Automatic Techniques for Data Model Translation

Paolo Papotti and Riccardo Torlone

Dipartimento di Informatica e Automazione
Università degli studi Roma Tre
Via della Vasca Navale 79
Roma, Italy
{papotti,torlone}@dia.uniroma3.it

Abstract

Distributed information systems deal with different data models and transformations between them are required. In this paper, we present a framework for the automatic translation of data between heterogeneous representations. Translations operate over formal representations of schemes and data and rely on a uniform description of models that we call metamodel. Complex translations of schemes and instances are derived automatically by combining a number of predefined basic transformations, which are implemented by means of query languages.

1 Introduction

Interoperability between different organizations is made difficult by the heterogeneity of the information systems involved. These differences rely on two main problems: data are stored under different formats in autonomous heterogeneous sources (XML documents, relational databases, objects), and information is represented in complex application by models that can differ significantly (UML, OO classes, E-R diagrams, web models).

There are already several ad-hoc solutions for pairs of models (e.g., from XML models to the relational one and vice versa). It is obvious however that the number of needed translation can largely increase with the number of distinct models involved. The problem becomes even more time consuming when we consider also variants (e.g. of the E-R model), versions (e.g. of the UML model) and subsets (e.g. of the XML Schema model).

What keep together the information through the developing cycle of a complex application are not the data models involved, but the concepts. Our approach relies on this intuition: most of the constructs of different models represent the same abstraction principle [8]. We believe that this simple observation provides a powerful tool for the management of schemes and data in a model independent way.

Solutions for a more general management of application artifacts have been recently proposed [3]. They are based on a high level approach, called *model management*, that offers a higher level programming interface than traditional techniques. The main abstractions are schemes and mappings between them that are used with high-level operators as Match, Merge, Diff, Compose, and ModelGen.

The goal of our research is the development of a really usable tool with three main objectives. (i) The management of information described according to a large variety of formats and models not fixed *a priori*. (ii) The automatic translation of schemes from one model to another, which can be seen as an implementation of the ModelGen operator. (iii) The automatic translation of data instances between models.

The system relies on a notion of *metamodel*, which has been inspired by an earlier work in the context of the management of multiple models in a database design tool [2]. A metamodel is a formalism that allows the uniform representation of models and the identification of differences between primitives used in the various models. Translations between models are automatically derived as follows. Source data is first serialized and represented in an XML formalism. Then, XML data is restructured to conform to the constructs allowed in the target model. Finally, it is deserialized into the specific syntax of the target. The restructuring operation is the more involved step and is performed by combining a number of predefined basic functions expressed in XML query languages.

The rest of the paper is organized as follows. Re-

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

lated work is reviewed in Section 2. In Section 3 we provide a general overview of our approach to model translation. In Section 4 we present the tool we are developing and the underlying algorithm. Finally, in Section 5, we sketch some conclusions and discuss open issues of research.

2 Related work

The problem of model translation is one of the issues that may arise when there is the necessity to combine heterogeneous sources of information in a coordinated and unified way. This include data integration [9], schema matching [18], schema merging [17], and database federation. Recently, Bernstein set the various problems within a very general framework that he called *model management* [3]. In this framework a set of basic operators has been identified: *Match*, *Compose* [11], *Merge*, *Diff*, and the *ModelGen* operator. They takes as input a scheme in a source model and returns the translation of the scheme into a target model with the appropriate mapping. Our proposal fits in this context since it actually refers to an extension of ModelGen in that we also translate schema instances from a model to another. There are several works on model management, although the majority of these approaches concentrate only on specific operators. Rondo is a rather complete framework for Model Management [12], but it does not address the problem of model translation. Cupid [10] and Clio [16] focus on the Match operator, whereas Merge has been studied in [17].

There are just few attempts to implement the *ModelGen* operator. Uni-Level Description [4] is a framework to represent data models that manages schemes and data within a representation inspired by the metamodel proposed in [2]. This system translates schemes and data between specific data models, making use of transformation rules based on Datalog, but, differently from our approach, these translations have to be defined for each pair of models (e.g., from the E-R model to the relational one). In our system, we relieve the user from the task of manually specifying such translation for each pair of model. An implementation of model management operators based on graph grammars, where schemes and mappings are represented by graphs, has been proposed in [20]. They investigate translation of schemes according to the ModelGen operator, but their approach requires a mapping between the source schema and the target as an input, rather than generating it as output of the translation. In the work presented by Atzeni *et al* [1], automatic scheme translation is reported, but they do not provide solutions for the restructuring of the instance.

Actually, there have been some attempts to set the problem in a general framework (for examples, see [5]). The main difference between our approach and these works relies on our notion of metamodel that introduces a higher level of abstraction with two main benefits: it provides a natural way to describe heterogeneous models and it allows us to identify generic transformations between primitives, independent of the specific models involved in a translation.

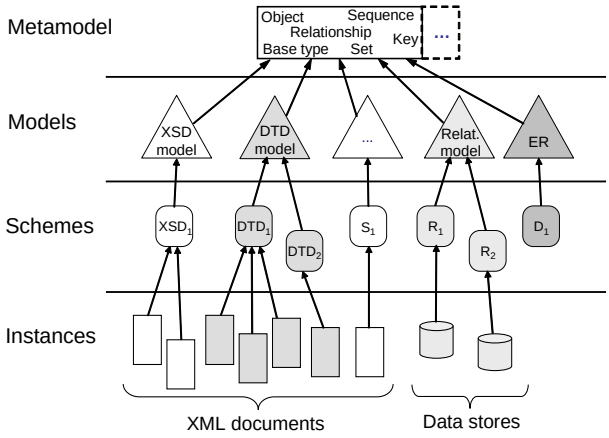

Figure 1: The Metamodel

3 The Approach

3.1 Metamodel

In our framework we identify four levels of abstractions, as reported in Figure 1. At the bottom level we have actual *data* (or *instances*) organized according to a variety of formats (relational tables, XML documents, HTML files, scientific data, and so on). At the second level we have *schemes*, which describe the structure of the instances (a relational schema, a DTD scheme, an XML Schema file, an E-R or UML diagram, etc.). Then, we have different formalisms for the description of schemes that we call *models* hereinafter (e.g., the relational model, the XML Schema model and conceptual models like the E-R model)[1]. Finally, we use the term *metamodel* to denote a general formalism for the definition of the various models.

Our *metamodel* is made of a set of *metaprimitives*. Each metaprimitive captures a class of constructs, used in different data models, which share common characteristics or, more precisely, that implement, possibly with different names, the same basic abstraction principle [2]. Examples of metaprimitives are: element, relation, attribute, base domain, relationship, set, (ordered) list, generalization, disjoint union, key, foreign key, and so on. In this framework, a model is defined as a set of *primitives*, each of which is classified according to a metaprimitive of the metamodel. For

[1]Note that we refer here to a "database" terminology that is different from the one used by Bernstein [3] and by OMG [13]: they actually use the term *model* to denote our notion of scheme, *metamodel* to denote our notion of model, and *metametamodel* to denote our notion of metamodel.

instance, the relational model offers the *table* primitive which is an instance of the metaprimitive *relation* over basic domains. Every model is described with its own primitives: the XML Schema model is defined with a set of constructs like *Complex Element, Atomic Element, Attribute, Sequence, All, Choice, Base Type* and so on. Each primitive has to be associated to the corresponding metaprimitive: e.g., the *all* construct is an instance of the *unordered sequence* metaprimitive. As we said above, the set of models is not fixed *a priori*: a user can define a new model and its translations from/to the other defined models are derived by the system with limited or none user intervention.

3.2 The Translation Process

A *translation* is defined as follows. Given as input two models M_s (the *source model*) and M_t (the *target model*), a scheme S_s (the *source scheme*) for M_s and a set of data D_s (the *data source*) for S_s; the translation of S_s and D_s into M_t returns a set of data D_t (the *data target*) of a scheme S_t (the *target scheme*) for M_t containing the same information as D_s.

In our approach, the translation of D_s and S_s into M_t relies on an instance M^* of the metamodel, that is used as a *pivot* element in the translation. Schemes are expressed in an XML-based syntax that makes use of the metaprimitives as tags. The translation process is composed by the following steps, as illustrated in Figure 2, which describes a complete translation of a scheme and an instance from the relational model to ODL.

1. A plain XML conversion that preserves the original structure is performed on both D_s and S_s, this task is usually supported by the source system in which D_s is stored. The output is a set $\widehat{D}_s$ of XML data and an XML description $\widehat{S}_s$ of the source scheme (e.g., an XML Schema).

2. The scheme $\widehat{S}_s$ is translated into a representation in terms of metamodel primitives, that we call $\widehat{S}^*_s$. The rational under this step is that in this way $\widehat{S}^*_s$ can be matched against the target model.

3. The scheme $\widehat{S}^*_s$ is restructured by translating metaprimitives used in the source scheme that are not allowed in the target model. The output of this operation is a scheme $\widehat{S}^*_t$ that makes use only of constructs allowed in the target model. Accordingly, the data set $\widehat{D}_s$ is translated into a format $\widehat{D}_t$ that is coherent with $\widehat{S}^*_t$. Note in Figure 2 that the *Relation* construct has been replaced by a *Class* construct and, for each element of the data set, an *oid* (object identifier) has been created, as required in an object-oriented model.

4. The scheme $\widehat{S}^*_t$ is translated into a scheme $\widehat{S}_t$ using the syntax of the target model M_t and finally, both $\widehat{S}_t$ and the data set $\widehat{D}_t$ are deserialized and delivered to the target system.

3.3 Scheme and data restructuring

The third step is the crucial point of the translation sequence. The other steps can be supported by external systems and are not always needed when source and/or target are already represented in XML. Since each primitive of the metamodel represents a class of similar constructs from different models, we can apply "generic" transformations that are independent of the particular translation at hand. It follows that, as the number of metaprimitives is limited, it is possible to predefine a number of basic transformations that can be composed to build complex translations. These basic transformations implement rather standard translations between primitives (e.g., from a generalization to a relationship or from a class to a table). Representatives of such procedures have been presented in [14].

Each of these transformations has indeed two components: a schema-level function f_S, which performs translations of metaprimitives, and a function f_I, which operates at instance-level by transforming actual data according to the translations operated by f_S. Specifically, these functions must satisfy the following consistency property. We say that a basic transformation $t[f_S, f_I]$ is *consistent* when, for each scheme S and for each instance I of S, $f_I(I)$ is an instance of $f_S(S)$. One of the main goals of our work is indeed to guarantee this property within the framework. We are exploring the property of consistency with respect to the requirements of the data exchange problem, as defined in [6, 16].

3.4 Properties of translations

A relevant issue related to model translation is the analysis of the *quality* of a translation. Clearly, the basic condition is the *correctness*, which requires that: (i) the output scheme be a valid scheme for the target model, and (ii) the output instance be a valid instance for the target scheme. The property is checked by the translation algorithm, on the basis of the consistency of the basic transformations as defined above. Actually, several other properties characterizing the efficiency and/or the effectiveness of a translation can be defined and this has been largely debated in the literature (see for instance [2]). We focus our attention on two properties that, we believe, are able to characterize in our framework "good" translations with a reasonable degree of confidence.

- The *minimality*, which requires that the number of basic transformations for a translation has to been be minimized.

- The *recall*, which requires that the result of a translation makes use of the maximum possible

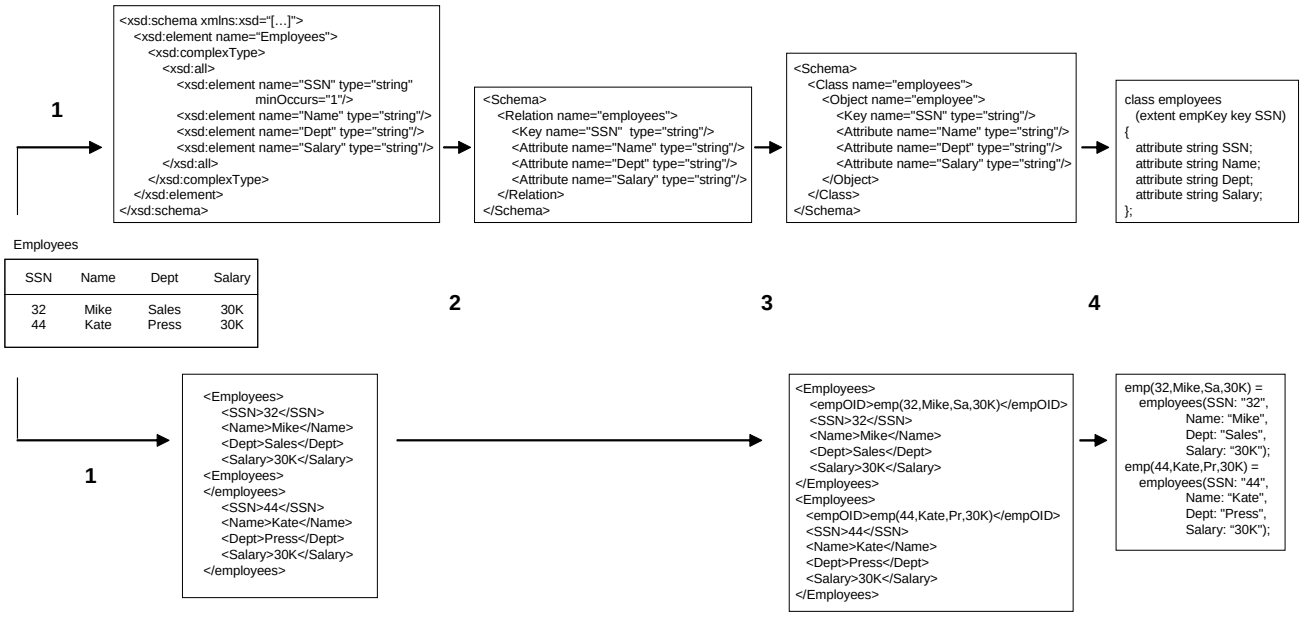

Figure 2: A complete translation from the relational model to ODL

number of primitives of the target model. The rationale under this choice, is that a translation is preferable than another if it is able to take more advantage of the *expressiveness* of the target model in terms of the primitives it offers.

4 Implementation

To evaluate the effectiveness of our approach, we have implemented a prototype of a system for model translation and we have used it in a variety of real application scenarios. The tool manipulates XML representations of models with DOM and performs translations by means of XML query languages (XQuery and XSLT) over materialized temporary results.

The system has a modular architecture whose heart is a *search engine* that implements an algorithm for the automatic composition of basic transformations. These are represented by a *signature* $t[P_{in} : P_{out}]$, that is, an abstract description of the set of metaprimitives P_{in} on which t operates and of the set of metaprimitives P_{out} introduced by t. For instance, to translate the *Relation* metaprimitive we can have different basic transformations: the first one translate it to a *ComplexElement* with *AtomicElement*, while the second returns a *ComplexElement* with *Attribute*. Their behaviors are represented by these signatures:

$$t[\{Relation\} : \{ComplexElement, AtomicElement\}]$$

$$t[\{Relation\} : \{ComplexElement, Attribute\}]$$

The signature is used by the algorithm to select the appropriate procedure to apply, without actually executing it, and it makes the algorithm independent of the particular implementation of the various transformations.

The search engine exhaustively composes basic transformations to automatically infer a valid translation of the metaprimitives of the input scheme that are not allowed in the target model. It turns out that even for a simple source scheme (e.g. a DTD with a few constructs) there are a lot of possible solutions and can be time consuming to generate all of them. For instance, the valid translations from an XML scheme into ODL are in order of hundreds. To limit the expansion of the search tree we take advantage of the A^* strategy, which avoid the expansion of paths of the tree that are already expensive and return as first result the best solution with respect to an appropriate function that *estimates* the total cost of a solution. We have defined a cost function as a linear combination of two functions: the first one measures the length of the actual translation, while the second one is related to the number of primitives allowed in the target model that are used in the actual target scheme. The weights associated with the two function are chosen to privilege the minimality of the recall of the translation.

We have conducted a series of experiments on the inference of the translation for schemes that vary in terms of dimension and complexity [15]. To evaluate the quality of the translations generated by the tool we used the two properties defined above. We first ran a test aimed at finding all the possibile solutions. We realized that there are two different class of results: those with higher minimality, and the one with best recall. The intersection of these sets contains what we have called the *optimal solution*. After this step, we ran the same translations with the A^* techniques. The

results of these experiments are encouraging: it turns out that the quality of the solution based on the A^* strategy is usually comparable to the optimal one. In particular, it is optimal for translations between XML based models, whereas in the other cases one of the value of the two property is equal to the best value.

5 Conclusion and future works

In this paper, we have presented an approach to the translation of schemes and data between heterogeneous models. Models, schemes, and instances are represented in XML; which allows a natural and flexible description of information at the different levels of abstraction. Moreover, XML is widely accepted as the standard for data exchange and data management tools usually provide import/export capabilities. Translations between models are derived automatically by combining a number of predefined basic transformations. The overall approach relies on a uniform description of models that we call metamodel.

To evaluate the effectiveness of our approach we are developing a system and we are conducting practical experiments that refer to real world scenarios within a first set of models (DTD, XML Schema, relational model, E-R, ODL). Currently, the tool is able to translate schemes and data between several data models and we are extending it with other formalisms and models for the Web and for scientific data (like genomic and protein sequences).

We plan to further investigate the qualities of the generated translations. In particular, a significant aspect within our framework is the *completeness* of the basic transformations set. Intuitively, a set of basic transformations is complete, with respect to a set of models, if it is possible to find a correct translation for any pair of models. As we said in Section 3.3 we are also exploring the consistency of the transformation and how to guarantee it. Finally, we plan to evaluate other cost functions that take into account the practical impact of the translations of instance on the performance of the system.

References

[1] P. Atzeni, P. Cappellari, and P. A. Bernstein. Modelgen: Model independent schema translation. In *ICDE*, pages 1111–1112, 2005.

[2] P. Atzeni and R. Torlone. Management of Multiple Models in an Extensible Database Design Tool. In *EDBT*, pages 79–95, 1996.

[3] P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *CIDR*, pages 209–220, 2003.

[4] S. Bowers, L. Delcambre. The Uni-level Description: A Uniform Framework for Representing Information in Multiple Data Models. In *ER*, pages 45–58, 2003.

[5] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your Mediators Need Data Conversion! In *ACM SIGMOD*, pages 177–188, 1998.

[6] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. In *PODS*, pages 90–101, 2003.

[7] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3): 27–34, 1999.

[8] R.B. Hull and R. King. Semantic database modelling: survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[9] M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, pages 233–246, 2002.

[10] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *VLDB*, pages 49–58, 2001.

[11] J. Madhavan and A. Y. Halevy. Composing Mappings Among Data Sources. In *VLDB*, pages 572–583, 2003.

[12] S. Melnik, E. Rahm, P. A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *SIGMOD*, pages 193–204, 2003.

[13] OMG Model Driven Architecture. Internet document, 2001. http://www.omg.org/mda/.

[14] P. Papotti and R. Torlone. Heterogeneous Data Translation through XML Conversion. *Journal of Web Engineering*, to appear, 2005.

[15] P. Papotti and R. Torlone. A Methodology for the Automatic Generation of Web Data Translations. *Submitted for publication.*

[16] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Herndez, and R. Fagin. Translating Web Data. In *VLDB*, pages 598–609, 2002.

[17] R. Pottinger and P. A. Bernstein. Merging Models Based on Given Correspondences. In *VLDB*, pages 826–873, 2003.

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

[19] E. A. Rundensteiner (editor). Special Issue on Data Transformations. *IEEE Data Eng. Bull.*, 22(1), 1999.

[20] G. L. Song, K. Zhang, J.Kong. Model Management Through Graph Transformation. In *VL/HCC*, pages 75–82, 2004.