere to. A fundamental conflict occurs when the result of Merge would not be a model due to violations of the meta-meta-model. This is unacceptable because later operators would be unable to manipulate it.

One possible meta-meta-model constraint is that an element has at most one type. We call this the *one-type restriction*. Given this constraint, an element with two types manifests a fundamental conflict. For example in the model fragments in Figure 4(a) ZipCode has two types: Integer and String. In the merge result in Figure 4(b), the two ZipCode elements are collapsed into one element. But the type elements remain separate, so ZipCode is the origin of two type relationships.



(a) (b)

Figure 4: A merge that violates the one-type restriction

Since Merge must return a well-formed instance of the meta-meta-model, it must resolve fundamental conflicts. Resolution rules for some fundamental conflicts have been proposed, such as [6] for the one-type restriction. We have identified other kinds of fundamental conflicts and resolution rules for them which we describe in Section 4 and incorporate into our generic Merge.

The choice of meta-meta-model, particularly the constraints on the relationships, is therefore integrally related to Merge. However, since we are skeptical that there is a meta-meta-model capable of solving all meta data management problems, we chose the following approach: We define the properties of Merge using very few assumptions about the meta-meta-model — only that it consists of elements and relationships. We then define fundamental conflict resolution for a meta-meta-model that includes many of the popular semantic modeling constructs. Finally we describe other typical meta-meta-model conflicts and provide conflict resolution strategies for them.

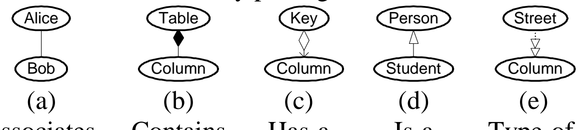
4 Resolving Fundamental Conflicts

The meta-meta-models we consider are refinements of the one described in Section 2.1. Section 4.1 describes *Vanilla*, an extended entity-relationship-style meta-meta-model that includes many popular semantic modeling constructs. Section 4.2 describes our merging strategy, both for Vanilla and for relationship constraints that may be used in other meta-meta-models.

4.1 The Vanilla Meta-Meta-Model

Elements are first class objects with semi-structured properties. Name, ID, and History are the only required properties. Note that these are properties of the element viewed as an instance, not as a template for instances. For example, suppose an element e represents a class definition, such as Person. Viewing e as an instance, it has a Name property whose value is "Person," and might have properties CreatedBy, LastModifiedBy, Comments, and IsInstantiable. To enable instances of Person to have a property called Name (thereby viewing e as a template for an instance), we create a relationship from e to another element, a, where Name(a) = "Name."

Relationships are binary, directed, kinded, and have an optional cardinality constraint. They are also ordered, as in XML, but the order can be ignored in meta-models that do not use it. A *relationship kind* is one of "Associates", "Contains", "Has-a", "Is-a", and "Type-of" (described below). Reflexive relationships are disallowed. Between any two elements we allow at most one relationship of a given kind and cardinality pairing.



(a) Associates (b) Contains (c) Has-a (d) Is-a (e) Type-of

Figure 5: Different relationship kinds in Vanilla

There are cases where the previous restriction is inconvenient. For example, one might want two distinct Has-a relationships between "Movie" and "Person", namely "director" and "actor". This can be handled either by specializing Person into two sub-elements, or by reifying the director and actor Has-a relationships (i.e., turn the relationships into objects), which is the choice used in Vanilla. We disallow multiple named relationships of the same cardinality and kind between two elements because it leads to a need for correspondences between named relationships of different models. E.g., if the director and actor relationships are called "réalisatuer" and "acteur" in another model, we need a relationship between director and réalisatuer and between actor and acteur. These correspondences between relationships would complicate the meta-meta-model. Reifying relationships retains the same expressiveness while avoiding this complexity. Merge does not need to treat these reified relationships specially; since they are ordinary elements that Merge will preserve, just like relationships (see GMRs 1 and 3).

A relationship R(x, y) between elements x and y may be a mapping relationship, M(x, y), described earlier, or one of the following:

- Associates - A(x, y) means x is Associated with y. This is the weakest relationship that can be expressed. It has no constraints or special semantics. Figure 5(a) says that Alice is Associated with Bob.
- Contains - C(x, y) means *container* x Contains *containee* y. Intuitively, a containee cannot exist on its own; it is a part of its container element. Operationally, this means that if all of the containers of an element, y, are deleted, then y must be deleted. Contains is a transitive relationship and must be acyclic. If C(x, y)

and x is in a model M, then y is in M as well. Figure 5(b) says that Table Contains Column.

- Has-a - H(x, y) means x Has-a sub-component y (sometimes called "weak aggregation"). Has-a is weaker than Contains in that it does not propagate delete and can be cyclic. Figure 5(c) says that Key Has-a Column.
- Is-a - I(x, y) means x Is-a specialization of y. Like Contains, Is-a is transitive, acyclic, and implies model membership. Figure 5(d) says that Student Is-a Person.
- Type-of - T(x, y) means x is of type y. Each element can be the origin of at most one Type-of relationship (the one-type restriction described in Section 3.3). Figure 5(e) says that the Type-of Street is Column.

Vanilla has the following *cross-kind-relationship implications* that imply relationships based on explicit ones:

- If T(q, r) and I(r, s) then T(q, s)
- If I(p, q) and H(q, r) then H(p, r)
- If I(p, q) and C(q, r) then C(p, r)
- If C(p, q) and I(q, r) then C(p, r)
- If H(p, q) and I(q, r) then H(p, r)

A model L is a triple (E_L, $Root_L$, Re_L) where E_L is the set of elements in L, $Root_L \in E_L$ is the root of L, and Re_L is the set of relationships in L. Given a set of elements E and set of relationships Re (which may include mapping relationships), membership in L is determined by applying the following rules to $Root_L \in E$, adding existing model elements and relationships until a fixpoint is reached (i.e., until applying each rule results in no new relationships):

- I(x, y), $x \in E_L \rightarrow y \in E_L$; if an element x is in the model, then its generalization y is in the model
- C(x, y), $x \in E_L \rightarrow y \in E_L$; if a container x is in the model, then its containee y is in the model
- T(x, y), $x \in E_L \rightarrow y \in E_L$; if an element x is in the model, then its type y is in the model
- R(x, y), $x \in E_L$, $y \in E_L \rightarrow$ R(x, y) $\in Re_L$
- M(x, y), $x \in E_L \rightarrow$ M(x, y) $\in Re_L$

Since a mapping is a model, its elements must be connected by relationships indicating model membership (Contains, Is-a, or Type-of). However, since these relationships obfuscate the mapping, we often omit them from figures when they do not affect Merge's behavior.

In what follows, when we say relationships are "implied", we mean "implied by transitivity and cross-kind-relationship implication."

We define two models to be *equivalent* if they are identical after all implied relationships are added to each of them until a fixpoint is reached (i.e., applying each rule results in no new relationships). A *minimal covering* of a model is an equivalent model that has no edge that is implied by the union of the others. A model can have more than one minimal covering. To ensure that the merge result G is a model, we require that $Root_{MapAB}$ is an equality mapping element with M($Root_{MapAB}$, $Root_A$) and M($Root_{MapAB}$, $Root_B$), and that $Root_{MapAB}$ is the origin of no other mapping relationships.

4.2 Meta-Meta-Model Relationship Characteristics and Conflict Resolution

This section explores resolution of fundamental conflicts in Merge with respect to both Vanilla and other meta-meta-models: what features lead to an automatic Merge, when manual intervention is required, and default resolutions. The resolution strategies proposed here are incorporated in the Merge algorithm in Section 5. Since the default resolution may be inadequate due to application-specific requirements, Merge allows the user to either (1) specify an alternative function to apply for each conflict resolution category or (2) resolve the conflict manually.

Vanilla has only two fundamental constraints (i.e., that can lead to fundamental conflicts): (1), the Is-a and Contains relationships must be acyclic and (2) the one-type restriction. These fundamental conflicts can be resolved fully automatically in Vanilla.

4.2.1 Relationship-Element Cardinality Constraints

Many meta-meta-models restrict some kinds of relationships to a maximum or minimum number of occurrences incident to a given element. For example, the one-type restriction says that no element can be the origin of more than one Type-of relationship. Such restrictions can specify minima and/or maxima on origins or destinations of a relationship of a given kind.

Cardinality Constraints in Vanilla - Merge resolves one-type conflicts using a customization of the BDK algorithm [6] for Vanilla, a discussion of which can be found in the full version of our paper [16]. Recall Figure 4 where the merged ZipCode element is of both Integer and String types. The BDK resolution creates a new type that inherits from both Integer and String and replaces the two Type-of relationships from ZipCode by one Type-of relationship to the new type, as shown in Figure 6. Note that both of the original relationships (ZipCode is of type Integer and String) are implied.

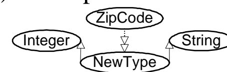


Figure 6: Resolving the one-type conflict of Figure 4

This creates a new element, NewType in Figure 6, whose Name, ID, and History properties must be determined. The ID property is assigned an unused ID value, and Name is set to be the names of the elements it inherits from, delineated by a slash; e.g., NewType in Figure 6 is named "Integer/String." The History property records why the element came into existence, in this case, that Merge created it from the elements Integer and String. As with any other conflict resolution, this behavior can be overridden.

This approach to resolving one-type conflicts is an example of a more general approach, which is the one we use as a default: to resolve a conflict, alter explicit relationships so that they are still implied and the GMRs are still satisfied. Thus, the more implication rules in the meta-meta-model, the easier conflict resolution is.

Requiring that G, the output of Merge, is a model is a form of a minimum element-relationship cardinality; by Vanilla's definition, a model G *satisfies model membership* if all elements of G are reachable from G's root by following containment relationships: Is-a, Contains, and Type-of. Hence, each element must be the origin or destination of at least one such relationship (depending on the relationship containment semantics). Ignoring conflict resolution, we know that G adheres to this constraint:

1. $\chi(Root_A, Root_G)$, $\chi(Root_B, Root_G)$, $\chi(Root_{MapAB}, Root_G)$ from the input and GMR 2 (Equality preservation).
2. $Root_G$ is not the destination of any relationships (and hence is a candidate to be root) because of GMR 6 (Extraneous item prohibition) and because it only corresponds to $Root_A$, $Root_B$, and $Root_{MapAB}$ which likewise are roots.
3. Each element $g \in G$ can be determined to be a member of the model with root $Root_G$: Each element e such that $\chi(e, g)$ must be a member of A, B, or Map_{AB}. Assume without loss of generality that $e \in A$. Then there must be a path P of elements and relationships from $Root_A$ to e that determines that e is in A. By GMR 1 (Element preservation) and GMR 3 (Relationship preservation), a corresponding path P′ must exist in G, and hence g is a member of the model with root $Root_G$.

Hence, conflict resolution notwithstanding, G is guaranteed to satisfy model membership. After conflict resolution for Vanilla, G still satisfies model membership; the BDK solution to the one-type restriction only adds relationships and elements that adhere to model containment. As shown in Section 4.2.2, the acyclic resolution only collapses a cycle, which cannot disturb the model membership of the remaining element.

Cardinality Constraints in General - There are two kinds of relationship-element cardinality constraints: for some *n*: (1) at least *n* relationships of a given kind must exist (*minimality constraints*) and (2) at most *n* relationships of a given kind may exist (*maximality constraints*).

Since Merge (excluding conflict resolution) preserves all relationships specified in the input, the merged model is guaranteed to preserve minimality constraints. For example, one potential minimality constraint is that each element must be the origin of one Type-of relationship. If this were the case, then each of the input models, A, B, and Map_{AB} would have to obey the constraint. Hence each element in A, B, and Map_{AB} would be the origin of at least one Type-of relationship. Since Merge preserves the relationships incident to each element, each element in G is also the origin of at least one Type-of relationship. Conflict resolution may break this property, so conflict resolution strategies must consider these kinds of constraints.

More care is required for a maximality constraint, such as the one-type restriction. If it occurs in a meta-meta-model, the generic merge attempts resolution by removing redundant relationships. Next, the default Merge resolution will look for a cross-kind implication rule that can resolve the conflict (i.e., apply the default resolution strategy). If no such rule exists, then we know of no way to resolve the conflict while still adhering to the GMRs. To continue using the one-type restriction as an example, first we calculate a minimal covering of the merged model and see if it still has a one-type restriction conflict. If so, then we apply a cross-kind implication rule (if T(q, r) and I(r, s) then T(q, s)) which allows us to resolve the conflict and still adhere to the GMRs.

4.2.2 Acyclicity

Many meta-meta-models require some relationship kinds to be acyclic. In Vanilla, Is-a and Contains must be acyclic. In this section, we consider acyclic constraints first in Vanilla and then in general.

Acyclicity in Vanilla - Merging the example in Figure 7 (a) would result in Figure 7 (b) which has a cycle between elements a and b. Since Is-a is transitive, a cycle of Is-a relationships implies equality of all of the elements in the cycle. Thus Merge's default solution is to collapse the cycle into a single element. As with all conflicts, users can override with a function or manual resolution. To satisfy GMR 7 (Property preservation), the resulting merged element contains the union of all properties from the combined elements. GMR 8 (Value preference) dictates the value of the merged element's properties.

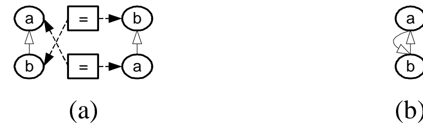


Figure 7: Merging the models in (a) causes the cycle in (b)

Acyclicity Constraints in General - If the constrained relationship kind is not transitive, collapsing the cycle would not retain the desired semantics in general. The default resolution is to see if any cross-kind-relationship implications allow all relationships to exist implicitly without violating the acyclicity constraint. If so, the conflict can be resolved automatically. Without such a relationship implication it is impossible to merge the two models while retaining all of the relationships; either some default resolution strategy must be applied that does not retain all relationships, or human intervention is required.

4.2.3 Other Relationship Conflicts

The following are conflicts that may occur in meta-meta-models other than Vanilla:

- Certain relationships kinds many not be allowed to span meta-levels or Isa-levels. For example, an Is-a hierarchy may not cross meta-levels, or a Type-of relationship may not cross Is-a levels.
- If a meta-meta-model allows only one relationship of a given kind between a pair of elements, the cardinality of the relationship must be resolved if there is a conflict. For example, in Figure 8 what should be the cardinality of the Contains relationship between Actor and ActID? 1:n? m:1? m:n? One could argue that it

should be m:n because this is the most general, however this may not be the desired semantics. Any resolution of this conflict is going to lose information and therefore will not adhere to GMR 3 (Relationship preservation), so no generic resolution can adhere to the GMRs.

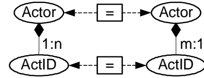


Figure 8: Merging multiple cardinalities

- If only one set of specializations of an element may be declared disjoint, merging two orthogonal such sets requires conflict resolution, e.g., if actors are specialized as living/dead in one model and male/female in another.

5 The Merge Algorithm

This section describes an algorithm for Merge that satisfies the GMRs; an implementation of this algorithm is discussed in Section 7.1.

1. **Initialize** the merge result G to ∅.
2. **Elements:** Induce an equivalence relation by grouping the elements of A, B, and Map_{AB}. Initially each element is in its own group. Then:
   a. If a relationship M(d, e) exists between an element e ∈ (A ∪ B) and a mapping equality element d ∈ Map_{AB}, then combine the groups containing d and e.
   b. After iterating (a) to a fixpoint, create a new element in G for each group.
3. **Element Properties:** Let e be a merged element in G corresponding to a group I. The value v of property p of e, p(e) = v, is defined as follows:
   a. Excluding the property HowRelated, the properties of e are the union of the properties of the elements of I. Merge determines the values of properties of e other than History, ID, and HowRelated as follows:
      J = {j ∈ (I ∩ Map_{AB}) | p(j) is defined}
      K = {k ∈ (I ∩ the preferred model) | p(k) is defined}
      N = {n ∈ I | p(n) is defined}
      i. If J ≠ ∅, then p(e) = p(j) for some j ∈ J
      ii. Else if K ≠ ∅, then p(e) = p(k) for some k ∈ K
      iii. Else p(e) = p(n) for some n ∈ N

   By definition of N, some value for each property of e must exist. In (i) – (iii) if more than one value is possible, then one is chosen arbitrarily.
   b. Property ID(e) is set to an unused ID value. Property History(e) describes the last action on e. It contains the operator used (in this case, Merge) and the ID of each element in I. This implicitly connects the Merge result to the input models and mapping without the existence of an explicit mapping between them.
   c. Element e is a mapping element if and only if some element in I is in (A ∪ B) and is a mapping element (i.e., A and/or B is a mapping). Hence, HowRelated(e) is defined only if e is a mapping element; its value is determined by GMR 8 (Value preference). This is the only exception to GMR 7 (Property preservation).
4. **Relationships:**
   For every two elements e′ and f′ in G that correspond to distinct groups E and F, where E and F do not contain similarity elements, if there exists e ∈ E and f ∈ F such that R(e, f) is of kind t and has cardinality c, then create a (single) relationship R(e′, f′) of kind t and cardinality c. Reflexive mapping relationships (i.e., mapping relationships between elements that have been collapsed) are excluded since they no longer serve a purpose. For example, without this exclusion, after the Merge in Figure 2 is performed, the mapping relationship between elements ActorName and m_4 would be represented by a reflexive mapping relationship with both relationship ends on ActorName. However, this relationship is redundant, so we eliminate it from G.
   a. If element e in G corresponds to a similarity mapping element m in Map_{AB}, replace each mapping relationship, M, whose origin is m by a Has-a relationship whose origin is e and whose destination is the element of G that corresponds to M's destination's group. For example, if the two Bio elements in Figure 1 were connected by a similarity mapping element instead of an equality element, the result would be as in Figure 9.
   b. Relationships originating from an element are ordered as follows:
      - First those corresponding to relationships in Map_{AB}.
      - Then those corresponding to relationships in the preferred model but not in Map_{AB}.
      - Then all other relationships.

      Within each of the above categories, relationships appear in the order they appear in the input.
   c. Finally, Merge removes implied relationships from G until a minimal covering remains.

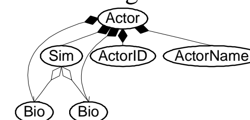


Figure 9: Results of the Merge in Figure 1 if the Bio elements were connected by a similarity mapping element

5. **Fundamental conflict resolution:** After steps (1) – (4) above, G is a duplicate-free union of A, B, and Map_{AB}, but it may have fundamental conflicts (i.e., may not satisfy meta-meta-model constraints). For each fundamental conflict, if a special resolution strategy has been defined, then apply it. If not, apply the default resolution strategy described in Section 4.2.

Resolving one conflict may interfere with another, or even create another. This does not occur in Vanilla; resolving a one-type conflict does create two Is-a relationships, but they cannot be cyclic since their origin is new and thus cannot be the destination of another Is-a relationship. However, if interference between conflict resolution steps is a concern in another meta-meta-model, then Merge can create a priority scheme based on an ordered list of conflict resolutions. The conflict resolutions are then

applied until fixpoint. Since resolving one-type conflicts cannot create cycles in Vanilla, conflict resolution in Vanilla is guaranteed to terminate. However, conflict resolution rules in other meta-meta-models must be examined to avoid infinite loops.

The algorithm described above adheres to the GMRs in Section 2.3. We can see this as follows:

- Step 1 (Initialization) initializes G to the empty set.
- Step 2 (Elements) enforces GMR 1 (Element preservation). It also enforces the first direction of GMR 2 (Equality preservation); elements equated by Map_{AB} are equated in G. No other work is performed in step 2.
- Step 3 (Element properties) performs exactly the work in GMR 7 (Property preservation) and GMR 8 (Value preference) with the exceptions of the refinements in steps 3b and 3c for the ID, History, and HowRelated properties. No other work is performed in step 3.
- In step 4 (Relationships), step 4a enforces GMR 3 (Relationship preservation) and step 4b enforces that a relationship exists between elements mapped as similar, as required in GMR 4 (Similarity preservation). Step 4d removes only relationships that are considered redundant by the meta-meta-model. Step 4c (relationship ordering) is the only step not explicitly covered by a GMR, and it does not interfere with any other GMRs.
- Step 5 (Fundamental conflict resolution) enforces GMR 5 (Meta-meta-model constraint satisfaction) and performs no other work.

If special resolution strategies in step 5 do nothing to violate any GMR or equate any elements not already equated, GMRs 2 (Equality preservation), 4 (Similarity preservation) and 6 (Extraneous item prohibition) are satisfied, and all GMRs are satisfied. Other than special properties (ID, History, and HowRelated) and the ordering of relationships, no additional work is performed beyond what is needed to satisfy the GMRs.

6 Alternate Merge Definitions

Many alternate merge definitions can be implemented using our Merge operator in combination with other model management operators. In this section we consider three-way merge, a common merging problem that occurs in file versioning and computer supported collaborative work [1]. Given a model and two different modified versions of it, the goal is to merge the modified versions into one model. Other Merge variations can be found in [16].

For example, consider Figure 10 where model O has been modified in two different ways to create both models A and B. Suppose there are mappings between O and A and between O and B based on element name equivalence. Notice that in A, element d has been moved to be a child of element b, and in B the element c has been deleted.

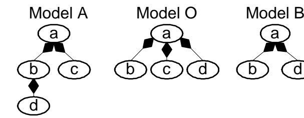


Figure 10: A three-way merge assuming name equality. Model O is the common ancestor of models A and B.

There are several variations of three-way merge which arise due to different treatments of an element modified in one model and deleted or modified in the other. One variation assumes that elements deleted in one model but modified in the other should be included in the merged model. More precisely it assumes that the merged model L should have the following properties:

- If an element e was added in A or B, then e is in L.
- If an element e is present and unmodified in A, B, and O, then e is in L.
- If an element e was deleted in A or B and unmodified or deleted in the other, then e is not in L.
- If an element e was deleted in A or B and modified in the other, then e is in L (because by modifying e the model designer has shown that e is still of interest).
- If an element e was modified in A or B and unmodified in the other, then the modified version of e is in L.
- If an element e was modified in both A and B, then conflict resolution is needed to determine what is in L.

This 3-way merge can be implemented as follows. We determine equality for elements in A and B based on the History property.

1. Create a mapping Map_{AB} between A and B such that:
   a. If $a \in A$ and $b \in B$ are equal, a mapping element expressing equality between a and b is added to Map_{AB}.
   b. If an element e exists in each of O, A, and B, and a property of e has been changed in exactly one of A or B, then Map_{AB} has the changed property value in the mapping element corresponding to e.
2. Create model D such that if an element or relationship has been deleted in one of A or B and is unmodified in the other, it is included in D.
3. G = Merge(A, Map_{AB}, B).
4. Map_{GD} = Match(G, D) – based on History property
5. Return Diff(G, D, Map_{GD}).

Note that this does not handle equating a new element x created independently in both A and B. To allow this, a new mapping could be created to relate A.x and B.x.

Creating the information contained in Map_{AB} and D can be done using a sequence of model management operators. Details are in the full version of our paper [16].

Most algorithms for three-way merge have (1) a "preferred" model that breaks ties and (2) a method for resolving conflicts such as when an element is deleted in one descendent model and modified in the other. We support the former with Merge's preferred model and the latter by applying the model management Apply operator.

7 Evaluation

Our evaluation has two main goals: Section 7.1 shows that Merge can be applied to a real world application where it scales to large models and discovers relevant conflicts and Section 7.2 shows that our Merge definition subsumes previous work.

7.1 Applying Merge to Large Ontologies

We tested Merge on a large bioinformatics application to show that Merge scales to large models and uncovers real conflicts caused by merging such large models. The goal was to merge two models of human anatomy: the Foundational Model of Anatomy (FMA) [18], which is designed to model anatomy in great detail, and the GALEN Common Reference Model [17], which is designed to aid clinical applications. These are very large models; as expressed in a variant of Vanilla, FMA contains 895,307 elements and 2,032,020 relationships, and GALEN contains 155,307 elements and 569,384 relationships. Both of the models were larger in the Vanilla variant than in their "native" format since many of their relationships required reification. The two models have significant structural differences (e.g., some concepts expressed in FMA by three elements are expressed in GALEN by four elements), so merging the two is challenging. Note that there is no additional instance information for either model. Merge was implemented generically in approximately 7,500 non-blank lines of C# with SQL Server as a permanent store.

A database researcher familiar with FMA, GALEN, and model management took 13 weeks to import the models into a variant of Vanilla and create a mapping consisting of 6265 correspondences. The mapping is small relative to the model sizes since the models have different goals and thus different contents. It contains only 1-to-1 correspondences, so we were unable to test our hypothesis that having the mapping as a first class model enables more accurate merging. Hence we concentrated on three other issues: (1) few changes to Vanilla and Merge would be needed to merge the models, even though Merge was not tailored for this domain, (2) Merge would function on models this large, and (3) the merged result would not be simply read from the mapping (i.e., the conflicts that we anticipated would occur).

For the first issue, the researcher needed to add to Vanilla two relationship kinds: Contains-t(x, y), which says that x *can contain* instances of y, and Has-t(x, y), which says that x *can have* instances of y. Neither relationship kind led to new fundamental conflicts Also, the one-type restriction was not relevant to the anatomists. The only change to Merge's default behavior was to list the two new relationship kinds and ignore the one-type restriction.

Merging these models took approximately 20 hours on a Pentium III 866 with 1 GB of RAM. This is an acceptable amount of time since Merge would only be run occasionally in a relatively long project (13 weeks in our case). The merge result before fundamental conflict resolution had 1,045,411 elements and 2,590,969 relationships. 9,096 relationships were duplicates, and 1,339 had origins and destinations that had been equated.

Since the input mapping only uses 1-to-1 correspondences, we would expect most elements in the merged model to correspond to exactly two elements: one in FMA and one in GALEN. However, 2344 merged elements correspond to exactly three elements in FMA and GALEN, and 623 correspond to more than 3 elements. One merged element corresponds to 1215 elements of GALEN and FMA.

The anatomists verified that the specialization hierarchy should be acyclic, as it was in both inputs. However, before conflict resolution the merge result contained 338 cycles in the specialization hierarchy, most of length 2. One was of length 18.

The anatomists agreed that the result of the merge was useful both as a final result, assuming that the input mapping was perfect, and as a tool for determining possible flaws in the input mapping. Exploring the former is a largely manual process and is the subject of ongoing medical informatics research.

7.2 Comparison to Previous Approaches

There has been considerable work on merge in other contexts and applications. An important result of our work is that it subsumes previous literature on merge. In this section we show how Merge, assisted by other model management operators, can implement previous approaches to generic merging (Section 7.2.1), view integration (Section 7.2.2), and ontology merging (Section 7.2.3) even though it is not tailored to their meta-models.

7.2.1 Generic Merging Algorithms

BDK provides the basis for our work: their algorithm creates the duplicate free union of two models based on name equality of the models' elements. Their meta-meta-model contains elements with a name property and two relationship kinds, Is-A and Has-a, where Has-a must obey the one-type restriction.

Essentially Merge encompasses all of the BDK work by taking the duplicate free union of two models and then applying the one-type conflict resolution. Their work considers no other meta-meta-model conflicts, and no other resolutions when their solution to the one-type conflict is inappropriate. In addition, BDK cannot resolve representation conflicts because it lacks an explicit mapping to allow it to do so. Further details of how Merge corresponds to the BDK algorithm can be found in [16].

Rondo [12] is a model management system prototype that includes an alternate Merge definition based entirely on equality mappings. Two elements can be declared to be equal, and each 1-1 mapping relationship can specify a preference for one element over another. Like our Merge and BDK's, Rondo essentially creates the duplicate-free union of the elements and relationships involved. Some

conflicts require removing elements or relationships from the merged model (e.g., if a SQL column is in two tables in a merge result, it must be deleted from one of them). Just as our Merge resolves such meta-model conflicts later, Rondo does such resolutions in a separate operator.

Our Merge is richer than Rondo's in several respects:

1. It can resolve representation conflicts more precisely, since the input mapping structure can relate elements in some fashion other than equivalence.
2. It can resolve conflicts that require the creation of additional elements and relationships rather than pushing the work to a subsequent manual step.
3. By specifying that a choice is first taken from the mapping, then the preferred model, and then any model, it allows for some preferences to be made once per Merge in addition to those made at each mapping element

7.2.2 View Integration

View integration is the problem of combining multiple user views into a unified schema [2]. View integration algorithms (1) ensure the merged model contains all of the objects in the two original models, (2) reconcile representation conflicts in the views (e.g., if a table in one view is matched with a column in another), and (3) require user input to guide the merge.

Spaccapietra and Parent have a well known algorithm [19] that consists of a set of rules and a prescribed order in which to apply them. Their meta-meta-model, ERC+, has three different object types: attributes, entities, and relations. An entity is an object that is of interest on its own. An attribute describes data that is only of interest while the object it characterizes exists. A relation describes how objects in the model interact. ERC+ has three kinds of relationships: Is-a, Has-a, and May-be-a, which means that an object may be of that type.

Vanilla can encode ERC+ by representing attributes, entities and relations as elements. ERC+ Is-a relationships are encoded as Vanilla Is-a relationships. ERC+ Has-a relationships are encoded as Vanilla Contains relationships (the semantics are the same). To encode in Vanilla the May-be-a relationships originating at an element e, we create a new type t such that Type-of(e, t) and for all f such that e May-be-a f, Is-a(f, t).

The Spaccapietra and Parent algorithm for merging models can be implemented using model management by encoding their conflict resolution rules either directly into Merge or in mappings.

Below, we summarize each of their rules and how it is covered by GMRs to merge two ERC+ diagrams A and B to create a new diagram, G. Again we use $\chi(e, e')$ to say that $e \in A \cup B$ corresponds to an element $e' \in G$.

1. Objects integration – If $a \in A$, $b \in B$, $a = b$, and both a and b are not attributes, then add one object g to G such that $\chi(a, g)$ and $\chi(b, g)$. Also, if a and b are of differing types, then g should be an entity. This corresponds to GMR 1 (Element preservation) plus an application of the EnforceConstraints operator to coerce the type of objects of uncertain type into entities.
2. Links integration – If there exist relationships R(p, c) and R(p′, c′), where $p, c \in A$, $p', c' \in B$, $p = p'$, $c = c'$, $\chi(p, g)$, $\chi(p', g)$, $\chi(c, t)$, and $\chi(c', t)$ (i.e., two parent-child pairs are mapped to one another), where neither g nor t are attributes, then R(g, t) is added to G. This is covered by GMR 3 (Relationship preservation).
3. Paths integration rule - Exclude implied relationships from the merged model. This is covered by GMR 3 (Relationship preservation) and Merge algorithm step 4d (Relationships: removing implied relationships). If the user indicates other (non-implied) redundant relationships, they must be either removed outside Merge to avoid violating GMR 3 (Relationship preservation) or expressed by an element representing an integrity constraint in the mapping and hence in the merge result.
4. Integration of attributes of corresponding objects – If there exist relationships R(p, c) and R(p′, c′) where $p, c \in A$, $p', c' \in B$, $p = p'$, $c = c'$, $\chi(p, g)$, $\chi(p', g)$ (i.e., two parent-child pairs are mapped to one another), and c and c′ are attributes, then add an attribute t to G such that $\chi(c, t)$, $\chi(c', t)$ and R(g, t). This is covered by GMRs 2 and 3 (Equality and Relationship preservation).
5. Attributes with path integration – if for some attributes $c \in A$ and $c' \in B$, $c = c'$, there is no relationship R such that R(p, c) and R(p′, c′) where $p = p'$ (i.e., c and c′ have different parents), add an element g to G such that $\chi(c, g)$, $\chi(c', g)$, and add all relationships necessary to attach g to the merged model. If one of the relationship paths is implied and the other is not, add only the non-implied path. This is covered by GMRs 1 and 3 (Element and Relationship preservation).
6. Add objects and links without correspondent – All objects and relationships that do not correspond to anything else are added without a correspondent. This is covered by GMR 1 (Element preservation) and 3 (Relationship preservation).

7.2.3 Ontology Merging

The merging of ontologies is another model merging scenario. A frame-based ontology specifies a domain-specific vocabulary of objects and a set of relationships among them; the objects may have properties and relationships with other objects. The two relationships are Has-a and Is-a. Ontologies include constraints (called *facets*), but they were ignored by all algorithms that we studied. We describe here PROMPT [14], a.k.a. SMART [15], which combines ontology matching and merging.

PROMPT focuses on driving the match, since once the match has been found, their merge is straightforward. As in Merge, their merging and matching begin by including all objects and relationships from both models. As the match proceeds, objects that are matched to one another are collapsed into a single object. Then PROMPT suggests that objects, properties, and relationships that are related to the merged objects may match (e.g., if two

objects each with a "color" property have been merged, it suggests matching those "color" properties).

Our algorithm allows us to provide as much merging support as PROMPT. In the merge of two models, A and B, to create a new model G, PROMPT has the following merge functionality, which we relate to our GMRs. We consider PROMPT's match functionality to be outside Merge's scope.

1. Each set of objects $O \in A \cup B$ whose objects have been matched to each other correspond to one object in G. This is covered by GMR 2 (Equality preservation).
2. Each object $o \in A \cup B$ that has not been matched to some other object corresponds to its own object in G. This is covered by GMR 2 (Equality preservation).
3. An object $g \in G$ consists of all of the properties of the objects in A or B that correspond to it. This is covered by GMR 7 (Property preservation).
4. If a conflict exists on some property's name or value, it is resolved either (1) by the user, corresponding to the user input in Merge's mapping or (2) by choosing from the "preferred" model. This is covered by GMR 8 (Value preference).

Hence, given the input mapping, our algorithm provides a superset of PROMPT's merge functionality.

8 Conclusions and Future Work

In this paper we defined the Merge operator for model merging, both generically and for a specific meta-meta-model, Vanilla. We defined and classified the conflicts that arise in combining two models and described when conflicts from different classes must be resolved. We gave resolution strategies for conflicts that must be resolved in Merge, both for Vanilla and in general. We evaluated Merge by showing how Merge in Vanilla can be used to subsume some previous merging algorithms and by testing Merge on two large real-world ontologies.

We envision several future directions. The first involves showing that the Merge result, when applied to models and mappings that are templates for instances, has an appropriate interpretation on instances. This will demonstrate the usefulness of Merge in specific applications such as data integration and view integration [13, 20].

In some of our experiments we encountered a complex structure in one model that expressed a similar concept to a complex structure in another model, but there was no obvious mapping for the individual elements even though the structures as a whole were similar. An open question is how best to express such similarities and exploit them.

Finally, we would like to see a model-driven implementation of the EnforceConstraints operator that we proposed in Section 3.2.

Acknowledgements

We thank Alon Halevy, Sergey Melnik, Renée Miller, and Erhard Rahm for their continuing collaborations and Peter Mork for the match used in Section 7.1 and many helpful conversations. We thank Michael Ernst, Zack Ives, and Steve Wolfman for their comments on earlier drafts of this paper. This work is partially funded by a Microsoft Research Graduate Fellowship.

References

1. Balasubramaniam, S. and Pierce, B.C., What is a File Synchronizer? MOBICOM, 1998, 98-108.
2. Batini, C., Lenzerini, M. and Navathe, S.B. A Comparative Analysis of Methodologies for Database Schema Integration. Computing Surveys, 18(4). 323-364.
3. Beeri, and Milo, Schemas for Integration and Translation of Structured and Semi-Structured Data. ICDT, 1999, 296-313.
4. Bergamaschi, S., Castano, S. and Vincini, M. Semantic Integration of Semistructured and Structured Data Sources. SIGMOD Record, 28 (1). 54-59.
5. Bernstein, P.A., Applying Model Management to Classical Meta Data Problems. CIDR, 2003, 209-220.
6. Buneman, P., Davidson, S.B. and Kosky, A., Theoretical Aspects of Schema Merging. EDBT, 1992, 152-167.
7. Calvanese, D., Giacomo, Lenzerini, M., Nardi, D. and Rosati. Schema and Data Integration Methodology for DWQ, 1998.
8. Doan, A., Domingos, P. and Halevy, A., Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. SIGMOD, 2001, 509-520.
9. Guarino, N., Semantic Matching: Formal Ontological Distinctions for information Organization, Extraction, and Integration. Summer School on Information Extraction, 1997.
10. Hernández, M., Miller, R.J. and Haas, L.M., Clio: A Semi-Automatic Tool For Schema Mapping. SIGMOD, 2001.
11. Madhavan, J., Bernstein, P.A. and Rahm, E., Generic Schema Matching with Cupid. VLDB, 2001, 49-58.
12. Melnik, S., Rahm, E. and Bernstein, P.A., Rondo: A Programming Platform for Generic Model Management. SIGMOD, 2003, 193-204.
13. Motro, A. Superviews: Virtual Integration of Multiple Databases. Trans. on Soft. Eng., SE-13(7). 785-798.
14. Noy, N.F. and Musen, M.A., PROMPT: Algorithm and Tool for Ontology Merging and Alignment. AAAI, 2000.
15. Noy, N.F. and Musen, M.A., SMART: Automated Support for Ontology Merging and Alignment. Banff Workshop on Knowledge Acquisition, Modeling, and Management, 1999.
16. Pottinger, R.A. and Bernstein, P.A. Merging Models Based on Given Correspondences, U of Washington. Technical Report UW-CSE-03-02-03, 2003.
17. Rector, A., Gangemi, A., Galeazzi, E., Glowinski, A. and Rossi-Mori, A., The GALEN CORE Model Schemata for Anatomy: Towards a re-usable application-independent model of medical concepts. The Twelfth International Congress of the European Federation for Medical Informatics, 1994.
18. Rosse, C. and Mejino, J.L.V. A Reference Ontology for Bioinformatics: the Foundational Model of Anatomy. 2003. Journal of Biomedical Informatics. In press.
19. Spaccapietra, S. and Parent, C. View Integration: A Step Forward in Solving Structural Conflicts. TKDE, 6(2).
20. Ullman, J.D., Information Integration Using Logical Views. ICDT, 1997, 19-40.