MIDST: Model Independent Schema and Data Translation

Paolo Atzeni, Paolo Cappellari, and Giorgio Gianforme*
Dipartimento di Informatica e Automazione
Universita Roma Tre
Via della Vasca Navale, 79
00146 Roma, Italy
atzeni@dia.uniroma3.it, cappellari@dia.uniroma3.it, giorgio.gianforme@gmail.com

ABSTRACT

MIDST is a tool for the translation of schemas and databases from a model to another, in a framework that is flexible and extensible with respect to the family of models. It is based on a metamodel approach, with models described in terms of the constructs they involve, taken from a given set of predefined ones. Translations are obtained as compositions of elementary translations that refer to the individual constructs. The major novelties with respect to existing proposals consist in the generation of data-level translations and on the customizability of translations (at both schema and data level).

Categories and Subject Descriptors

H.2.7 [**Database administration**]: Data dictionary/directory; H.2.5 [**Heterogeneous databases**]: Data translation; H.2.1 [**Logical design**]: Data models

General Terms

Design, Documentation

Keywords

Model management, schema translation, data translation

1. INTRODUCTION

We demonstrate a prototype that handles models, within a large and extensible universe, and translations of schemas and data from a model to another. It is a framework that implements, in an extended version, the ModelGen operator proposed by Bernstein [5]. Indeed, in the original definition, ModelGen refers to the "schema level": given two data models M_1 and M_2 and a schema S_1 of M_1 the operator generates a schema S_2 of M_2. Our tool also considers the "data level": if a database D_1 over S_1 is given, it generates a corresponding database D_2 over S_2. In this context, it is common to refer to M_1, S_1, and D_1 as the *source* model, schema, and database, respectively, and to M_2, S_2, and D_2 as the *target* ones.

The tool demonstrates the results illustrated in a recent paper (Atzeni et al. [3]), which is in turn based on the MDM proposal [4] (which considered only the schema level, and with a number of significant differences). Let us briefly recall the main points. The basic idea is that a large family of models can be defined in terms of a rather small set of (predefined but extensible) *metaconstructs* (which abstract features of similar constructs in different models). Specifically, a model is defined by specifying which metaconstructs its constructs refer to. For example, the relational model involves (i) aggregations of lexicals (the tables), with the indication, for each component (a column), of whether it is part of the key or whether nulls are allowed; (ii) foreign keys defined over components of aggregations. As each construct can have variations (just as an example, within ER models, relationships can be binary or n-ary), even with few constructs the number of models can be large, and so translations cannot be defined individually for each pair of models; however, it is possibile to build them by composing elementary transformations, which refer to specific features. A library of elementary transformations can be predefined and complex translations could be generated out of them, on the basis of model descriptions. As elementary transformations can refer to constructs common to various models, they are often reused.

Additional important features of the tool (which are not described here, for the sake of space, but will be demonstrated) are (i) the use of a relational dictionary, which is visible and generated out of a very small core [1]; (ii) the use of Datalog rules with OID-invention, which are independent of the engine that executes them; (iii) the use of Skolem functions for OID-invention, which is not new in itself, but it is original for the management of mappings between elements (at both the schema and the data level), because of their materialization in the relational dictionary.

We have experimented (and will demonstrate) the approach with a family of models that includes relational, object-relational, object-oriented (from a structural point of view), ER, XSD, each in a significant number of variants.

The prototype presented here is different from others recently presented. The work by Atzeni et al. [2] considered only system generated translations of schemas, and so there are two major novelties here: data-level translations (automatically generated from those at the schema level) and customization of translations, which can be tailored to specific application features. The work by Bernstein et al. [6] focuses on some specific translations, specifically flexible mapping of inheritance hierarchies and the incremental regeneration of mappings after the source schema is modified. It is not based on a relational dictionary.

*Supported by a Microsoft PhD Fellowship.

SIGMOD'07, June 12–14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

2. DEMONSTRATION

In this section we briefly illustrate the major features of the tool, with the description of work sessions of a user. The tool has use cases at various levels:

1. Translations between existing models
2. Definition of new models, using the available metaconstructs
3. Extension of the metamodel with the definition of new metaconstructs

We mainly refer to the first one, where the contributions of this proposal appear, with the data level and the customization of translations. (The other use cases have been discussed, at the schema level, in previous pieces of work [2]). Therefore, the demo will have the models of interest already defined, with a suitable library of translations, and the tool will be used to translate a schema and a database from the source to the target model. Also, for each of the models, the tool has import/export modules, which allow the exchange of schemas and databases with external systems. It is worth noting that while we have the possibility of defining many different models, on the basis of variations and restrictions of constructs, we really need just a few import/export modules, essentially one for each "family" of models: for example, one module for all variants of the relational and object-relational model, one module for all variants of XSD-based models: the idea is that a module refers to the most general model in the respective family, and includes tests to verify the membership of a schema to a more restricted model, if needed.

In order to show the features of our tool, we refer to a rather common translation task: the translation of an XML document into the relational model. We will show how the tool generates the translation and, even more important, how it handles variants of models. Suppose the user goal is to translate an XML document described by the XML schema (XSD) graphically shown in Figure 1(a) into a relational database. Now, in XSD, we have that there exist `key` elements, but they are not always used: in our terminology, we can say that there are two different models, one that uses them and one that does not. Let us initially assume that the source model does not use keys: that is, even if in practice different patients will have different *PatientNo*, this is not specified explicitly and so cannot be assumed in the translation process.

As a first step, the user imports the schema sketched in Figure 1(a) and an XML document (not shown for the sake of space, but valid with respect to the schema) into the tool using the import module. This requires that the tool contains the definition of the source model and the process, while importing both the schema and the data in the internal dictionary. It will check that the schema belongs to such a model and that the document is valid. The source model here is specified, in our metamodel, as having abstract objects (used to model top elements, e.g., *Patient*), with simple attributes (*PatientNo*, *Name*, ...) and nested ones (**Symptom* and **Contact*, where the star * means that there is a set of symptoms and contacts for each patient). In simple words, we could say this is a model for "nested objects with identity."

As we said in the introduction, translations are performed by composing elementary steps. In this case, the translation could be performed in two steps. The first step removes nested attributes, by introducing additional abstract objects, with references to the original containing object. The second step then translates the "object" model to a value based one. As we said in the introduction, the tool has a library of predefined translations, and algorithms to find the appropriate translation given a source and a target model (using ideas similar to those proposed in the MDM proposal [4]). Figures 1(b) and (c) show intuitive sketches of the intermediate and of the final schemas in this process. The translation into the relational model requires the introduction of a new identifying column for each table (*PatientID*, *SymptomID*, *ContactID*), as no key is defined in the source model.

The major novelty in this tool is that translations for the data level are generated out of those for schemas: that is, not only is the schema translated as shown in Figure 1, but also any source document valid for the source schema can be translated into the corresponding database for the target schema. Instance level rules are generated at schema translation time: when a user performs a schema translation, instance level translations are generated. In the sample translation case we have that the translation process produces a complex-object database for the schema in Figure 1(b) first, and a relational database for the schema in Figure 1(c) then (both in the internal representation). As the relational database has additional columns for keys, we need a technique to generate values for them, and the tool includes value-generating functions (essentially, Skolem functions that produce new values whenever needed). As the final step, the user has to export the resulting schema and database through the export feature. The technical details for the data level translation have been presented in the research paper that discussed the approach (Atzeni et al. [3]).

Experimentation has demonstrated that the generation process produces instance level translations that reflect user intents specified within the schema level translations. We have experimented translations with various data models, including various restrictions of XSD-based models and of relational and object-relational ones.

A comment is useful with respect to variants of models. In the translation from the object model to the (value-based) relational one, we have had to introduce new columns for the keys of tables. If instead we had had *key* constraints specified for the XSD elements in the source model, we could have used them as keys of tables (in the usual relational sense): in the example, we would have used *PatientNo*, without introducing *PatientId*; this confirms that it is important to handle details of models, as they can induce significant differences in the results of translations. In the demonstration, we will show the differences we obtain in the target database, depending on whether keys are specified or not in the schema of the source document.

Another novel feature offered by the tool is the possibility, for the user, to customize predefined rules in the repository. Rules in the repository describe a translation between constructs or are designed to perform specific tasks (such as the unnesting in the previous example). However, the application of these rules affects the whole schema: in the previous example, by applying the unnesting rule all the set elements have been unnested (two in the specific case). Now, let us suppose that the user intent is to modify the source schema in Figure 1(a) by unnesting only the *Symptom* element (thus

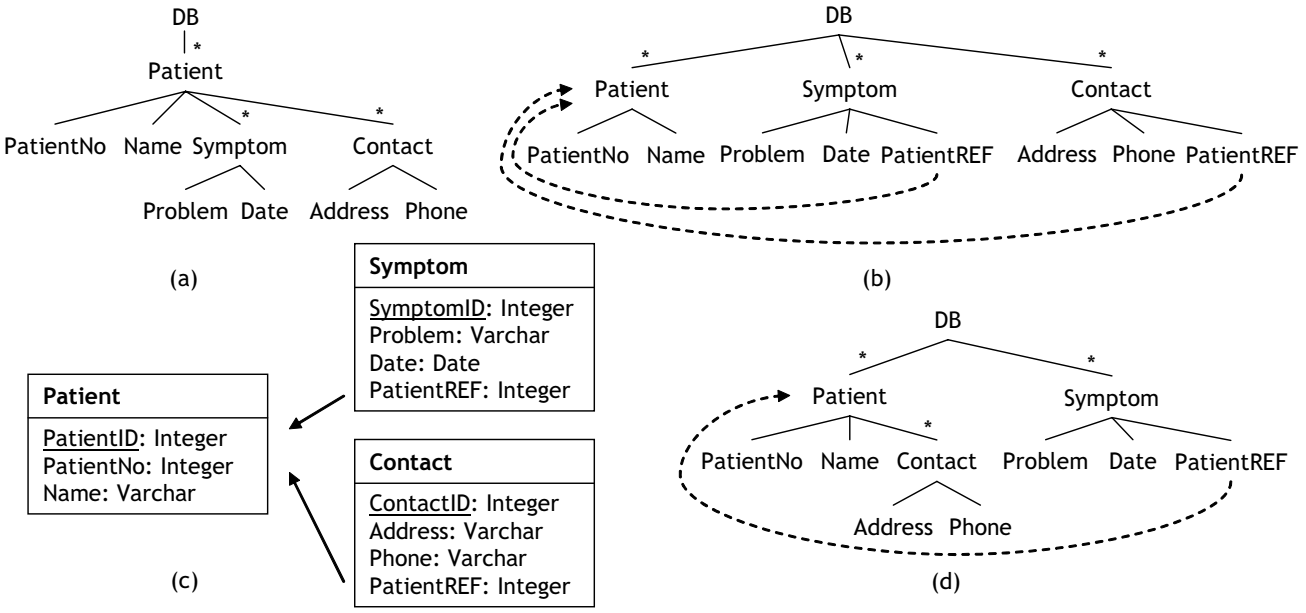

Figure 1: Databases Schemas

remaining in a complex-object world, with references, for example XSD with `key` and `keyref`). The target schema would in this case be that shown in Figure 1(d). Clearly, the unnest rule does this task, but it is too generic: the user has to add selection conditions to it. The user has to customize the rule by editing it and bounding the variable associated with the name of the unbounded complex element with the value "symptom." By adding this condition, the rule translates a subset of the unbounded complex elements in the schema. Specifically, the subtree rooted in *Contact* is not translated because it doesn't satisfy the selection condition. The tool is also equipped with an analysis engine that is able to detect untranslated portion of the source schema that, when needed, warns the user. Warning may occurs because of an incomplete customization or because the set of chosen translations doesn't cover all the constructs of the source model used in the source schema. Our experiments confirm that the instance rule generation process preserve customization on data. In the specific case, only the data instances of *Symptom* will be unnested.

Because of the declarative translation approach and the compactness of the rules, customization as well as maintenance tasks are simplified. Moreover, the generation of instance level translation reduces to zero the effort of dealing with instance level problems. Indeed, customization can be made at instance level too. For instance, in case a new attribute has been introduced in the schema, its values are not known to the system: the user has to tune the generated rule by defining a desired function that generates those values. Another case is when the user specifies the "merge" or "split" between two lexical attributes of a schema: she has to define the corresponding "merge" or "split" function. A third case is when the user wants to migrate a subset of the data in the database: she has to define selection rules to specify data of interest.

3. REFERENCES

[1] P. Atzeni, P. Cappellari, and P. A. Bernstein. A multilevel dictionary for model management. In *ER 2005, LNCS 3716*, pages 160–175. Springer, 2005.

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

[3] P. Atzeni, P. Cappellari, and P. A. Bernstein. Model-Independent Schema and Data Translation. In *EDBT 2006, LNCS 3896*, pages 368–385. Springer, 2006.

[4] P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. In *EDBT 1996, LNCS 1057*, pages 79–95. Springer, 1996.

[5] P. A. Bernstein. Applying model management to classical meta data problems. *CIDR*, pages 209–220, 2003.

[6] P. A. Bernstein, S. Melnik, and P. Mork. Interactive schema translation with instance-level mappings. In *VLDB*, pages 1283–1286, 2005.