A runtime approach to model-independent schema and data translation (extended abstract*)

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

Dipartimento di informatica e automazione
Università Roma Tre

Abstract. A runtime approach to model-generic schema and data translation is proposed. It is based on our previous work on MIDST, a platform conceived to perform translations in an off-line fashion. In the original approach, the source database is imported into MIDST dictionary, where it is stored according to a universal model. Then the translation is applied within the tool as a composition of elementary transformation steps, specified as Datalog programs. Finally, the result is exported into the operational system.
Here we illustrate a new, lightweight approach where the database is not imported. The tool needs only to know the model and the schema of the source database and generates views on the operational system that transform the underlying data (stored in the source schema) according to the corresponding schema in the target model. Views are generated in an almost automatic way, on the basis of the Datalog rules for schema translation.

1 Introduction

The problem of translating schemas between data models is acquiring progressive significance in heterogeneous environments and has received attention in many works [3, 5, 7, 11, 13, 16]. Applications are usually designed to deal with information represented according to a specific data model, while the evolution of systems (in databases as well as in other technology domains, such as the Web) led to the adoption of many representation paradigms.

For example, many database systems are nowadays object-relational (OR) and so it is reasonable to exploit their full potentialities by adopting such a model while most applications are designed to interact with a relational database. Also, object-relational extensions are often non-standard, and conversions are needed. The explosion of XML, with all its applications (for example, as a format for information exchange or as the language for the semantic Web), has increased the heterogeneity of representations. In general the presence of several coexisting models introduces the need for runtime translation techniques and tools.

* This is the extended abstract of a paper that appears in the Proceedings of EDBT 2009, St. Petersburg, Russia, March 23-26.

We have recently proposed MIDST [3, 5], a platform for model-independent schema and data translation in order to provide a paradigm to face issues of this kind. MIDST adopts a metalevel approach towards translations by performing them in the context of a universal model (called the *supermodel*), which allows for the management of schemas in many different data models. In fact, the set of models that can be dealt with is large and extensible. The current version handles relational, object-oriented, object-relational, entity-relationship, XML-based, each in many different variants, and new metaconstructs can be added, if needed for handling features not covered by the current ones. Translations in MIDST are organized according to the following pattern. First, the source database is imported into the tool and described according to the supermodel whose expressiveness is also discussed in [5]. Then translations are performed by means of elementary transformation steps; finally, the obtained database, whose schema is conformed to the one expected by the application, is exported into the operational system.[1] MIDST approach provides a general solution to the problem of schema translation, with *model-genericity* (as the approach works in the same way for many models) and *model-awareness* (in the sense that the tool knows models, and can use such a knowledge to produce target schemas and databases that conform to specific target models). However, as pointed out by Bernstein and Melnik [8], this approach is rather inefficient for data exchange. In fact, the necessity to import and export a whole database in order to perform translations is out of step with the current need for interoperability in heterogeneous data environments.

Here we propose a runtime approach towards the translation problem, where data is not moved from the operational system and translations are performed directly on it.

What the user obtains at runtime is a set of views (the target schema) conform with the target model. The approach is model-generic and model-aware, as it was the case with MIDST, because we leverage on MIDST dictionary for the description of models and schemas and also on its key idea of having translations within the supermodel, obtained as composition of elementary ones, each dealing with a specific aspect (construct or feature thereof) to be eliminated or transformed. The main difference is that the import process concerns only the schema of the source database and the rules for schema translation are here used as the basis for the generation of views in the operational system. In such a way data is managed only within the operational system itself. In fact, our main contribution is the definition of an algorithm that generates executable data level statements (view definitions) out of schema translation rules.

A major difference between an off-line and a runtime approach to translation is the following. For an off-line approach, as transformations are performed within the translation tool (MIDST in our case), the language for expressing translations can be chosen once, for all models. A significant difficulty is in the import/export components, which have to mediate between the operational sys-

[1] We use the term *operational system* to refer to the system that is actually used by applications to handle their data.

tems and the tool repository, in terms of both schemas and data. In fact, in the development of MIDST, a lot of effort was devoted to import/export modules, whereas all translations were developed in Datalog. In a runtime approach, the difficulties with import/export are minor, because only schemas have to be moved, but the translation language depends on the actual operational systems. In fact, if there is significant heterogeneity, then stacks of languages may be needed (involving for example, SQL, SQL/XML, XQuery). Also, different dialects of the various languages may exist, and our techniques need to cope with them.

In order to deal with heterogeneous languages, we propose an approach that, after a preliminary abstract representation, first generates views organized according to the constructs in the target model, but independent of the specific languages, and then actually concretizes them into executable statements on the basis of the specific language supported by the operational system.

In this paper we provide, with some detail, a general explanation of the language-independent solution we obtain. The second step of the solution (where the language-independent views are adapted to the operational system representation) will not be pursued here as its specific details are not essential for a general illustration of the approach.

The remainder of the paper is organized as follows. In Section 2 we provide an end-to-end description of the approach to illustrate the achieved results first by means of a motivating example and then focusing on some technical details. In particular, we will see how relational views can be generated to access an object-relational schema with references and inheritance. In Section 3, we discuss the approach and make comparisons with related works. Section 4 concludes the paper.

2 The approach

The goal of a tool for schema and data translation is to provide support to the adoption of a wide family of heterogeneous data models. In a runtime perspective, this means that application programs, designed to interact with a specific data model M_t, would be allowed to work with another data model M_s in a transparent way. The tool we propose supports this feature by translating the schemas of M_s (which actually contain the data of interest for the programs) in terms of views of model M_t. Then the application programs would use these views to access data organized according to M_s.

The starting point for this work is MIDST [3, 5], a platform for model-independent schema and data translation based on a metalevel approach over a wide range of data models. In MIDST the various data models are described in terms of a small set of *basic constructs*. Schemas of the various models are described within a common model, the *supermodel*, which generalizes all of them, as it involves all the basic constructs. Translations refer to the basic constructs and are performed within the supermodel. In the current implementation, they are specified in Datalog.

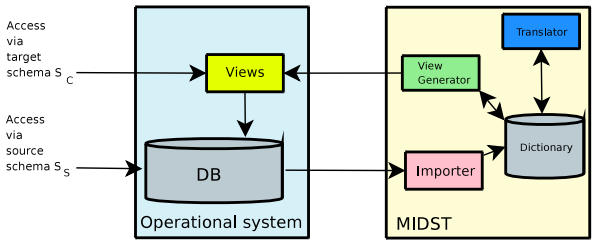

Fig. 1. The runtime translation procedure

In the previous work on MIDST, translations are dealt with in an off-line fashion, meaning that the import of both schema and data into MIDST is needed as well as an export of the result. Here we describe the approach of the enhanced version of our platform that enables the creation of executable statements generating views in the operational system. Let us illustrate the new approach we propose here, by following the main steps it involves, with the help of Figure 1:

1. given a schema S_s (of a *source* model M_s) in the operational system, the user (or the application program) specifies a *target* model M_t;
2. schema S_s (but not the actual data) is imported into MIDST, and specifically in its dictionary, where it is described in supermodel terms;
3. MIDST selects the appropriate translation $\mathcal{T}$ for the pair of models (M_s, M_t), as a sequence of basic ones available in its library;
4. the schema-level translation $\mathcal{T}$ is applied to S_s to obtain the target schema S_t (according to the target model M_t);
5. on the basis of the schema-level translation rules in $\mathcal{T}$, the tool generates views over the operational system, in three phases: first it generates an abstract description of views that specify schema S_t (in model M_t) in terms of the elements of the source schema in S_s; then it translates these abstract descriptions into system-generic SQL-like view definitions; finally it compiles statements that define the actual views in the specific language available in the operational system.

Let us observe that steps 1-4 appear also in the previous version of MIDST, whereas 5 is completely new, in all its phases, and clearly significant. It is worth observing that while steps 1-4 do not depend on the expressive power of the involved models, since translation only defines compositions and rearrangements of constructs at schema levels, for step 5 it is different. In fact, two cases may be involved: a) migration of data from a more expressive model to a less expressive one; b) migration of data from a less expressive model to a more expressive one. Obviously, case a) does not present any drawbacks, on the other hand, case b) causes the loss of the logical constraints of the more expressive model. However this issue can be forecast with a consequential customization of the behaviour operated by the user through the selection of an appropriate translation strategy.

As a motivating example, consider the following. Assume we have an environment where application programs are designed to interact with relational

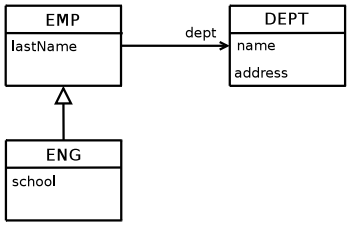

Fig. 2. A simple object-relational schema

databases while we have an actual database on the operational system based on the object-relational (OR) model, with the following features:[2] tables, typed tables, references between typed tables and generalizations over typed tables. In this scenario, we use MIDST to generate relational views over the object-relational schema, which can be directly used by application programs.

A concrete case for this example involves the schema graphically represented in Figure 2. The boxes are typed tables: engineer (ENG) is a specialization of employee (EMP) and department (DEPT) is referenced by employee. The goal of the runtime application of MIDST is to obtain a relational database for this, such as the one that involves the following tables:[3]

EMP (EMP_OID, lastname, DEPT_OID)
DEPT (DEPT_OID, name, address)
ENG (ENG_OID, school, EMP_OID)

Given the schema in Figure 2, our tool first imports it in its dictionary. Then, given the specification of the target model (the relational one), it selects an appropriate schema-level translation that is a sequence of basic translations, each of which is specified by means of a Datalog program. In this case, the schema-level translation should perform the following tasks: it first eliminates the generalizations (in the example, the one between ENG and EMP) and then transforms the typed tables (all tables in the source) into value-based tables.

In MIDST this would be done in four steps, with a first Datalog program for the elimination of generalizations, then two auxiliary ones, for the introduction of keys and the replacement of references with foreign keys, and a final one for the transformation of typed tables into value-based ones. The major task of our new version of the tool is the generation of a set of view statements for each of these Datalog programs.

The following is a sketch of a view definition generated in the first step.

```
CREATE VIEW ENG_A ...
AS (SELECT ... SCHOOL, ... EMP_OID
    FROM ENG );
```

[2] This is just a possible version of the OR model, and our tool can handle many others.
[3] As it is well known, there are various ways to map generalizations to tables, and here we use one of them.

It extends ENG (denoted as ENG_A to distinguish the new version from the original one) with a supplementary attribute, EMP_OID. It implements a strategy for the elimination of generalizations, where both the parent and child typed tables are kept, with a reference from the child to the parent. Details are omitted for the sake of space in this extended abstract, but they are shown in the full paper.

As it should have been clarified by the example, the core goal of the procedure is to generate executable statements defining views. This is obtained by means of an analysis of the Datalog schema rules (the ones used here to implement translations). The analysis gives the system-generic statement as an output.

The key idea of the procedure is a classification of MIDST metaconstructs according to the role they play. There are three categories: *container-, content-* and *support-constructs*. Containers are the constructs that correspond to sets of structured objects in the operational system (i.e. Aggregation and Abstract corresponding to tables and typed tables respectively). Content constructs represent elements of more complex constructs, such as columns, attributes or references: usually a field of a record (i.e. Lexical and AbstractAttribute) in the operational system. Support constructs do not refer to data-memorizing structures in the system, but are used to model relationships and constraints between them in a model-independent way. Examples are Generalizations (used to model hierarchies) and ForeignKeys (used to specify referential integrity constraints).

Our Datalog translation rules, in turn, can be classified according to the construct their predicate refers to. Therefore we have *container-*, *content-* and *support-generating* rules.

Exploiting the above observations, the procedure defines a view for each container construct, with fields that derive from the corresponding content constructs. Instead, as support constructs do not store data, they are are not used to generate view elements (while they are kept in the schemas). More precisely, given a Datalog schema rule $H \leftarrow B$, if H refers to a container construct, we will generate one view for each record matching the body of the rule. If H refers to a content construct, we generate a field of the view for the container construct this content belongs to.

This procedure does not depend on the specific constructs nor on the operational system or language. It is not related to constructs because we only rely on the concepts of container and content to generate statements. Other constructs may be added to MIDST supermodel without affecting the procedure: it would be sufficient to classify them according to the role they play (container, content, support). Moreover, it is not related to the operational system constructs or languages since the statements are designed as system-generic. A specification step, exploiting the information coming from the information schema of the operational system, will be then needed to generate system-specific statements. Furthermore, this approach is extensible because we might also consider adding *annotations* to functors whenever conditions get more complex and in order to handle specific cases.

The procedure is not bound to a single language and the generation of statements could involve the integration of several dialects fetching data from heterogeneous sources. This would not increase the complexity of the analysis nor the system-generic statements.

3 Discussion and related work

The problem of translating schemas between models has a largely recognized significance and has been pursued in the literature according to several perspectives of model management. Bernstein and Melnik [8] present the current state of the art in this field and, indirectly, outline an overview of the major achievements.

The approach towards runtime translation illustrated in this paper is based on MIDST [3–5], a platform allowing for model-independent schema and data translation. Its theoretical basis are laid in [3, 5–7] and provide a framework to perform model-independent schema and data translation. In this paper we provide the framework with a runtime design and go beyond the limitations expressed in [8, Sec.3.1].

The problem of translating schemas between models has been pursued by various other authors, including [11, 13], with approaches that are either focused on the schema level or on abstract models and languages.

Mork et al. [15] also adopt a runtime approach (based on [3, 7]) to solve the specific problem of deriving a relational schema from an extended entity-relationship model. They use a rule-driven approach and write transformations that are then translated into the native mapping language. However, although they face many issues such as schema update propagation and inheritance, indeed they solve a specific subset of problems and provide an object-relational mapping tool similar to [12]. Bernstein et al. [9] adopt a runtime approach to allow a developer to interact with XML or relational data in an object-oriented fashion. On the one hand their perspective is different since they only deal with a specific kind of heterogeneity; in addition they address the problem by translating the queries while we aim at generating views on which the original queries can be directly applied.

Our approach shares some analogies with Clio [10, 14, 17] too. Its aim is building a completely defined mapping between two schemas, given a set of user-defined correspondences. As for our translations, these mappings could be translated into directly executable SQL, XQuery or XSLT transformations. However, in the perspective of adopting Clio in order to exchange data between two heterogeneous schemas, the needed mappings should be defined manually; moreover, there is no kind of model-awareness in Clio, which operates on a generalized nested relational model.

The presented runtime extension of MIDST is a significant step with respect to the process of turning the platform into a complete model management system [1]. In such a perspective, Datalog rules are not only seen as model-to-model translations, but encode more general transformations that implement schema evolution and model management operators. Therefore the possibility of apply-

ing translation, hence operators, at runtime allows for the runtime solution to model management problems with model-independent approaches like the ones illustrated in [2].

4 Conclusions

The main contribution of this paper is a runtime version of MIDST. We have showed how we can generate executable statements out of translation rules. The approach aims at being general, in the sense that the final objective is to derive an executable statement for any possible translation.

References

1. P. Atzeni, L. Bellomarini, F. Bugiotti, and G. Gianforme. From schema and model translation to a model management system. In *BNCOD*, pages 227–240, 2008.
2. P. Atzeni, L. Bellomarini, F. Bugiotti, and G. Gianforme. A platform for model-independent solutions to model management problems (extended abstract). In *SEBD*, pages 310–317, 2008.
3. P. Atzeni, P. Cappellari, and P. A. Bernstein. Model-independent schema and data translation. In *EDBT Conference, LNCS 3896*, pages 368–385. Springer, 2006.
4. P. Atzeni, P. Cappellari, and G. Gianforme. MIDST: model independent schema and data translation. In *SIGMOD Conference*, pages 1134–1136. ACM, 2007.
5. P. Atzeni, P. Cappellari, R. Torlone, P. Bernstein, and G. Gianforme. Model-independent schema translation. *VLDB Journal*, 17:1347–1370, 2008.
6. P. Atzeni, G. Gianforme, and P. Cappellari. Reasoning on data models in schema translation. In *FOIKS Symposium, LNCS 4932*, pages 158–177. Springer, 2008.
7. 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.
8. P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.
9. P. A. Bernstein, S. Melnik, and J. F. Terwilliger. Language-integrated querying of xml data in sql server. In *VLDB*, pages 1396–1399, 2008.
10. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 805–810. ACM, 2005.
11. J.-L. Hainaut. The transformational approach to database engineering. In *GTTSE, LNCS 4143*, pages 95–143. Springer, 2006.
12. Hibernate. http://www.hibernate.org/.
13. P. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *CAiSE Conference, LNCS 1626*, pages 333–348, 1999.
14. R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *VLDB*, pages 77–88, 2000.
15. 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.
16. P. Papotti and R. Torlone. Heterogeneous data translation through XML conversion. *J. Web Eng.*, 4(3):189–204, 2005.
17. Y. Velegrakis, R. J. Miller, and L. Popa. Mapping adaptation under evolving schemas. In *VLDB*, pages 584–595, 2003.