Generic Schema Merging

Christoph Quix, David Kensche, and Xiang Li

RWTH Aachen University, Informatik V (Information Systems), 52056 Aachen, Germany
{quix,kensche,lixiang}@i5.informatik.rwth-aachen.de

Abstract. Schema merging is the process of integrating several schemas into a common, unified schema. There have been various approaches to schema merging, focusing on particular modeling languages, or using a lightweight, abstract metamodel. Having a semantically rich representation of models and mappings is particularly important for merging as semantic information is required to resolve the conflicts encountered. Therefore, our approach to schema merging is based on the generic role-based metamodel *GeRoMe* and intensional mappings based on the real world states of model elements. We give a formal definition of the merged schema and present an algorithm implementing these formalizations.

1 Introduction

Management of models is an important activity in the design of complex information systems. The availability of data sources and the need to analyze the existing data in an integrated way, has led to applications which are able to integrate and present data from various sources in a uniform way. The integration of the models of data sources into a unified schema of the integrated information system is a prerequisite to build such applications. *Schema integration* (or *schema merging*) is the process of integrating several schemas into a common, unified schema. This problem is also addressed in *Model Management* [3], which aims at defining an algebra for models and mappings. Merge is one of the proposed operators in this algebra and addresses the problem of generating a merged model given two input models and a mapping between them. The merged model should contain all the information contained in the input models and the mapping; it should *dominate* the inputs in terms of information capacity [8,15].

A mapping is not just a simple set of 1:1 correspondences between model elements; it might have itself a complex structure and is therefore often regarded also as a *mapping model*. A mapping model is necessary because the models to be merged also have complex structures, which usually do not correspond to each other [16] (e.g. the address of a person is represented in one ER model as a complex attribute, in another model as a separate entity type with a relationship type to person). These structural heterogeneities are one class of conflicts which occur in Merge. Other types of conflicts are semantic conflicts (model elements describe overlapping sets of objects), descriptive conflicts (the same elements are described by different sets of properties; this includes also name conflicts), and heterogeneity conflicts (models are described in different modeling languages) [19]. The resolution of these conflicts is the main problem in Merge.

Schema integration has been addressed for various metamodels, such as variants of the ER metamodel [19], relational and conceptual models in the context of data warehouses [5], graph-based models [18], or a simple generic metamodel [16]. In these works, it has also been argued, that a semantically rich representation of the models and mappings simplifies schema integration, as semantic information is required to resolve the conflicts. Our work is therefore based on the semantically rich generic metamodel *GeRoMe* [10]. It provides a generic, but yet detailed representation of models originally represented in different metamodels. Therefore, an implementation of the Merge operator based on *GeRoMe* can merge models from different metamodels (e.g. XML Schema and Relational). It is not always possible to represent the result of Merge in one of the input metamodels. For instance, merging a column with a table yields a model with a composite attribute that is not allowed in the relational model. Therefore, *GeRoMe* enables the representation of such results. The transformation of the merge result into a specific metamodel is the task of other model management operators: ModelGen transforms the modeling constructs which are not allowed in the target model, and Export translates the *GeRoMe* representation into the representation in a native metamodel.

Another important aspect is the representation of mappings between models. As argued before, the mapping is itself a model and contains information required for Merge. In contrast to recent approaches to mapping composition and executable mappings [7,13], which focus on the *extensional* relationships between models for data translation, the mappings here represent the *intensional* relationships of model elements. These mapping definitions often overlap, but are not necessarily identical. This distinction between intensional and extensional relationships has also been made in [5].

In this work, we present a merge algorithm which is based on the intensional relationships between models. The contributions of this paper are (1) a definition of the intensional semantics of models, (2) a mapping model representing intensional relationships, (3) a formal characterization of the desired merged model, and (4) an algorithm implementing these formalizations using *GeRoMe*, thereby enabling the integration of models represented in different modeling languages.

The paper is structured as follows. The next section discusses existing work on model and mapping representation. Section 3 introduces a real world semantics, that we use to define the semantics of merging, and the mapping representation. In section 4 we describe our algorithm for merging *GeRoMe* models given a mapping between them. Section 5 compares our approach to some existing works on schema merging. The last section concludes the paper and gives an outlook.

2 Background

Considerable research has already been done in the fields of model management and data integration. The Merge operator in model management receives two models and a mapping as inputs. Hence, besides existing algorithms, the representation of models and mappings are particularly important prerequisites in the context of schema merging.

Mapping Representations. Depending on the application area, such as data translation, query translation or model merging, schema mappings come in different flavors. The first step in specifying a mapping between two schemas is usually the automatic derivation of informal correspondences between elements of the two respective schemas, called schema matching [17]. These simple binary correspondences, often called *morphisms* [14], state only informally that the respective model elements are similar. Thus, relationships between model elements must be represented more accurately, but morphisms are usually the starting point for specifying more formal mappings.

Formalization of mappings is done in form of view definitions or as correspondence assertions. These will be called in the following *extensional* and *intensional* mappings, respectively. Extensional mappings are defined as local-as-view (LAV), global-as-view (GAV), source-to-target tuple generating dependencies (tgds) [12,13], second order tuple generating dependencies (SO tgds) [7], or similar formalisms. These are pairs of queries with an implication or equivalence operator in between. Each of these classes has certain advantages and disadvantages when it comes to properties such as composability, invertibility or execution of the mappings.

Extensional mappings are used for tasks such as data translation or query rewriting but they are inappropriate for ontology alignment and schema merging. Schema merging is about integrating models according to their intensional semantics. It has the goal to construct a duplicate free union of two input models and a mapping in between. The union is "duplicate free" with respect to the real world concepts described by the model elements. So, the mappings interrelate the *intensions* of model elements. The integrated model describes each real world concept only once. On the other hand, if mappings are to be executed by using them for query rewriting or data translation, intensional mappings are not useful. Thus, these are two options of mapping representation, each of which has certain advantages with respect to the goal of the mapping.

Model Representations. Any system that allows the usage of different native metamodels should employ some generic schema representation. Most model management systems either refrain from providing a generic representation and instead require some operators to be implemented for different combinations of native metamodels or employ very lightweight metamodels [14,16]. However, some model manipulation operations require much more information about the schemas than is expressable in such lightweight languages. Resolving conflicts in model integration is eased by additional information about the schemas to be merged [16,19]. This particular challenge of providing a generic model representation [3] has been adressed in particular in [1,2,10].

The metamodel that we use for our implementation of Merge is the generic role based metamodel *GeRoMe* [10,9]. In *GeRoMe* each model element of a native model (e.g. an XML Schema or a relational schema) is represented as an object that plays a set of roles which decorate it with features and act as interfaces to the model element.

Figure 1 shows in the left part a simple EER model for a "Customers ordering Products" scenario. It contains a relationship type with two attributes and two entity types, the attributes of which we omitted from the example. The right part of the figure presents the equivalent representation in our generic metamodel (we omitted some details of the model for clarity of the presentation). Each entity type, attribute, and relationship type is represented by an individual model element (shown as grey rectangles). Every *GeRoMe* model element plays a number of roles depending on the features of the source element that it represents. The date element plays an *Attribute* role (Att) as it represents an EER attribute, the entity types play ObjectSet roles (OS) since their instances have object identity (as opposed to the instances of a relational table) and *Aggregate* roles (Ag) as they are aggregates of attributes (not shown here). The orders

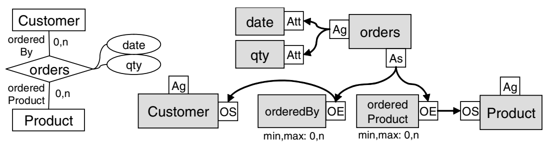

Fig. 1. A small EER model and its representation in *GeRoMe*

element is represented similarly. It plays the role of an *Association* which connects to two *ObjectAssociationEnds* (OE) (orderedBy and orderedProduct). The *ObjectAssociationEnds* specify also the cardinality constraints. Other features of native models can be represented in a similar fashion in *GeRoMe*. The same role classes are used to describe models in different metamodels, thereby providing a common datastructure for the polymorphic implementation of model management operators.

It is important to emphasize that this representation is not to be used by end users. Instead, it is a representation employed internally by model management applications, with the goal to generically provide more information to model management operators than the usual graph based model.

3 Semantics of Models and Mappings

Mappings between the models to be merged are an important source of information for Merge. In order to define the semantics of the mappings in a clear and formal way, we first need to specify the semantics of the models and their elements which are going to be related. As discussed in the previous section, mappings in the form of view definitions are not useful for schema integration, as they usually specify many-to-many (or at least many-to-one) relationships between model elements. Such relationships cannot be used to detect elements in two models which have the same semantics. Relationships must be based on the intended semantics of the model elements rather than extensional relationships. For example, consider the schemas of two universities, each representing the students of that university. The extensions of the databases are disjoint, but the concept “student” should be merged if the schemas are going to be integrated.

Therefore, we need relationships between model elements which are based on their real world semantics, i.e. the set of real world objects a model element represents. Only with respect to this semantics, we can decide whether two elements should be merged. Such an approach based on real world semantics has also been taken in [11,19]. In contrast to these previous approaches, our definition of real world objects is more detailed, i.e. it aims at making this abstract concept more concrete so that it is possible to use it in an implementation of the Merge operator.

In the following, we will first define model elements with respect to their real world semantics, then explain how this representation is related to our generic metamodel *GeRoMe*, and finally define the mapping model which we will use to express intensional relationships between models.

Model Elements. To define the semantics of a model element, we first define the real world objects it should represent.

Definition 1 (Real World Object). *A* real world object (RWO) *is defined as a vector in the feature space with some arity. Each dimension is called an* axis. *The* universe $\mathcal{U}$ *is defined as the set of all RWOs. A* projection *of a real world object o wrt. one axis α, $\pi_\alpha(o)$, is ϵ, a literal or RWO, a set of literals or RWOs, or a tuple of literals or RWOs. A projection wrt. a tuple of axes is a tuple of which each component is the projection wrt. the corresponding axis: $\pi_{\alpha_1,\ldots,\alpha_n}(o) =< \pi_{\alpha_1}(o), \ldots, \pi_{\alpha_n}(o) >$.*

When a projection wrt. one axis is ϵ, this means that the RWO is not defined over this axis. The empty set denotes that the RWO is defined for that axis, but it has no value. For each axis α, we denote all the RWOs over which the axis α is defined as $\mathcal{U}_\alpha$.

Definition 2 (Model Element and Real World Set). *A* model element *m consists of a tuple of axes, denoted by $axes(m) = \{\alpha_1, \ldots, \alpha_n\}$. The* real world set (RWS) *of a model element m is a subset of the RWOs for which all the axes of m are defined: $RWS(m) \subseteq \mathcal{U}_{\alpha_1} \cap \ldots \cap \mathcal{U}_{\alpha_n}$, if $axes(m) = \{\alpha_1, \ldots, \alpha_n\}$.*

Relationship to *GeRoMe*. These definitions characterize the real world semantics of a model element. In [10], we defined also a formal semantics for *GeRoMe* models which characterizes the structure of their instances. In *GeRoMe*, there are four different roles for which the corresponding model elements can have instances. These roles are *Domain*, *ObjectSet*, *Aggregate*, and *Association*. Elements playing a *Domain* role are a special case; these model elements cannot play any of the other roles. Their instances are just sets of literal values. However, *ObjectSet*, *Aggregate*, and *Association* roles can be combined. The instances of model elements playing at least one of these roles are specified by a triple which has the following components: (i) an object identifier, if it plays an *ObjectSet* role, (ii) a tuple of object identifiers, denoting the participating objects in an association (one for each association end), if it plays an *Association* role, and (iii) a tuple of literal value sets (one for each attribute), if it plays an *Aggregate* role.

Thus, a RWO can be easily mapped to a *GeRoMe* instance. The axes having a literal value or a set of literal values correspond to the third component of a *GeRoMe* instance which specifies the attribute values. The axes referring to RWOs are mapped to the second component which expresses associations to other objects.

Based on this relationship between the real world semantics and *GeRoMe*, we can later use the mapping model defined in the following to specify mappings between *GeRoMe* models. The transition from the real world semantics to *GeRoMe* makes this definition useful for the implementation of a model management system.

Mappings. A mapping specifies how the two models will be merged. In [16], the mapping model is a nested structure consisting of mapping elements; each mapping element is related to at least one model element. The mapping elements can specify, that the related model elements are either equivalent or similar. A similarity mapping states that the elements are related by a complex expression which is not further specified.

A richer set of relationships between model elements is defined in [19], which is also based on the real world semantics of model elements. Possible relationships are equivalence, inclusion, intersection and exclusion. However, only equivalence relationships are used in the integration rules. A simple form of nesting of elements can be specified using the "with corresponding attributes" clause for two related model elements.

Our approach is a combination of the ideas of both approaches: a nested mapping model with rich semantic relationships based on the RWS of model elements. In addition, all this information will be used in the Merge algorithm as we will see in section 4. We will first define how model elements can be related at the top-level.

Definition 3 (Element Mapping). *An element mapping ϕ between two model elements m and m' is an expression $m\theta m'$ with $\theta \in \{=, \subseteq, \cap, \neq\}$. The semantics of the mapping is defined by the RWS of the model elements:*

1. *$m\theta m'$ with $\theta \in \{=, \subseteq\}$ implies that $RWS(m)\theta RWS(m')$.*
2. *$m \cap m'$ states that $RWS(m)$ and $RWS(m')$ have a non-empty intersection.*
3. *$m \neq m'$ specifies that $RWS(m)$ and $RWS(m')$ are disjoint, but there is some m'' with $RWS(m) \subseteq RWS(m'')$ and $RWS(m') \subseteq RWS(m'')$.*

The disjointness of elements is useful for a case where two model elements have a common super-type (cf. the example of fig. 4 in sec. 4). In *GeRoMe*, element mappings are only allowed between model elements playing either the *ObjectSet* or the *Aggregate* role, as they have instances. Associations may also have instances, but this information is already covered by the axes representing *AssociationEnds*. In addition to this simple type of mapping, a complex mapping represents 1:N relationships between elements.

Definition 4 (Complex Mapping). *A complex mapping ϕ between a set of model elements is an expression $m\theta f(m_1, \dots, m_n)$ with $\theta \in \{=, \subseteq\}$ for some function f. The semantics of this mapping is defined by applying the corresponding operations to the RWS of the model elements.*

This enables us to represent that a model element in one model is represented by a combination of model elements in the other model, e.g. $Parent = Mother \cup Father$. This also subsumes the definition of *paths* as defined in [19] as functions can specify arbitrary relationships between model elements.

A nested mapping is a mapping between axes of model elements. It must be nested into an element or complex mapping, so that a context for this mapping is given, i.e. we need to know the model elements of which the axes are mapped.

Definition 5 (Nested Mapping). *A nested mapping under a mapping ϕ between model elements m and m' is an expression (i) $\alpha\theta\beta$ with $\theta \in \{=, \subseteq\}$ and $RWS(m) \cup RWS(m') \subseteq \mathcal{U}_\alpha \cap \mathcal{U}_\beta$. The semantics is $\forall o \in RWS(m) \cup RWS(m') : \pi_\alpha(o)\theta\pi_\beta(o)$; (ii) $\alpha = f(\alpha_1, \dots, \alpha_n)$ with $RWS(m) \cup RWS(m') \subseteq \mathcal{U}_\alpha \cap \mathcal{U}_{\alpha_1} \cap \dots \cap \mathcal{U}_{\alpha_n}$. The semantics is $\forall o \in RWS(m) \cup RWS(m') : \pi_\alpha(o) = f(\pi_{\alpha_1}(o), \dots, \pi_{\alpha_n}(o))$.*

The functional relationships between axes are necessary to represent complex relationships between axes (as before between model elements). Examples are amounts represented in different currencies or aggregations (e.g. salary=base salary+bonus).

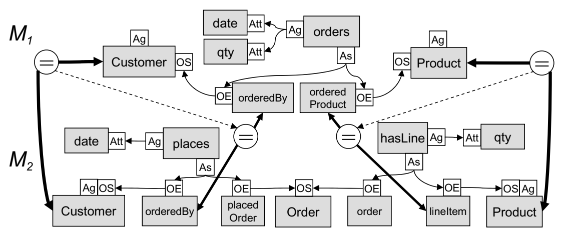

Fig. 2. Mapping including complex and nested mappings

In *GeRoMe*, nested mappings can be applied to model elements playing the *Attribute* or *AssociationEnd* roles. Fig. 2 shows an example of a mapping including a nested mapping. The upper model M_1 is as in section 2, the lower model M_2 reifies the orders relationship as entity type and has therefore two relationship types places and hasLine. The mapping relates the corresponding Customer and Product entity types. The mappings of the *ObjectAssociationEnds* are nested within these mappings. The example contains also a complex mapping (not shown in the figure), as the attributes of the orders relationship are distributed over two relationships in M_2. Therefore, the complex mapping orders $\subseteq f($places, Order, hasLine$)$ needs to be defined in which f performs a join over these elements. In addition, mappings for the attributes date and qty will be nested into this mapping.

As a consequence of this mapping, the Merge algorithm will produce M_2 as result, as it contains "more detailed" information than M_1: the entity types are represented by the corresponding entity types in M_2; the same holds for the *ObjectAssociationEnds*; the orders association is represented by a combination of elements in M_2 and its attributes are also present in M_2. The difficult question in the definition of Merge is, what does it exactly mean when we say that a model contains "more detailed" information than another model, and how can we verify that M_2 is a correct result of the Merge operator in this example. These issues will be addressed in the following section.

4 Model Merging

In this section we first provide a definition of the concept of a merged model using our previously defined real world semantics. Then, we explain our solution to the problem of merging of schemas represented in *GeRoMe*.

Definition of Merged Model. Our definition of schema merging is closest related to that of [19] as it also defines the problem of schema integration with respect to the real world sets described by the input schemas. The difference is that we use our real world sets to define the notion of a merged model. In addition, we relate the real world sets to information capacity [15] and, in doing so, enrich this notion with meaning.

The following definition introduces successively more general concepts of subsumption ending with a definition of when a model element is subsumed by a set of other elements. This means that all the information represented by the model element is also represented by combinations of the properties of the other model elements.

Definition 6 (Subsumption ($\sqsubseteq$)).

a. *Given two axes α and β, we say α is subsumed by β wrt. a RWO o ($\alpha \sqsubseteq_o \beta$) if these two axes are defined over o and $\pi_\alpha(o) \subseteq \pi_\beta(o)$.*
b. *an axis α is subsumed by a set of axes $\mathcal{A} = \{\beta_1, \ldots, \beta_n\}$ wrt. a RWO o ($\alpha \sqsubseteq_o \mathcal{A}$) if the axes $\mathcal{A} \cup \alpha$ are defined over o and $\pi_\alpha(o) \subseteq \pi_{\beta_1}(o) \cup \ldots \cup \pi_{\beta_n}(o)$ or more general $\exists f : \pi_\alpha(o) \subseteq f(\pi_{\beta_1}(o), \ldots, \pi_{\beta_n}(o))$.*
c. *an axis α is subsumed by a set of axes $\mathcal{A}$ wrt. a set of RWOs R ($\alpha \sqsubseteq_R \mathcal{A}$) if $\forall o \in R : \alpha \sqsubseteq_o \mathcal{A}$.*
d. *an axis α of model element m is subsumed by a set of model elements $\mathcal{M}$ ($\alpha \sqsubseteq_{RWS(m)} \mathcal{M}$) if $\forall o \in RWS(m), \exists m_1, \ldots, m_n \in \mathcal{M} : o \in RWS(m_1) \cap \ldots \cap RWS(m_n) \wedge \alpha \sqsubseteq_o axes(m_1) \cup \ldots \cup axes(m_n)$.*
e. *a model element m is subsumed by a set of model elements $\mathcal{M}$ ($m \sqsubseteq \mathcal{M}$) if $\forall \alpha \in axes(m) : \alpha \sqsubseteq_{RWS(m)} \mathcal{M}$.*

By defining $RWS(\mathcal{M}) = \bigcup_{m \in \mathcal{M}} RWS(m)$, the definition is extended to a definition for subsumption of models, which is used to define the *upper bound* of two models.

Definition 7 (Subsumption and Upper Bound Models). *Let $\mathcal{M}$ and $\mathcal{M}'$ be two models. We say $\mathcal{M} \sqsubseteq \mathcal{M}'$ if $RWS(\mathcal{M}) \subseteq RWS(\mathcal{M}') \wedge \forall m \in \mathcal{M} : m \sqsubseteq_{RWS(\mathcal{M})} \mathcal{M}'$. A model $\mathcal{M}_{UB}$ is an* upper bound model *of two models $\mathcal{M}_1$ and $\mathcal{M}_2$ if $\mathcal{M}_1 \sqsubseteq \mathcal{M}_{UB} \wedge \mathcal{M}_2 \sqsubseteq \mathcal{M}_{UB}$.*

An upper bound model is a model that represents the same (or a larger) set of real world objects and that does not lose any properties of the two models. However, this definition allows that subclasses that add no axes are removed from the models. Therefore, we need the following definition of granularity.

Definition 8. *A model element m is* retained in granularity *by a model $\mathcal{M}$ if (i) $\exists m' \in \mathcal{M}$ such that $RWS(m) = RWS(m')$, and (ii) $\forall \alpha \in axes(m)\ \exists m_1, \ldots, m_n \in \mathcal{M} : \alpha \sqsubseteq_{RWS(m)} \{m', m_1, \ldots, m_n\}$.*

The first condition requires that $\mathcal{M}$ must include a model element m' that represents exactly that same set of real world objects that m represents. The second condition states that each axis of each object in the set represented by m, is either explicitly represented in $\mathcal{M}$ or can be computed from some of its axes. This also includes any notion of inheritance between the model elements in $\mathcal{M}$. This is because it is not required that the axes representing α are axes of m' but they may be inherited as well, whereas the object itself must be in m' and all the other model elements.

A model $\mathcal{M}'$ subsumes a model $\mathcal{M}$ while retaining all its model elements in granularity, if it represents the same (or a larger) set of RWOs and of all objects represented by $\mathcal{M}$ it represents the same (or more) information either explicitly or by means of some functional relationship. Also, we do not want a model to be redundant. The last definition before we can define the notion of a *merged model* is that of duplicate-freeness:

Definition 9 (duplicate-free). *A model $\mathcal{M}$ is* duplicate-free *if (i) for each model element $m \in \mathcal{M}$ there is no other model element $m' \in \mathcal{M}$ that represents the same set of real world objects, and (ii) for each axis α of any model element in $\mathcal{M}$ there exists no other axis β that represents the same property of the model element.*

Definition 10 (Merged Model). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two input models and let M be a mapping between $\mathcal{M}_1$ and $\mathcal{M}_2$. A model $\mathcal{G}$ is the result of $Merge < \mathcal{M}_1, \mathcal{M}_2, M >$ (a merged model), if it satisfies the following properties:*

- *$\mathcal{G}$ is an upper bound model of $\mathcal{M}_1$ and $\mathcal{M}_2$.*
- *$\mathcal{G}$ is duplicate-free.*
- *$\mathcal{G}$ retains granularity of all model elements in $\mathcal{M}_1$ and $\mathcal{M}_2$.*
- *$\mathcal{G}$ contains all constraints of the input models and the mapping, and in case of conflicting constraints, the least restrictive constraint.*
- *There is no other model $\mathcal{G}'$ with $\mathcal{G}' \sqsubset \mathcal{G}$, which fulfills these conditions.*

Informally, given two models and a mapping, merging these models means to create a model that contains no duplicate model elements or axes and only structural elements from any of the two input models. However, derivations or constraints may be added to the integrated model in order to relate these model elements to each other. The information necessary for adding such elements stems from the mapping model.

Constraints of the input models and the mapping should be retained in $\mathcal{G}$. If there is a conflict between constraints, the least restrictive constraint is represented in $\mathcal{G}$. For instance, given two cardinality constraints $(0,1)$ and $(1,n)$, $\mathcal{G}$ contains $(0,n)$.

This definition may be relaxed such that the resulting model is allowed to represent axes that can be derived from other represented axes. These are, for instance, derived attributes in an EER model or methods in an object oriented model that compute values from member variables. Such an extension would necessitate to partition the axes of the models into two kinds of axes and define subsumption only with respect to one of these classes. This extension is straightforward and adds only little information.

The definitions of subsumption and merged models can also be related to the notion of relative information capacity of schemas [15].

Definition 11. *An* information capacity preserving mapping *between the instances of two schemas $S1$ and $S2$ is a total, injective function $f : I(S1) \rightarrow I(S2)$. If such a mapping exists $S2$* dominates *$S1$ via f, denoted $S1 \preceq S2$.*

A common criticism about this definition is that it allows the models and the function to be arbitrary, given only that such a function can be constructed. By using our relationship to real world sets, we add some meaning to this definition.

Theorem 1. *If $\mathcal{M} \sqsubseteq \mathcal{M}'$ then $\mathcal{M} \preceq \mathcal{M}'$* and *the domain of discourse of $\mathcal{M}'$ encompasses the domain of discourse of $\mathcal{M}$ ($RWS(\mathcal{M}) \subseteq RWS(\mathcal{M}')$).*

Proof. The last part of the theorem ($RWS(\mathcal{M}) \subseteq RWS(\mathcal{M}')$) follows directly from the definition of subsumption between models. To show that $\mathcal{M}'$ dominates $\mathcal{M}$, we have to construct an information capacity preserving mapping. Instances of the models in our context are sets of RWOs. According to the definition of subsumption, the axes of $\mathcal{M}$ are represented in $\mathcal{M}'$, either directly or they can be computed by some function f. Thus, each RWO in $RWS(\mathcal{M}')$ is represented in more detail than in $RWS(\mathcal{M})$. This means that there are no RWOs $o_1, o_2 \in RWS(\mathcal{M})$, such that both correspond to the same RWO in $RWS(\mathcal{M}')$. As the RWS of $\mathcal{M}'$ is a superset of the RWS of $\mathcal{M}$, the information capacity preserving mapping is the identity function on the RWOs.

Input: Two models $\mathcal{M}_1$, $\mathcal{M}_2$, and a mapping M
Output: Merged model $\mathcal{G}$ according definition 10

1. Group equivalent model elements transitively. For each group, introduce a corresponding model element in $\mathcal{G}$. Each of the new model element is created as follows:
   (a) Singletons are copied with no linking axes to other model elements.
   (b) Collapse groups into one model element with a union of all properties and roles.
   (c) Conflicts are resolved according to the strategy described below.
2. Introduce all the classification relationships (IsA) and constraints (Disjoint) from both the input models and the mapping. Remove redundant or cyclic IsA links.
3. Insert in $\mathcal{G}$ a most specific supertype for model elements connected by $\cap$ or $\neq$.
4. The singleton side of complex mappings are removed from the merged model if all of its axes are related in the nesting mappings.
5. For all axes nested under model elements connected by element mappings
   (a) Equally related axes are collapsed and pulled up to their most specific supertype.
   (b) $\subseteq$ related axes will lead to a partition of the subsumer axes.
   (c) Axes related by functions are handled depending on configuration: leaving only the inputs, the output or both.
   (d) The remaining axes are retained at the corresponding element in $\mathcal{G}$.
6. All links between model elements are handled as follows:
   (a) duplicate links are only introduced once;
   (b) conflicting links such as multiple types for one attribute are resolved according to the strategy described below.
7. Check "local" constraints and resolve possible conflicts as described below.
8. Check "global" conflicts that can not detected locally, e.g. recursion of complex attribute.

Fig. 3. Merge Algorithm

Our definition of a merged model has the following consequences with respect to information capacity preserving mappings: (i) $\mathcal{G} \succeq \mathcal{M}_1$ and $\mathcal{G} \succeq \mathcal{M}_2$ because it retains each element $m \in \mathcal{M}_1 \cup \mathcal{M}_2$ in granularity, and (ii) $RWS(\mathcal{G}) = RWS(\mathcal{M}_1) \cup RWS(\mathcal{M}_2)$. This adds some meaning to the notion of information capacity as now outright absurd functions are not possible. Please note that it is not necessarily the case that $\mathcal{M}_1 \cup \mathcal{M}_2 \succeq \mathcal{G}$ because the mapping may add relationships to the models.

Merge Algorithm. Most previous algorithms are rule-like [11,19] operational procedures or semi-automatic procedures [16]. The first type usually goes through a continuous pattern-transformation procedure. The latter type first collapses all equivalent elements and then introduce links between the grouped elements.

Our merging algorithm consists of several steps described in fig. 3. In the first step, all model elements equally related in the mapping are grouped transitively. Groups of equivalent model elements are collapsed into one element. This might cause conflicts which must be resolved as described below. The second and third steps deal with *IsA*, *Overlapping*, and *Disjoint* relationships and introduce helper elements as necessary.

Step 4 deals with complex mappings. For the orders-example in section 3 the orders element $\mathcal{M}_1$ is removed as it can be represented by combination of other elements. The next step addresses the axes of elements. If we state that $\alpha \subseteq \beta$, then we will keep α and introduce a new element "$\beta - \alpha$" for the remaining part of β (e.g. $father \subseteq parent$ is replaced by $father$ and $mother$). The handling of axes connected by functions depends on the setting; it may be useful to keep the inputs, the output, or both. In step 6, redundant and conflicting links introduced by Merge are removed.

The remaining steps deal with conflicts and constraints; these must be handled by specific procedures as described below. Due to space constraints, we can only sketch them briefly. The general rule is, first to check whether it is solvable by an existing automatic resolution strategy depending on the given configuration. Please note that our merge algorithm can be configured with several parameters which allow to fine tune the algorithm for a given situation. The general context can be specified (database vs. view integration), or the handling of the inputs and outputs of functions. For instance, given two equivalent attributes with a simple type and a complex type, respectively, this is a structural conflict which can be resolved by chosing the more detailed representation (e.g. the complex type). Then, try to resolve the conflict using the mapping or preferred model. If the conflict cannot be resolved with the given information, the user has to be involved. The handling of conflicts is often done on a case-by-case basis; for each specific problem an individual conflict resolution strategy must be found. Often, different strategies have to be applied to different scenarios (e.g. view integration vs. database integration). However, as we based our implementation on *GeRoMe*, these resolution strategies have to be implemented only once for *GeRoMe* and can then be applied to several merge scenarios involving different modeling languages.

Constraints Integration. Several types of constraints are represented in *GeRoMe* explicitly (such as keys, references, derivation, and type constraints) and can therefore be addressed in Merge. Each type of constraint requires individual methods for merging and conflict resolution. Even worse, conflict resolution might depend on the scenario. A non-null constraint on an axis not represented in one input model might be removed if we insist that the integrated schema should host all data instances of the input models for database integration, while in view integration we would better retain it.

Key constraints are handled in Merge in the following way. A key is a set of axes of an element. Assume there are two elements with different keys. As the key of one model element might not be unique within the other model element, we can only introduce a uniqueness constraint over the union of the keys by default. If a new key is required, different strategies are possible: using the union as a new key (if it is the smallest key), introducing an artificial key or asking the user for a key. All foreign keys are rechecked after integration of keys, and the key components are updated to match the new keys. Other constraints such as default values for attributes and sequences (ordered attributes/associations) cannot be handled by a reasonable integration strategy. They can only be merged by preferring one input model, or asking the user.

Conflicts in Merging. In general, the types of conflicts caused by merging schemas are determined by the nature of the target metamodel. For example, we have several options of names for one model element in the merged model. In the relational model, this leads to a name conflict, while in OWL multiple labels for one model element are allowed and thus there is no conflict at all. As we represent our models in *GeRoMe*, such metamodel conflicts have to be addressed by the ModelGen operator, as this operator translates constructs not supported by a target metamodel. Please note that in general the task of the ModelGen operator is to translate a model from one metamodel to another metamodel (e.g. EER to Relational). The use of a generic metamodel allows us to reuse the functionality of this operator in the context of schema merging.

However, there are still conflicts which might also occur in *GeRoMe*, e.g. multiple roles of the same type for one model element, an attribute with more than one type, or recursion in the types of complex attributes. Resolution of conflicts is ad-hoc, there is usually no universal way to solve all of them [16].

We handle conflicts in a multi-level procedure. Firstly, we take an automatic resolution strategy if possible. The information to resolve the conflicts might be given by the parameters of the merge algorithm (e.g. view vs. database integration), the input models and mapping, or the metamodel. For example, if we have two conflicting roles for one model element, we keep the more general one (e.g. if a key reference is in conflict with an association, we keep the association). Secondly, metamodel heterogeneity conflicts are resolved by taking the most flexible construct. For example, conflicts among foreign keys, complex attributes and associations lead to a representation as an association. Thirdly, explicitly encoded choices (e.g. prefer one input model) are taken for mutually excluded properties such as name of the element or default values.

The final fallback is to ask the user for a resolution. It has to be noted that the problem of schema merging will remain an activity which requires human intervention as schema merging is a *design* activity. Some of the conflicts addressed in [16] are solved by having a complex mapping model as input. However, this input needs to be defined by some user. Our current on-going implementation of the mapping editor and merge algorithm integrates the process of mapping definition and merging in an interactive way, i.e. while defining the mapping, the user will be notified about conflicts during merging the schemas.

An Application of the Merge Algorithm. Fig. 4 depicts an example of two models to be merged with our algorithm. The models are the partial *GeRoMe* representations of two models showing courses with the students assisting in these courses. Schema M_1 is the representation of an XML Schema. It shows a Course complex type with a nested element hasAssistant of type CSStudent. The XML Schema element is represented in *GeRoMe* as an *Association* with an anonymous *ObjectAssociationEnd* (OE) and a *CompositionEnd* (CE) due to the fact that in an XML document students must be nested into courses. For the same reason, the association end linking to the CSStudent type has a cardinality constraint of $min = max = 1$; whenever a CSStudent occurs in a document it *must* be nested into a Course via this element link. Schema M_2 is the representation of an object oriented model, e.g. a UML object model. Again, we have two classes Course and GradStudent with the same relationship in between. The CourseStud element represents an association class which has an attribute hours. Also, the association ends in this relationship are both named and do have more relaxed cardinality constraints as the used metamodel is not constrained to define tree structures. Students' names are represented as an attribute with a composite type instead of two simple attributes and the class adds an attribute program.

The mapping between the models equates the Course types, the associations, their association ends, and the the firstname and lastname attributes (not shown in the figure). The GradStudent and CSStudent are declared overlapping, as each graduate student that studies computer science is also a computer science student.

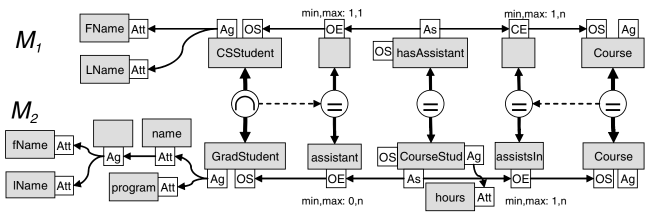

Fig. 4. Merging *GeRoMe* representations of an XML Schema and an OO model

The merging procedure first copies all elements that are not mapped to the merged model. This includes the hours, program, and name attributes and the anonymous type. Then, equally related model elements that are not attributes are collapsed. This involves collapsing the Course types, the associations and the association ends. When collapsing the association ends, conflicts between constraints will be resolved by using the least restrictive constraints. That is, $\mathcal{G}$ will contain two association ends with cardinality constraints (0, n) and (1, n) respectively. The corresponding element in $\mathcal{G}$ to assistsIn will play an *ObjectAssociationEnd* as in M_2 which is less restrictive than the *CompositionEnd* in the XML Schema. During collapsing these elements, unmapped axes of the elements will be linked to their types in $\mathcal{G}$. The hours attribute will be an attribute of the merged association. The same applies to the program attribute and GradStudent. As CSStudent and GradStudent are overlapping, a most specific supertype of the two types will be introduced. According to step 5 of the merge algorithm the new type will have a composite attribute name that results from merging the original attributes.

It must be emphasized that, although one of the input schemas was an XML Schema, the merge result is not an XML Schema. This is because merging has destroyed the nesting structure of the XML element hasAssistant and the result contains an attribute with composite type, which is not allowed in XML Schema. The only requirement in our approach is that the result is a valid model in the generic metamodel *GeRoMe*.

5 Comparison with Other Approaches

Most existing work on schema merging deals with integration of models in one particular metamodel and rarely considers integration across various metamodels. In Rondo, the Merge operator is implemented using morphisms and simple graph representations of models [14]. In [16] a nested mapping model is utilized for merging of simple object oriented models. However, as the generic metamodel is relatively simple, some constraints cannot be described and hence cannot be used by the merging algorithm.

One result of [16] is a list of *generic merge requirements*. Our Merge solution satisfies all but two of these requirements which can be adapted to our mapping representation: *extraneous item prohibition* and *property preservation* demand that no new model elements are added and that a model element in the merged model has a property if and only if one of the corresponding source elements had that property. In [16] there are only two types of mappings allowed, namely similarity and equality. Therefore, the main operations are collapsing of elements declared equal and nesting of similar elements under a helper element that is given in the mapping model. Our mapping model allows more kinds of assertions such as disjointness, overlap, and subset relationships. In the example in the last section a new element *Student* has been added as a common superclass of the original elements *GradStudent* and *CSStudent*, due to the overlap of these elements. In the same example the common properties of these overlapping, (but unequal) elements have also been "pulled up" to the new supertype (representing their union). Because the two requirements demand all elements and properties in the merged model to be given in either of the input models *or* the mapping model, an element such as the superclass and its relationship to the original elements must be defined in the mapping. This amounts to giving the result of merging in the input of merging. In our case it suffices to declare the elements as overlapping to achieve the same result.

Another approach to merging is that of [19] which also presents a comprehensive taxonomy of schema integration conflicts. Like ours, their work is based on the real world sets of model elements. We have used our extended notion of real world sets to define mappings and the merged model formally. Our solution does not only allow a metamodel independent specification of mappings but also the merging algorithm itself is independent of native modeling languages. We provide just one solution for merging models represented in our generic metamodel [10]. The Merge algorithm can therefore be applied polymorphically. Other approaches [4,19] use also generic metamodels, but these are not as detailed as our generic metamodel *GeRoMe*.

While a variety of integration approaches exist in database practices [6,16,19], theoretical aspects of merging are first covered in Buneman et al [4]. The authors introduce the notion of a least upper bound for merging. We extend their work to accommodate more complex mappings using real world semantics and allow configuration options for different scenarios or requirements instead of one single solution.

6 Conclusion and Outlook

By giving formal definitions of models, mappings, and merging based on their intensional semantics and relating this to the notion of information capacity we formalized the term "duplicate-free union" that is usually used informally to describe the merge result. We also gave a Merge algorithm that uses accurate intensional mappings. Strategies for solving conflicts in schema merging are highly case based. This problem is aggravated by the number of metamodels. Our merge solution contributes to solving such heterogeneity conflicts [19] as it is based on the rich generic metamodel *GeRoMe* which makes it possible to apply resolutions polymorphically for different metamodels.

In the future we will develop generic conflict resolution strategies for a representative set of structural conflicts, and we will investigate the question how intensional mappings can be derived from extensional mappings (cf. section 2) and vice versa.

Acknowledgements. The work is supported by the Research Cluster on Ultra High-Speed Mobile Information and Communcation UMIC (`www.umic.rwth-aachen.de`).

References

1. Atzeni, P., Cappellari, P., Bernstein, P.A.: A Multilevel Dictionary for Model Management. In: Proc. Conf. Conceptual Modeling(ER 2005), LNCS, vol. 3716, pp. 160–175. Springer, Heidelberg (2005)
2. Atzeni, P., Torlone, R.: Management of Multiple Models in an Extensible Database Design Tool. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 79–95. Springer, Heidelberg (1996)
3. Bernstein, P.A., Halevy, A.Y., Pottinger, R.: A Vision for Management of Complex Models. SIGMOD Record 29(4), 55–63 (2000)
4. Buneman, P., Davidson, S., Kosky, A.: Theoretical Aspects of Schema Merging. In: Pirotte, A., Delobel, C., Gottlob, G. (eds.) EDBT 1992. LNCS, vol. 580, pp. 152–167. Springer, Heidelberg (1992)
5. Calvanese, D., Giacomo, G.D., Lenzerini, M., Nardi, D., Rosati, R.: Description Logic Framework for Information Integration. Proc. KR, pp. 2–13 (1998)
6. Euzenat, J.: State of the art on ontology alignment. Deliv. D2.2.3, KnowledgeWeb (2004)
7. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: Second-order dependencies to the rescue. ACM Trans. Database Syst. 30(4), 994–1055 (2005)
8. Hull, R.: Relative Information Capacity of Simple Relational Database Schemata. SIAM Journal of Computing 15(3), 856–886 (August 1986)
9. Kensche, D., Quix, C.: Transformation of Models in(to) a Generic Metamodel. Proc. BTW Workshop on Model and Metadata Management, pp. 4–15 (2007)
10. Kensche, D., Quix, C., Chatti, M.A., Jarke, M.: GeRoMe: A Generic Role Based Metamodel for Model Management. Journal on Data Semantics VIII, 82–117 (2007)
11. Larson, J.A., Navathe, S.B., Elmasri, R.: A Theory of Attribute Equivalence in Databases with Application to Schema Integration. IEEE Trans. Software Eng. 15(4), 449–463 (1989)
12. Lenzerini, M.: Data Integration: A Theoretical Perspective. Proc. PODS, pp. 233–246 (2002)
13. Melnik, S., Bernstein, P.A., Halevy, A.Y., Rahm, E.: Supporting Executable Mappings in Model Management. In: Proc. SIGMOD Conf, pp. 167–178. ACM Press, New York (2005)
14. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: A Programming Platform for Generic Model Management. In: Proc. SIGMOD, pp. 193–204. ACM, New York (2003)
15. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: The Use of Information Capacity in Schema Integration and Translation. In: Proc. VLDB, pp. 120–133. Morgan Kaufmann, Washington (1993)
16. Pottinger, R., Bernstein, P.A.: Merging Models Based on Given Correspondences. In: Proc. VLDB, pp. 826–873. Morgan Kaufmann, Washington (2003)
17. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10(4), 334–350 (2001)
18. Sabetzadeh, M., Easterbrook, S.: View merging in the presence of incompleteness and inconsistency. Requirements Engineering 11(3), 174–193 (2006)
19. Spaccapietra, S., Parent, C., Dupont, Y.: Model Independent Assertions for Integration of Heterogeneous Schemas. VLDB Journal 1(1), 81–126 (1992)