ModelGen: Model Independent Schema Translation

Paolo Atzeni, Paolo Cappellari
Università Roma Tre
{atzeni,cappe}@dia.uniroma3.it

Philip A. Bernstein
Microsoft Research
philbe@microsoft.com

Abstract

A customizable and extensible tool is proposed to implement ModelGen, the model management operator that translates a schema from one model to another. A wide family of models is handled, by using a metamodel in which models can be succinctly and precisely described. The approach is novel because the tool exposes the dictionary that stores models, schemas, and the rules used to implement translations. In this way, the transformations can be customized and the tool can be easily extended.

1. Introduction

Model management is a high-level approach to solving meta data problems [3]. A major operator in model management is *ModelGen*: given a source data model M_1, a target data model M_2 , and a source schema S_1 expressed in M_1, ModelGen generates a target schema S_2 in M_2. All database designers implicitly use ModelGen when they translate a conceptual schema expressed, for example, in the ER model, into a corresponding relational schema. From here on, we use *model* to mean *data model*.

Like other model management operators, ModelGen should be *generic* (i.e., model-independent), so that it works for all data models of interest. An early approach was proposed by Atzeni and Torlone [1,2] who developed a tool to implement it based on a notion of metamodel.

A *metamodel* is a set of constructs that can be used to define models, which are instances of the metamodel. The approach is based on Hull and King's observation [4] that the constructs in most models can be expressed by a small set of generic *metaconstructs*. Each model is defined by its constructs and the metaconstructs they refer to.

The translation of a schema from one model to another is defined in terms of translations over the metaconstructs. A *supermodel* is defined as a model with constructs corresponding to all metaconstructs known to the system. Each model is a specialization of the supermodel, so a schema in any model is also a schema in the supermodel. A translation is performed by eliminating constructs not allowed in the target model, and possibly introducing new constructs. Translations are built from elementary transformations, each of which is essentially an elimination step.

The solution above is effective, but has the following limitations because the representation of the models and transformations are hidden within the tool's source code:

- Only the designers of the tool can extend the models.
- Correctness of the rules has to be accepted by users as a dogma. They can check it only by using the tool.
- To customize the transformations and the target schemas, the tool's source code must be modified.

We present a new tool that overcomes these limitations by exposing the dictionary and translations. This permits rapid development and maintenance of models and translations and reasoning about the correctness of translations

2. The dictionary and translation rules

The tool is based on a relational dictionary that stores the metadata of interest: the metamodel, models and schemas. It has four parts: (i) the MetaSuperModel, which describes the structure of the metaconstructs of interest, (ii) the SuperModel, which stores the schemas to be translated; (iii) the MetaModels, which describe the constructs of all models of interest, each construct associated with the supermodel metaconstruct it corresponds to, and (iv) the Models, which stores schemas of interest.

The translation process is a composition of basic transformations. For example, going from an n-ary ER model to the relational one, we can first eliminate n-ary relationships and then go from the binary ER model to the relational one. Each basic transformation (e.g., from binary ER to relational) is expressed by a set of rules written in a Datalog dialect with OID-invention based on Skolem functions [5]. This technique has several advantages:

- rules are independent of the main engine that interprets them, enabling rapid development of translations;
- the system itself can verify basic properties of sets of transformations (e.g., some form of correctness) by reasoning about the bodies and heads of Datalog rules;
- transformations can be easily customized. E.g., one can add "selection conditions" that specify the schema elements to which a transformation is applied.

Another benefit of using Skolem functions is that their values can be stored in the dictionary and used to represent the mappings from a source to a target schema. This is needed in more general scenarios for model management, e.g., as reverse engineering or schema evolution [3].

The translation process is based on the supermodel: (1) the source schema is translated into the supermodel (2) the translation to the target schema is executed within the supermodel; and (3) the target schema is translated into the target model. Steps (1) and (3) are laborious but straightforward, as each model is subsumed by the supermodel. The only transformations we hand-coded

were those in (2). We have a set of rules that perform basic translations over the available metaconstructs and can therefore be combined to form complex translations.

3. The architecture

The tool has the architecture shown in Figure 1. The core is composed of the dictionary, the transformation repository and the rule applicator. The basic management of the dictionary is performed by two generic modules: DefineModel, to define new models based on the available constructs, and DefineSchema, to define a schema for a chosen model. After a model is defined, the tool automatically creates the structures needed to handle its schemas.

The transformation repository contains two kinds of artifacts (as we saw in Section 2):

- Basic transformations (e.g., the transformation that produces a binary ER schema from an n-ary one)
- Datalog rules, which can be assembled into basic transformations and customized by adding further conditions to specify to which concepts the rule is applied.

Translations are specified by composing basic transformations. The tool can verify whether the translation process generates schemas in the target model and can detect redundancies in a sequence of basic transformations.

4. The tool demonstration

The tool offers functions for three categories of users:

1. A *designer* defines schemas in available models and asks ModelGen to translate them.
2. A *model engineer* defines new models by using the available metaconstructs.
3. A *metamodel engineer* adds new metaconstructs to the metamodel and defines translation rules for them, thereby extending the models handled by the system.

The above activities are done without touching the tool's source code. Let us illustrate a possible usage scenario. A *designer* defines schemas by choosing a model and then instantiating the associated constructs. For example we can define an ER_Entity Person and add ER_Attributes SSN and Name to it. Schema definition is handled by interactive interfaces and batch importers for XML formats produced by current design tools. A report function is available for schemas. After defining a schema, a designer can request its translation to any other model defined in the system.

A *model engineer* defines models. She defines the constructs allowed in a model, giving them names and adding the desired properties available in the metaconstructs. For example, she can choose AttributeOfAbstract, naming it ER_Attribute and inheriting all properties but isNullable (meaning that null values on an entity's attributes are not allowed). At the end, she picks a name and saves it. The system automatically creates the corresponding dictionary structure and "copy rules" to copy constructs to and from the supermodel.

A *metamodel engineer* defines new basic transformations by writing new Datalog rules and reusing existing ones. This may also require the definition of new Skolem functions. An important but rare task is defining new metaconstructs, which, to be useful, require the definition of suitable basic transformations involving them.

Acknowledgements. We thank Rachel Pottinger for many helpful discussions on this work and the paper.

References

1. P. Atzeni, R. Torlone: Management of Multiple Models in an Extensible Database Design Tool. EDBT 1996: 79-95.
2. P. Atzeni, R. Torlone: MDM: a Multiple-Data-Model Tool for the Management of Heterogeneous Database Schemes. SIGMOD 1997 (demo): 528-531.
3. P. A. Bernstein: Applying Model Management to Classical Meta Data Problems. CIDR 2003: 209-220.
4. R. Hull, R. King: Semantic Database Modeling: Survey, Applications, and Research Issues. ACM Computing Surveys 19(3): 201-260 (1987).
5. R. Hull, M. Yoshikawa: ILOG: Declarative Creation and Manipulation of Object Identifiers. VLDB 1990: 455-468.

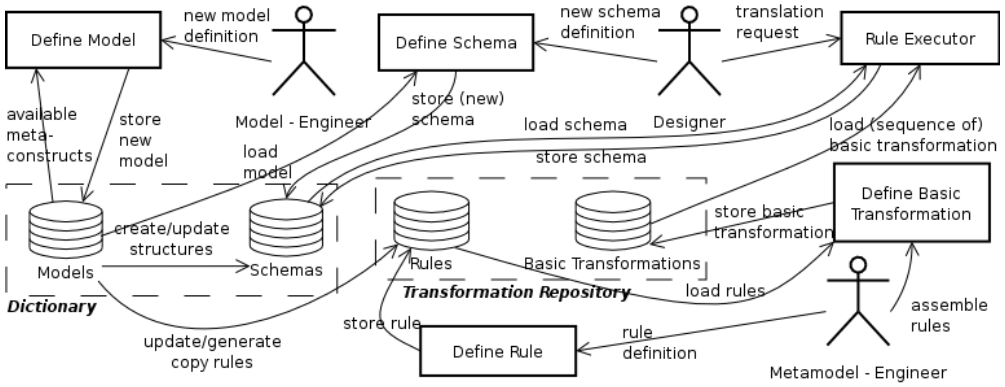

Figure 1: a sketch of the architecture of ModelGen