A platform for model-independent solutions to model management problems (extended abstract)

Paolo Atzeni, Luigi Bellomarini, Francesca Bugiotti, and Giorgio Gianforme*

Università Roma Tre

Abstract. Model management is an approach to metadata management aimed at supporting the productivity of developers by providing high level operators defined on schemas and mappings over them.
Here we propose a platform for model management that handles schemas in different models, with explicit reference to the models themselves. Therefore, this is a model-generic, model-aware approach, and we believe that it is the first proposal in this direction.
We consider the major operators in model management: merge, diff, modelgen. To confirm the effectiveness of the approach, we show a general solution to the round-trip engineering problem.

1 Introduction

The need for complex transformations of data arises in many different contexts, because of the presence of multiple representations for the same data or multiple sources that need to coexist or to be integrated [6, 8, 9]. A major goal of technology in the database field is to enhance the productivity of software developers, by offering them techniques that allow for high-level specifications and abstraction over recurring tasks. This has been stressed since the introduction of the relational model, with the emphasis on set-oriented operations [7], but it was implicit in much earlier developments of generalized techniques [10]. The *model management* proposal [5, 4] is a recent, significant effort in this direction: its goal is the development of techniques that consider metadata and operations over them. More precisely, a model management system [6] should handle schemas and mappings between them and support operations to match and merge schemas, translate them from a data model to another (by means of operators called MATCH, MERGE, DIFF, MODELGEN, and others). These operations should be specified at a high level, on schemas and mappings, and should allow for the (support to the) generation of data-level transformations. Many application areas can benefit from the use of model management techniques, including data integration over heterogeneous databases, data exchange between independent databases, ETL (Extract, Transform, Load) in data warehousing, wrapper generation for the access to relational databases from object-oriented

* Supported by a Microsoft fellowship.

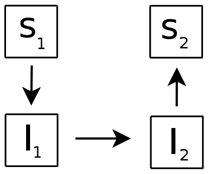
$$\begin{tikzcd} S_1 \arrow[d] & S_2 \\ I_1 \arrow[r] & I_2 \arrow[u] \end{tikzcd}$$

Fig. 1. The round-trip engineering problem.

applications, dynamic Web site generation from databases. As an example of a problem of interest in model management let us recall the "round-trip engineering problem" [5]. Consider (see Figure 1) a specification S_1 and its corresponding implementation I_1 as provided by a design tool, and a "manual" modification of the implementation, which leads to a new version I_2, which is not coherent with S_1; the goal is to find a specification S_2 from which I_2 could be generated. A concrete case for this would involve a high level specification tool which translates ER schemas into relational tables by generating appropriate SQL DDL. Then, if a data architect wants some changes on the DDL, the changes have to propagate backwards and the ER specification has to be updated. The model management solution to this problem is based on a script that requires the application of various operators, which detect the actual differences between the original and the modified implementation, translate this portion back to the specification and finally integrate the original specification with the elements that correspond to the new or modified portion of the implementation [5].

Most of the work in model management has considered the need for *model independence*, that is the fact that the techniques do not refer to individual data models, but are more general. This has usually been done by adopting some "universal data model" or *supermodel* [3], a model that is more general than the various models of interest in a heterogeneous framework. If the operations of interest include also translations from a data model to another (the MODELGEN operator), it is important that the individual data models are represented, in such a way that it becomes possible to describe the fact that a schema belongs to a data model. The various proposals for schema translation (the MODELGEN operator) [3, 12, 13] do include such a feature, to a larger or lesser extent. Instead, the major efforts in the model management area (as summarized by Bernstein and Melnik [6]) do not handle the explicit representation of data models.

The goal of this paper is to provide contributions towards the development of model-independent but model-aware solutions to model management problems. We leverage on our experience in the MIDST platform [1, 2], where a model-independent approach for MODELGEN was proposed. MIDST handles the artifacts of interest in a repository that represents data models, schemas and databases in an integrated way. Models are described with a metalevel approach. We use a reduced and extensible set of constructs to represent the concepts of the models of interest. The supermodel, in this approach, is the model that includes all the available constructs. Translations from a model to another are implemented in datalog with OID-invention. The tool has a library of translations and can find the appropriate sequence of rules for a specific translation (given source and target models).

Here we extend the MIDST platform to a general model management system, with the major operators, including MERGE, DIFF, and a version of MATCH, all implemented in a model-generic way, which takes advantage of the features of our multilevel dictionary. We will also show a complete solution to the round-trip engineering problem based on our platform. The implementations of the operators are written in datalog, with rules that are generated automatically, with respect to the given supermodel, so that if the supermodel is extended, the operators can be extended as well. To the best of our knowledge, this is the first proposal for a model-independent platform for model management.

The rest of this extended abstract is organized as follows. In Section 2 we discuss the definitions and implementation of the main operators, with a model-independent approach. In Section 3 we illustrate their application to a round-trip engineering scenario. Finally in Section 4 we draw our conclusions.

2 Operators

In this section we show how the MIDST platform can be extended to handle most model management operators.

MIDST [1, 2] is based on a metalevel approach, with a small set of metaconstructs (that is, types of constructs) which can be used to define models in terms of the constructs they involve (and of their types). It supports translation of schemas and data between different models (the MODELGEN operator). The platform includes a dictionary that handles models, schemas and data, over which translations are specified as datalog rules.

Here we show that the approach can be extended to a wider context. In fact, we use datalog rules to specify other general transformations among schemas, including all model management operators (DIFF, MERGE and MATCH). In this way model management operators do not refer to a specific schema or model, but are defined with respect to MIDST dictionary metaconstructs with significant benefits, as follows.

Operators are model-independent since they are defined over the metaconstructs, however they are model-aware, as each schema belongs to a model. This implies that if an operator is applied to a set of schemas of a given model, the output schemas will belong to a given model as well (which can be the same or another one, depending on the operator). With respect to the new model management operators we discuss here (DIFF and MERGE), this is essentially a *model closure property*: these transformations do not add constructs which are not consistent with the model the input schemas belong to.

The new operators are also supermodel-independent, in the sense that they do not depend on the chosen representation of models. In fact, each operator is composed of a set of rules defined on the basis of constructs: for example, for each type of construct there is a rule that compares constructs of such a type, a rule that copies them, and so on. As a consequence, the rules for these operators can be automatically generated by an appropriate MIDST metalevel preprocessing phase.

Most common operators in model management problems solving procedures [5] are DIFF, MERGE, and MATCH.

DIFF and MERGE work as follows: they respectively generate the set-oriented difference and union of two given schemas. As we said, in our working framework, schemas are composed of constructs directly derived from MIDST metamodel. A construct can be considered as an object with a unique identifier, a name and some properties describing the domain of interest. As it happens in object-oriented models, constructs are related to one another by means of references. Coherently with the object-oriented model, we handle typed references, in the sense that they define links between specific types of constructs. Therefore, set-oriented operations must also consider references in "summing" and "subtracting" constructs.

Both DIFF and MERGE are defined in a model-independent way: they operate on every possible MIDST metaconstruct without any specific knowledge of the role played by each construct in a given application scenario. They must only be provided with some initial information about correspondences between the involved schemas.

The difference has to be aware of the correspondences between the two input schemas so as to copy only the appropriate constructs into the destination by detecting the equal ones. Similarly, the MERGE operator has to be aware of those correspondences not to generate duplicates in the result schema: a construct which appears in both the input schemas must in fact appear in the result schema once only.

In model management, these issues are frequently handled by means of the specification of a mapping between the two schemas, where the mapping is usually obtained as the result of a MATCH operation. Here, we argue that some mapping information is necessary, but we do not implement a general matching operation. In fact, we adopt a *unique name assumption*: two constructs are equal if their names and properties are lexically identical and if their ancestors (referred by references) are equal themselves. Therefore there is the need for a hierarchical comparison of schema constructs which, in the base case (constructs without references), leads to a positive or negative answer. This is a sort of implicit match and could be replaced by a more general MATCH operator.

The comparison based on the unique name assumption is implemented in datalog as a part of the operators. Once the operators are provided with the necessary correspondences, specific tasks can be performed: while the difference operates a selective copy of the constructs present only in one input schema, the MERGE unites the constructs avoiding duplicates.

The definitions of DIFF and MERGE have some specific features, in terms of the number of arguments and results. DIFF takes two input schemas and computes an algebraic difference between them, with two results ("semi-differences"), a "positive" (the constructs in the second operand but not in the first one) and a "negative" one (the converse). MERGE takes three input schemas and returns one result: the first and the third parameters are united in a set-oriented fashion, while the second one is subtracted.

In the DIFF context, managing references leads to a further caveat: the outcome of a difference can be an inconsistent schema, meaning that dangling references could be present. Consider a schema S with a construct C_1 referring to C_2. Suppose that only C_1 belongs to a difference DIFF(S, S') because C_2 appears also in schema S'. Then the resulting schema is inconsistent because C_1 has a dangling reference. To handle this situation, which is indeed common, we introduce *stub constructs*. They are constructs which should not be in the difference result according to its semantics. They are inserted in order to avoid dangling references but they are tagged in such a way that subsequent operations deal with them in an appropriate way.

3 Round-trip engineering

In this section we show how our approach can handle the round-trip engineering problem, following the general technique provided by Bernstein [5], with some simplification due to our unique name assumption. Consider Figure 1 again: S_1 is the *specification schema* and I_1 the *implementation* schema. Let I_2 be a modified version of I_1. Bernstein substantially proposes the application of DIFF between I_1 and I_2 in order to detect added and removed constructs. Then a reverse step translates the difference result into the specification domain and, finally, a merge between the original specification and the reversed difference can be worked out in order to obtain the updated specification S_2.

On the basis of the briefly described procedure, we propose a revised script which uses MIDST model-independent operators (Figure 2).

1. $<G_2'^{+}, G_2'^{-}>$ = DIFF(I_1, I_2): we compute the difference between the implementation schema and its modified version. $G_2'^{+}, G_2'^{-}$ are the positive and negative semi-differences, respectively.
2. $S_3'^{-}$ = MODELGEN$(G_2'^{-}$, reverse): $S_3'^{-}$ is generated from $G_2'^{-}$ through the application of a datalog translation rule. If map_1 in Figure 2 is the datalog rule translating the specification schema into its corresponding implementation, the mapping *reverse* applies the opposite transformation. This step, known as *reverse step*, has fundamental importance in the round-trip engineering problem and distinguishes it from the *forward engineering problem*.
3. $S_3'^{+}$ = MODELGEN$(G_2'^{+}$, reverse): the same for the other semi-difference.
4. S_2 = MERGE$(S_3'^{-}, S_3'^{+}, S_1)$: we merge the original specification schema with the two reversed semi-differences in order to update S_1 by deleting the removed constructs and adding the new ones.

Now we present a complete example of application of the described round-trip solving procedure. The specification domain is the ER model, while the implementations are relational schemas.

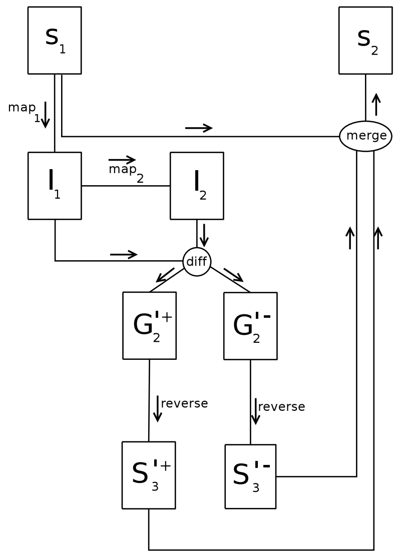

Fig. 2. A procedure for the round-trip engineering problem.

In Figure 3, the schema S_1 is composed of two entities, A and B, and a relationship R between them. A_1 and A_2 are A attributes, A_1 is key, while B_1 and B_2 are B attributes, B_1 is key. The relationship in S_1 corresponds to the foreign key in I_1. The mapping map_3 describes a schema evolution operator: it modifies the relation A by changing its key (from A_1 to A_2), and B by eliminating the attribute B_1 and adding the attribute B_3. The foreign key that in I_1 connects the attribute B_1 of table A with the relation B, does not exist anymore, it is replaced by a new one from the attribute B_2 of A to the relation B.

The first step of the solving procedure is the application of the DIFF rule to I_1 and I_2 which yields $G_2'^{-}$ and $G_2'^{+}$. $G_2'^{-}$, the negative difference, contains the attributes A_1 (key), A_2 and B_1 of the relation A because they are in I_1 but not in I_2, in fact they have been removed. The relation A is present as a stub for these attributes (denoted by the subscript "s" in Figure 3). As for the relation B, it is present in I_1 with the attributes B_1 (key) and B_2, and in I_2 with B_2 (key) and B_3, then the negative difference $G_2'^{-}$ contains B_1 (key) and B_2 and B itself as a stub. Notice that an attribute with a certain name is different from another one with the same name but different properties; for example A_1 is present in the negative difference as an identifier attribute because we find it in I_2 as a non identifying attribute. The negative difference also has the S_1 foreign key since it has been removed. $G_2'^{+}$, the positive difference, as for the relation A, contains A_2 (key), A_1 and B_2, because these attributes are present in I_2 but not in I_1. The same for the attributes B_2 and B_3 belonging to the relation B. Both the relations are stub as well as in $G_2'^{+}$. Then, since the key attributes have changed, in the positive difference there is the new foreign key from the attribute B_2 towards the relation B.

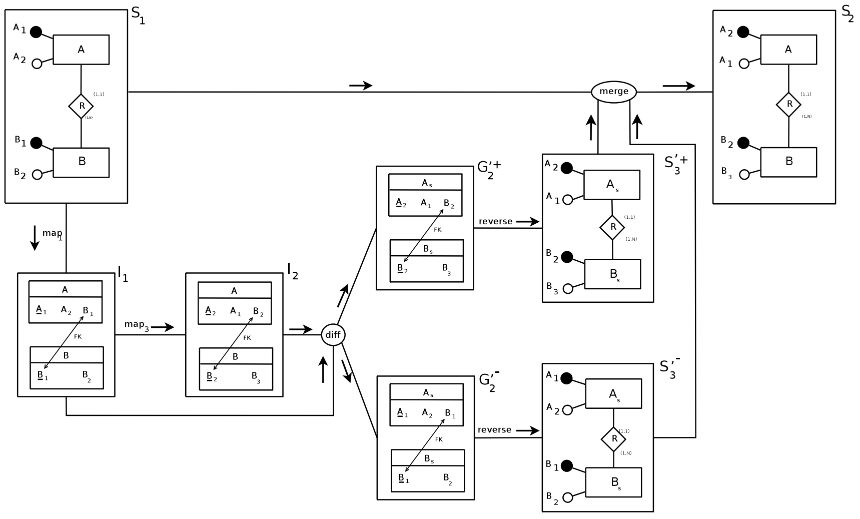

Fig. 3. An example of application of the round-trip solving procedure.

Then each semi-difference must be reversed with the application of the MODELGEN operator, loaded with the datalog rule which translates relational schemas into ER. In the case under examination the reverse translation is simple, in general it might be much more complex and involve several constructs. Notice that in the application of the reverse rule, the stubness property of constructs is preserved, then for example the entity A in $S_3'^{+}$ is stub as well as in $G_2'^{+}$.

Now we have three different versions of the specification: the original one, S_1, $S_3'^{-}$ including all the constructs that have to be removed, and $S_3'^{+}$ containing all the added ones. It is clear that the set-oriented merge of these three schemas leads to an updated specification. Then the procedure applies the MERGE operator that subtracts and adds constructs to S_1. As for the relation A, the attributes A_1 and A_2 in S_1 are also contained in $S_3'^{-}$ hence removed. The same happens to the attributes B_1 and B_2 of B. The relationship R in S_1 has the same name of the relationship in $S_3'^{-}$ and connects equivalent constructs, so it is removed since an equivalent construct has been found in the negative difference. The entities A and B in S_1 are copied into the destination schema S_2 because the two semi-differences only have their stub equivalents, which are noninfluent in the MERGE. Finally, all the non-stub constructs of the positive part of the difference $S_3'^{+}$ are copied to S_2: so the attributes are updated and the relationship between A and B is restored.

4 Conclusions and related work

This paper relies on our previous works on model-generic schema and data translation [1, 2] describing our conception and implementation of the MODELGEN operator. There are many proposals addressing model management problems which have been put forward since the original formulation of the problem.

In [4] Bernstein et al. recognize the possibility of a generic metadata approach to model management: their theoretical formalizations [5] and later studies converge in Rondo, a programming platform for model management [11]. However their approach is not supported by a dictionary of models and so they pursue model independence without a concrete characterization of models. Conversely, MIDST uses a relational dictionary of models and schemas to actually represent models and allows transparent transformations on them.

To summarize, our approach proposes MIDST as a platform for model-independent model management; its main innovative contributions are model-independent but model-aware operators and the automatic generation of operators, both enabled by a rich though simple and thorough metalevel representation of data models.

References

1. P. Atzeni, P. Cappellari, and G. Gianforme. MIDST: model independent schema and data translation. In *SIGMOD Conference*, pages 1134–1136. ACM, 2007.
2. P. Atzeni, P. Cappellari, R. Torlone, P. A. Bernstein, and G. Gianforme. Model-independent schema translation. In *VLDB Journal, To appear.*
3. P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. In *EDBT Conference, LNCS 1057*, pages 79–95. Springer, 1996.
4. P. Bernstein, L. Haas, M. Jarke, E. Rahm, and G. Wiederhold. Panel: Is generic metadata management feasible? In *VLDB*, pages 660–662, 2000.
5. P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR Conference*, pages 209–220, 2003.
6. P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.
7. E. Codd. Relational database: A practical foundation for productivity. *Commun. ACM*, 25(2):109–117, 1982.
8. L. M. Haas. Beauty and the beast: The theory and practice of information integration. In T. Schwentick and D. Suciu, editors, *ICDT*, volume 4353 of *Lecture Notes in Computer Science*, pages 28–43. Springer, 2007.
9. A. Y. Halevy, N. Ashish, D. Bitton, M. J. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD Conference*, pages 778–787, 2005.
10. W. C. McGee. Generalization: Key to successful electronic data processing. *J. ACM*, 6(1):1–23, 1959.
11. S. Melnik. *Generic Model Management: Concepts and Algorithms.* Springer-Verlag, 2004.
12. P. Mork, P. A. Bernstein, and S. Melnik. Teaching a schema translator to produce O/R views. In *ER Conference, LNCS 4801*, pages 102–119. Springer, 2007.
13. P. Papotti and R. Torlone. Heterogeneous data translation through XML conversion. *J. Web Eng.*, 4(3):189–204, 2005.