GeRoMeSuite: A System for Holistic Generic Model Management

David Kensche Christoph Quix Xiang Li Yong Li
Informatik 5, RWTH Aachen University
Ahornstr. 55, 52056 Aachen, Germany
{kensche,quix,lixiang,liyong}@i5.informatik.rwth-aachen.de

ABSTRACT

Manipulation of models and mappings is a common task in the design and development of information systems. Research in Model Management aims at supporting these tasks by providing a set of operators to manipulate models and mappings. As a framework, GeRoMeSuite provides an environment to simplify the implementation of model management operators. GeRoMeSuite is based on the generic role based metamodel *GeRoMe* [10], which represents models from different modeling languages (such as XML Schema, OWL, SQL) in a generic way. Thereby, the management of models in a polymorphic fashion is enabled, i.e. the same operator implementations are used regardless of the original modeling language of the schemas. In addition to providing a framework for model management, GeRoMeSuite implements several fundamental operators such as Match, Merge, and Compose.

1. INTRODUCTION

Information systems often contain components that are based on different models or schemas of the same or intersecting domains of discourse. This is by nature the case for peer data management systems or information integration systems but it is also a common problem of other information systems. Furthermore, these different models of related domains are described in modeling languages (or metamodels) that fit certain requirements of the components such as representation power or tractability. For instance, a database may use SQL or an object oriented modeling language. A web service described in XML Schema may be enriched with semantics by employing an ontology of the domain. The business objects in the middle tier may be implemented using another object oriented language such as Java or C#.

Integrating these heterogeneous models requires different means of manipulation for models and schema mappings. Research in model management aims at developing an algebra for manipulation of models and mappings. This includes operators such as Match that computes a mapping between two models [15], Compose that composes two mappings [6], ModelGen that transforms models between modeling languages [1], or Merge [14] that integrates two models based on a mapping in between.

VLDB '07, September 23-28, 2007, Vienna, Austria.

Scenarios for the application of model management operators have been described in [3, 5]. For instance, consider a system that integrates data from various sources (with models $M_1, \ldots, M_n$) into an integrated model M representing the data of a data warehouse or a semantic integration system in the semantic web. In such a heterogeneous environment, different modeling languages are used to model the data sources. If a new source with model M_{n+1} is added, the model has first to be matched with the existing integrated model to create a mapping between M_{n+1} and M.[1] If the new model M_{n+1} contains elements which cannot be mapped to the integrated model M, the models have to be integrated using the Merge operator to create a new integrated model M'. In addition, the Merge operator will deliver a mapping from the original model M to the new model M'. This mapping has to be composed with the existing mappings from $M_1, \ldots, M_n$ to M to create new mappings with the updated M' as target.

Apart from studying these metadata management tasks in an isolated fashion, a holistic framework for performing the operations is required that supports end-to-end model manipulation as the various models and mappings depend on each other. Our tool GeRoMeSuite is a framework for such model manipulations and is based on the generic role based metamodel *GeRoMe* [10]. It is a framework as it allows for fast and easy integration of new model management operator implementations. It includes a workspace for storing and loading models, mappings, and operator configurations. GeRoMeSuite provides a framework for *holistic* generic model management; unlike other model management systems it is neither limited by nature to certain modeling languages [8, 13] nor to certain model management operators [1, 2].

We integrated an implementation of the Match operator for computing correspondences between model elements. The implementation is highly configurable by enabling the definition of arbitrary matching strategies including individual and composite matchers, aggregation strategies and the configuration of each step. The correspondences computed by Match serve as the starting point for defining *formal intensional* mappings between models that are used by the Merge operator [14]. These mappings can be extended to extensional mappings for data translation. In our framework, the extensional mappings are based on second order tuple generating dependencies (SO tgds) [6]. GeRoMeSuite includes a mapping composition algorithm and a component to execute SO tgds by translating them into several query languages such as XQuery and SQL.

The next section describes our tool in general and dwells on the operator implementations that we already integrated into the tool. The last section describes our demonstration.

[1] There are other possibilities than using Match to create this mapping, but for simplicity of the presentation here, we just consider this scenario.

2. SYSTEM COMPONENTS AND ARCHITECTURE

The system is based on *GeRoMe* [10] that employs the role based modeling approach. Each elementary feature of a modeling construct is represented by a role class such as *Attribute*, *Association*, *Reference*, or *Abstract*. A single model element is described by letting it play an appropriate set of roles. In doing so, the element is decorated with features represented by roles that serve as interfaces to model elements. Operator implementations are indifferent to the native underlying metamodel such as XML Schema or OWL and are only interested in the roles exhibited by the manipulated model elements. Therefore, the operators can be used polymorphically regardless of the concrete metamodels. In addition, operators may focus only on roles relevant for their execution and may remain agnostic about other roles of a model element. Thus, operator implementation is simplified as irrelevant features of a model element may be ignored by the programmer.

GeRoMeSuite provides currently import and export operators for SQL, XML Schema, and OWL, but the tool is open for new modeling languages. As the import/export operators are implemented declaratively using a rule-based approach [9], a new metamodel can be integrated by specifying equivalence rules expressing the relationships between a concrete modeling language and *GeRoMe*. The use of equivalence rules guarantees a consistent implementation of the import and export operators. The rules are evaluated using a meta-program implemented in SWI-Prolog (`http://www.swi-prolog.org`). The main part of GeRoMeSuite is implemented in Java using SWT for the user interface and the Eclipse Modeling Framework (`http://www.eclipse.org/emf/`) for the representation of mappings.

GeRoMeSuite supports three different types of schema mappings. Informal morphisms are the result of the Match operator. These mappings are similarity values for pairs of elements from two models. Such morphisms usually serve as the starting point for defining formal intensional or extensional mappings. Intensional mappings describe the relation between the real world sets represented by model elements using statements about equality, disjointness or subset relationships. These mappings are necessary for schema merging and ontology alignment [14]. GeRoMeSuite contains an editor for intensional mappings that takes morphisms as input. On the other hand, extensional mappings are queries that can be actually executed for data and query translation. Our implementation of the Compose operator uses this type of mappings.

In addition to the *genericness* provided by *GeRoMe*, the tool supports *holistic* model management. That is, it is not limited by nature to certain operators such as Match. Instead, it provides appropriate hooks for storing and accessing the objects manipulated by model management operators. Models, different kinds of mappings, and operator configurations are managed in a central repository. Operator configurations are created through the user interface and stored uniformly no matter what operators they configure. Every configurable operator is required to provide a default configuration that is the basis for the definition of new configurations.

New operator implementations that use *GeRoMe* as their model representation can be integrated into the tool. All the necessary objects are available to operators through the workspace repository. So far, we have integrated implementations of the Match, Merge, and Compose operators into GeRoMeSuite. The following sections describe how to work with these implementations within our tool.

2.1 Schema Matching

A wide range of schema or ontology matching systems already exist, such as COMA++ [2] or Protoplasm [4]. In a survey on schema matching [15], existing approaches have been classified into somewhat orthogonal categories. One of the important dimensions that has been exploited already by the two mentioned tools is the combination of individual matchers to new composite matchers.

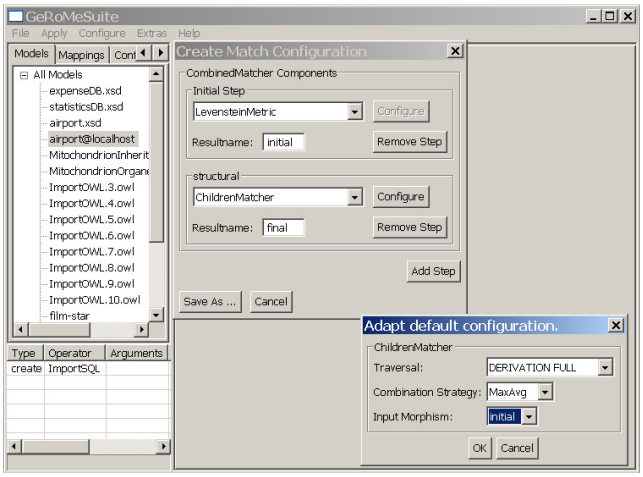

Figure 1: Creating a Match configuration

The Match implementation, that we integrated into GeRoMeSuite, also allows for definition of composite matchers. We provide high flexibility for matcher combination: element level matchers, structure level matchers, morphism filters, and aggregation strategies for combining match results can be assigned to subsequent steps. Thereby, it is possible to define new composite matchers that can serve again as building blocks for other more complex composite matchers. Fig. 1 shows such a configuration dialog.

New matcher steps can be easily added, named and configured. After storing the new match configuration, it can be used as a default component configuration in a new matcher configuration. When executing the match operator, an existing matcher configuration is chosen and can be adapted in a dialog that contains all the parameters exposed by its components in distinct panels.

We currently provide several element-level matchers to compare string labels, and structural matchers such as 'ChildrenMatcher' (the similarity of nodes is given by the similarity of their children) and Similarity Flooding [12]. Especially the structural matchers can exploit the semantically rich structure of a *GeRoMe* model. For example, a *GeRoMe* model can be iterated in several different ways using either the inheritance, namespace or aggregation/association hierarchy. A comparison with existing matching systems has shown that the uniform representation of models in *GeRoMe* is beneficial in particular for heterogeneous matching problems, such as matching an XML Schema with an OWL ontology. New filters, aggregation strategies, element level and structure level matchers can be

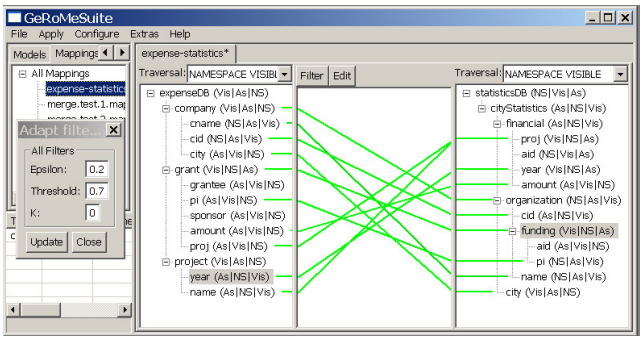

Figure 2: Filtering result morphisms

integrated by implementing the appropriate interface and providing a default configuration for the new component.

Like in other matching tools, the result of Match is a set of confidence measures that assess the similarity of the respective model elements informally. As a first step these can be filtered using different filters (similar to those available in COMA++ [2]), a simple threshold, a maximum distance from the best match, and a parameter k that specifies the number of best matches to be kept. These can be combined conjunctively. Fig. 2 shows this filtering step. These filters can also be integrated as components into a matching strategy which affects the operation of subsequent matching steps.

2.2 Schema Merging

The informal morphism, that is the result of schema matching, can be used as input for the intensional mapping editor that produces a formal mapping which is necessary for merging models. Our intensional mappings [14] are nested mappings that contain correspondence assertions such as equality, subset or disjointness relationships. Fig. 3 shows the intensional mapping editor. The implementation of the Merge operator is based on a well-defined theoretical foundation that defines the intensional mappings as set relationships between the real world semantics (RWS) of the corresponding model elements. The RWS is used to formally characterize the merged model which can be described in brief as the duplicate-free least upper bound which retains the granularity of the input models (details are given in [14]).

As the result of Merge also depends heavily on the application scenario, such as database or view integration, preferences have to be set here as well. In the first step of our merging algorithm [14] elements equally related in the mapping are grouped. Such groups of equivalent elements are collapsed into one element. Conflicts arising in this step must then be resolved. Subsequent steps similarly deal with other mapping relationships such as *IsA* or *Overlapping*. Conflicts are resolved according to configurable strategies. These may be automatic resolution strategies which use the semantic information given in the model and the mapping, or some simpler predefined default strategies such as always preferring the representation chosen in one of the two input models. The final fallback for conflict handling asks the user for manual conflict resolution.

The result of merging is a valid *GeRoMe* model. However, it need not be a valid model according to some certain native metamodel. For instance, when merging an XML Schema with a relational schema the result is most likely not a valid relational schema. Thus, the merge result has to be transformed to the target metamodel using the ModelGen operator before it can be exported.

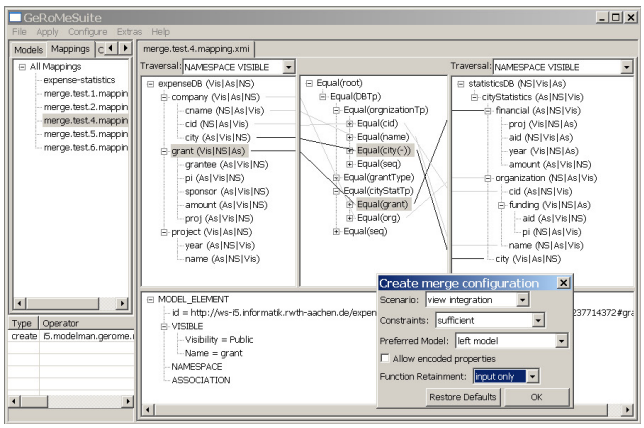

Figure 3: Defining intensional mappings and configuring Merge

2.3 Mapping Composition

Schema merging requires intensional mappings that describe the relationships of the real world sets represented by model elements. On the other hand, these mappings are less useful for mapping execution. For this task we need extensional mappings that describe the relationships between two models on the instance level, including conversion functions.

We integrated into GeRoMeSuite both, an implementation of the Compose operator for extensional mappings as second order tuple generating dependencies (SO tgds) between *GeRoMe* models, and an export functionality that produces a program from such an extensional mapping that migrates data between combinations of relational and XML schemas [11]. Our extensional mappings are closed under composition as they are based on the second order dependencies introduced in [6] and they also support grouping of elements such as the mappings of [7]. In doing so, complex transformations such as restructuring of data are possible which is a fundamental requirement for data translation between different modeling languages. Figure 4 shows an example of a mapping between two *GeRoMe* models representing XML Schemas, the generated XQuery to retrieve the data from the source, and example documents for source and target.

In accordance to the definition of SO tgds, the mapping allows universal quantifiers over variables, existential quantifiers over function symbols, and equalitites. To reflect the structure of *GeRoMe* instances [10], its body contains combinations of $inst$, $part$, $value$, $attrvalue$ predicates. Basically, these predicates describe the instances of a *GeRoMe* model using reification. The first component of each predicate is an abstract identifier for the object. The predicate $inst(o, x)$ specifies that the abstract object o is an instance of the model element x. In *GeRoMe*, XML Schema elements are represented as associations between the nesting complex type and the nested type (simple or complex). The entry into an XML document using an association hierarchy is a special case as the root element specifies an association between a "virtual" document object and the root object. In the example of fig. 4, the 'source' element o_1 is an association between the "virtual" document object o_0 and the object o_2 representing the root of the document. o_0 and o_2 are specified as participators of the association o_1 by using the $part$ predicate. The next level of nesting is represented as an association (o_3) between the root object o_2 and the nested 'person' object o_4. The $value$ predicate retrieves the value for simple typed XML elements such as 'name' and 'school'. The predicate $attrvalue(o, a, v)$ specifies that v is a value of the attribute a for the object o.

The structure of the target document is determined by the skolem functions which are introduced in the conclusion part of the rule. For example, for each instance of the 'source' element, we will have one corresponding 'target' element. This is trivial because there is only one such element (note that a valid XML document must have exactly one root element). For the 'school' element, we assume that the name s of the school is the key. Therefore, the skolem function fs for the school element depends on s. Thereby, the grouping of all students of a school is enabled. If we would have chosen a different skolem term (e.g. $fs(o_3)$), the target would contain a school element for each school/student pair, thus having repeated school elements for "Manuden School".

The example uses only skolem functions for which no explicit semantics has to be defined, i.e. the skolem functions determine only the structure of the generated code for the mappings but are not evaluated during mapping execution. For value conversions or some other data translation functions, explicit functions can be specified by giving the name of a Java class that implements the

$$\exists f \exists fs \exists fsc \exists fst \exists fstc \quad \forall o_0 \forall o_1 \forall o_2 \forall o_3 \forall o_4 \forall o_5 \forall o_6 \forall o_7 \forall o_8 \forall n \forall s$$
$$inst(o_1, \text{'source'}) \wedge part(o_1, \text{'parent'}, o_0) \wedge part(o_1, \text{'child'}, o_2) \wedge inst(o_3, \text{'person'}) \wedge part(o_3, \text{'parent'}, o_2) \wedge part(o_3, \text{'child'}, o_4) \wedge$$
$$inst(o_5, \text{'name'}) \wedge part(o_5, \text{'parent'}, o_4) \wedge part(o_5, \text{'child'}, o_6) \wedge value(o_6, n) \wedge$$
$$inst(o_7, \text{'school'}) \wedge part(o_7, \text{'parent'}, o_4) \wedge part(o_7, \text{'child'}, o_8) \wedge value(o_8, s) \rightarrow$$
$$inst(f(o_1), \text{'target'}) \wedge part(f(o_1), \text{'parent'}, f(o_0)) \wedge part(f(o_1), \text{'child'}, f(o_2)) \wedge$$
$$inst(fs(s), \text{'school'}) \wedge part(fs(s), \text{'parent'}, f(o_2)) \wedge part(fs(s), \text{'child'}, fsc(s)) \wedge attrvalue(fsc(s), \text{'name'}, s) \wedge$$
$$inst(fst(s,n), \text{'student'}) \wedge part(fst(s,n), \text{'parent'}, fsc(s)) \wedge part(fst(s,n), \text{'child'}, fstc(s,n)) \wedge value(fstc(s,n), n)$$

```
for $o2 in fn:doc(fname)/source
    for $o4 in $o2/person
        for $o6 in $o4/name
        for $o8 in $o4/school
return
<result>
   <s>$o6/text()</s>
   <n>$o8/text()</n>
</result>
```

```
<source>
  <person>
    <name>John Parker</name>
    <school>Manuden School</school>
    <school>York School</school>
  </person>
  <person>
    <name>Robert Adam</name>
    <school>Manuden School</school>
    <school>King School</school>
  </person>
</source>
```

```
<target>
  <school name="Manuden School">
    <student>John Parker</student>
    <student>Robert Adam</student>
  </school>
  <school name="York School">
    <student>John Parker</student>
  </school>
  <school name="King School">
    <student>Robert Adam</student>
  </school>
</target>
```

Figure 4: SO tgd between two XML Schemas, the generated XQuery, and the corresponding input and output documents

function. This class will then be used during mapping execution. As the example shows, extensional mappings are very complex; we will therefore develop a graphical mapping editor also for this type of mappings which will be based on the editors for morphisms and intensional mappings.

As it would be very inefficient to transform the source data into *GeRoMe* instances, we translate the mappings into queries which can be directly evaluated on the source data. Fig. 4 shows the generated XQuery to retrieve the data from the source. The result is a list of flat tuples with names of schools and persons which will be used to generate the target data. Due to the genericness of our approach, we do not generate the target XML document directly, as the target could be also a relational database or a Java object model.

3. DEMONSTRATION

The demo scenario will be based on the scenario sketched in the introduction. For two given input models, a relational schema and an XML Schema, we will first adapt and use a match configuration to execute the matcher subsystem. This will yield a morphism between the two input models. We will then use this mapping as the starting point for formulating a formal intensional mapping which serves as the input to our Merge implementation. A predefined merge configuration is adapted to the scenario and then applied. The merge operator will have to ask for user advice at certain stages, e.g. for choosing the result name of two collapsed model elements. The result will be a *GeRoMe* model that can be exported to a valid XML schema. As variations of this basic scenario, we can also match an XML schema and an ontology, or match and merge two ontologies to show the genericness of GeRoMeSuite.

In the next step, we will show the mapping composition and execution component. Two formal extensional mappings between the input models and the merged model will be composed. The execution of the composed mapping can be compared with the execution of the individual mappings. Again, the genericness of GeRoMeSuite can be shown by applying the mapping composition algorithm to mappings between models originally represented in different modeling languages, and by translating the mappings into executable queries in different query languages.

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

4. REFERENCES

[1] P. Atzeni, P. Cappellari, and P. A. Bernstein. Model-independent schema and data translation. In *EDBT*, volume 3896 of *LNCS*, pages 368–385. Springer, 2006.

[2] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *Proc. SIGMOD Conf.*, pages 906–908. ACM Press, 2005.

[3] P. A. Bernstein. Applying model management to classical meta data problems. In *Proc. CIDR*, Asilomar, CA, 2003.

[4] P. A. Bernstein, S. Melnik, M. Petropoulos, and C. Quix. Industrial-strength schema matching. *SIGMOD Record*, 33(4):38–43, 2004.

[5] P. A. Bernstein and E. Rahm. Data warehousing scenarios for model management. In *Proc. ER 2000*, pages 1–15, 2000.

[6] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[7] A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested mappings: Schema mapping reloaded. In *Proc. VLDB'06*, pages 67–78, Seoul, 2006.

[8] M. A. Hernández, R. J. Miller, and L. M. Haas. Clio: A semi-automatic tool for schema mapping. In *SIGMOD*, 2001.

[9] D. Kensche and C. Quix. Transformation of models in(to) a generic metamodel. In *Proc. BTW Workshop on Model and Metadata Management*, pages 4–15, 2007.

[10] D. Kensche, C. Quix, M. A. Chatti, and M. Jarke. *GeRoMe*: A generic role based metamodel for model management. *Journal on Data Semantics*, VIII:82–117, 2007.

[11] D. Kensche, C. Quix, Y. Li, and M. Jarke. Generic schema mappings. In *Proc. ER'07*, 2007. To appear.

[12] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. ICDE*, 2002.

[13] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A programming platform for generic model management. In *Proc. SIGMOD*, pages 193–204. ACM, 2003.

[14] C. Quix, D. Kensche, and X. Li. Generic schema merging. In *Proc. CAiSE'07*, LNCS, pages 127–141, 2007.

[15] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.