# Model Independent Schema and Data Translation (Extended abstract)

Paolo Atzeni(1), Paolo Cappellari(1), Philip A. Bernstein(2)

(1) Università Roma Tre, Italy
{atzeni, cappellari}@dia.uniroma3.it

(2) Microsoft Research, Redmond, WA, USA
philbe@microsoft.com

## 1 Introduction

Many applications involve the management of heterogeneous data and the need to translate actual data and their schemas from one framework to another. There are often different systems used to handle data, following different models, and there is the need to exchange data from one to another. Even variations of models are often enough to create difficulties: for example, the object-relational features of the various DBMSs almost never coincide.

Let us show an application scenario for this problem (very simple for the sake of space). Consider an OR database with the schema

EMPLOYEES(EmpNo,Name,Dept:*Departments)
DEPARTMENTS(Name,Address)

Each of the tables is assumed to have a system managed identifier, and the notation Dept:*Departments specifies that the Dept values are indeed references to tuples of DEPARTMENTS. A possible instance for the schema is shown in Fig. 1, where we also show for each tuple the (system managed) identifier. Now, if we want to translate this database into the relational model, we can follow the well known technique that replaces explicit references with key values. However, some of the details of this transformation depend upon the specific features we have in the source and target model. For example, in the object world, and even in OR databases, keys (or visible identifiers) are sometimes ignored. Then, if keys have

| | EMPLOYEES | | |
|---|---|---|---|
| | EmpNo | Name | Dept |
| E#1 | 134 | Smith | D#1 |
| E#2 | 201 | Jones | D#2 |
| E#3 | 255 | Black | D#1 |
| E#4 | 302 | Brown | NULL |

| | DEPARTMENTS | |
|---|---|---|
| | Name | Address |
| D#1 | A | 5, Pine St |
| D#2 | B | 10, Walnut St |

**Fig. 1.** A simple object-relational database

| Employees | | |
|---|---|---|
| EmpNo | Name | Dept |
| 134 | Smith | A |
| 201 | Jones | B |
| 255 | Black | A |
| 302 | Brown | NULL |

| Departments | |
|---|---|
| Name | Address |
| A | 5, Pine St |
| B | 10, Walnut St |

**Fig. 2.** A translation into the relational model

been specified on the OR tables, then a reasonable result would be that in Fig.2. If instead the keys have not been specified in the OR database (and we assume, as usual, that they are required in the relational model), then the most natural way to implement the translation involves the use of an additional attribute for each table, to be used as identifier (however, a visible one, as opposed to the system managed ones of the OR model).

These problems are often tackled by means of ad-hoc solutions, for example by writing a program for each specific application, but this is clearly not effective. Bernstein et al. [3, 4] have recently argued for generic solutions and proposed a high level approach, called *model management*, based on a set of operators to be applied to schemas. A specific operator in the family is *ModelGen*, which translates schemas from a source to a target model, exactly as we required above with respect to the schema level. We use the term *ExtendedModelGen* to refer to the extension of it that also translates database instances.

An early approach to ModelGen was MDM [1, 2] a tool to manage heterogeneous schemas based on a notion of *metamodel* with a set of constructs (the *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 then defined in terms of translations over the metaconstructs, in such a way that the same translation is used for a given metaconstruct in all the models where it appears. The approach is based on Hull and King's observation [5] that the constructs used in most known models can be expressed by a limited set of generic (i.e. model-independent) *metaconstructs*: lexical, abstract, aggregation, generalization, function. A major concept in the approach is the *supermodel*, a model that has constructs corresponding to all the metaconstructs known to the system. Thus, each model is a specialization of the supermodel, so a schema in any model is also a schema in the supermodel, apart from the specific names used for constructs.

The main limitation of MDM is the fact that it refers only to the schema level; also the representation of models and transformations is hidden within the tool's source code, and so any extension would be very complex, with respect to the management of instances and also to any customization need.

This paper proposes a framework for the development of an effective implementation of ExtendedModelGen based on a more transparent and flexible approach. In the rest of this extended abstract we illustrate the main features of our approach, the structure of our dictionary in Section 2 and the translation

| SM_Abstract | | |
|---|---|---|
| OID | sOID | Name |
| 101 | 1 | Employees |
| 102 | 1 | Departments |
| ... | ... | ... |

| SM_AttributeOfAbstract | | | | | | |
|---|---|---|---|---|---|---|
| OID | sOID | Name | IsKey | IsNullable | AbsOID | Type |
| 201 | 1 | EmpNo | T | F | 101 | Integer |
| 202 | 1 | Name | F | F | 101 | String |
| 203 | 1 | Name | T | F | 102 | String |
| ... | ... | ... | ... | ... | ... | ... |

| SM_RefAttributeOfAbstract | | | | | |
|---|---|---|---|---|---|
| OID | sOID | Name | IsNullable | AbsOID | AbsToOID |
| 301 | 1 | Dept | T | 101 | 102 |
| ... | ... | ... | ... | ... | ... |

**Fig. 3.** An object-relational schema represented in the dictionary

process and rules in Section 3; then in Section 4 we draw our conclusions, and briefly comment on the experiments we made and the models we used.

## 2 The dictionary

A major feature of our proposal is the structure of the dictionary used to represent both schemas and instances, by means of two distinct but highly correlated parts. The dictionary also has a meta level, which we neglect here for the sake of space, but can be used as a "core," to generate the structure of the other levels.

Schemas are described in the dictionary (which has a relational structure) by means of a table for each metaconstruct, with columns that specify the various properties of a construct and the other constructs it refers to. Going back to our example, we would have a number of tables to handle schemas in the OR model, including those shown in Fig.3. Let us comment on two of them as representatives:

– SM_Abstract describes the object tables (those with system-managed identifiers); for each of them it keeps the name and two identifiers that are used in all tables: OID, which identifies the construct, and sOID, to indicate the schema the construct belong to;
– SM_AttributeOfAbstract has information about the attributes of the two object tables: for each attribute we keep a reference to the abstract it belongs to (the AbsOID attribute) and two booleans that specify whether it belongs to the key and whether it allows null values.

The above tables describe schemas in terms of supermodel constructs, because translations, in order to be reusable, as we will see, refer to the supermodel. Clearly SM_Abstract would also contain tuples referring to other models, for example the ER model, where entities also correspond to abstracts.

Instances are described with structures similar to those for schemas, with some differences. A portion of the representation of the instance in Fig.1 is shown in Fig.4. Let us comment the main points:

– each table has an i-sOID (instance of SchemaOID) attribute, instead of the sOID attribute we had at the schema level;

| SM_InstOfAbstract | | |
|---|---|---|
| OID | i-sOID | AbsOID |
| 1001 | 1 | 101 |
| 1002 | 1 | 101 |
| ... | ... | ... |
| 1005 | 1 | 102 |
| ... | ... | ... |

| SM_InstOfAttributeOfAbstract | | | | |
|---|---|---|---|---|
| OID | i-sOID | AttOID | i-AbsOID | Value |
| 2001 | 1 | 201 | 1001 | 134 |
| 2002 | 1 | 202 | 1001 | Smith |
| 2003 | 1 | 201 | 1002 | 201 |
| 2004 | 1 | 202 | 1002 | Jones |
| ... | ... | ... | ... | ... |
| 2011 | 1 | 203 | 1005 | A |
| 2012 | 1 | 204 | 1005 | 5, Pine St |
| 2013 | 1 | 203 | 1006 | B |
| ... | ... | ... | ... | ... |

| SM_InstOfRefAttributeOfAbstract | | | | |
|---|---|---|---|---|
| OID | i-sOID | RefAttOID | i-AbsOID | i-AbsToOID |
| 3001 | 1 | 301 | 1001 | 1005 |
| 3002 | 1 | 301 | 1002 | 1006 |
| ... | ... | ... | ... | ... |

**Fig. 4.** Representation of an object relational instance

- each instance has a reference to the construct it instantiates; for example, the first table in the Fig.4 has an AbsOID column;
- "properties" of schema elements (e.g. IsKey, IsNullable) do not appear at the instance level;
- all identifiers (OID of the construct and the references to other constructs) that appear at the schema level are transformed into identifiers for instances; e.g., table SM_RefAttributeOfAbstract in Fig.3 includes columns (i) OID (the identifier for the row), (ii) AbsOID (the identifier of the abstract to which the attribute belongs), and (iii) AbsToOID (the identifier of the abstract to which the attributes "points"); in Fig.4 each of them is taken one level down: (i) each row is still identified by an OID column, but this is the identifier for the instance; (ii) each value of i-AbsOID indicates the instance of abstract the attribute is associated with (e.g., 1001 is the identifier for Smith); (iii) i-AbsToOID indicates the instance of abstract the attribute refers to;
- if the construct is lexical (that is, has an associated value [5]), then the table has a Value column; in Fig.4 SM_InstanceOfAttributeOfAbstract is the only lexical construct.

## 3 The translation process

The translation process from a source schema and instance (in a source model) to a target schema and instance (in a target model) is based on the supermodel, so extending the MDM approach (which would do this but only at the schema level):

1. the source schema and instance are "copied" to the supermodel; this is straightforward, as each model is subsumed by the supermodel;
2. the actual translation to the target schema and instance is performed within the supermodel;
3. the target schema and instance are copied into the target model; this is also a reasonably simple phase, for the same reasons as in 1 above.

Given the complete description of models in the dictionary and their correspondence to the supermodel, "copy transformations" that perform the "copy" tasks (phases 1 and 3) can be automatically generated.

Therefore, the only transformations that have to be explicitly specified in our approach are those within the supermodel. They are obtained as compositions of "basic transformations": for example, the translation we mentioned in the introduction, from an OR model (with keys) to a relational one, could be implemented by means of two steps: $B_1$ from the OR model to the binary ER model (transforming reference attributes into relationships[1]) and $B_2$ from the binary ER model to the relational one. If instead we wanted to use variations of these models, for example an OR model without keys, we could proceed with a preliminary step before $B_1$ and $B_2$, translating from the OR model without keys to the OR model with keys (adding "new" key attributes to objects and generating new values for them). In this way, basic steps are clearly reusable.

The basic transformations and the copy phases we discussed above are each specified by a set of Datalog rules, which consider the various constructs at hand. Rules are used both at the schema and at the instance level; let us concentrate first on the schema level and then on the instance level. Each rule generates elements of a schema for a given metaconstruct (the one that appears in the head of the rule), and we might have more rules for the same metaconstruct.

A basic transformation is composed of a set of Datalog rules; for example, the basic step for translating from the binary ER model to the relational model would include various rules, generating tables and columns from entities, relationships and attributes. Let us briefly comment the syntax by referring to one of them, the rule that generates columns for the attributes of many-to-many relationships:

```
SM_AttributeOfAggregationOfLexicals(
 OID:#attribute_4(attOid), sOID:target,
 Name:name, IsKey:FALSE, IsNullable:isN,
 AggOID:#aggregation_2(aggOid))  ⟵
SM_AttributeOfBinaryAggr...OfAbs...(
   OID:attOid, sOID:source, Name:name,
   IsNullable:isN, AggOID:aggOid),
SM_BinaryAggregationOfAbstracts(
   OID:aggOid, sOID:source,
   isFunct1:FALSE, isFunct2:FALSE)
```

The rule has two literals in the body, the first with the details for the attribute and the second specifying that the rule is applied only to many-to-many rela-

[1] As we will see shortly, rules refer to supermodel constructs, whereas we mention here model specific ones, for the sake of readability.

tionships (this is done by referring to the two properties isFunct1 and isFunct2, which specify the cardinality of the relationship). With respect to the head, there are two points to note. First, there is a constant value, FALSE, for IsKey: this is because the attribute generated here never belongs to a key, as it originates from an attribute of a relationship. Second, we use Skolem functors to generate new identifiers; in general, there are various functors associated with a given construct, as it may be generated from constructs of various types. Skolem functions are materialized in the dictionary: there is one table for the Skolem functions associated with each construct, in order to guarantee the disjointness of the functions. The tables are a representation of the mapping between constructs in the source schemas and those in the target schema, an issue that is considered very important in model management [3].

Rules for translating instances are derived from those at the schema level, on the basis of the close correspondence between the two levels in the dictionary, with only the need for some local refinement for specific issues (which we omit here for the sake of space). Let us consider a very simple rule in the translation from the binary ER model to the relational one, the one that generates columns of tables for attributes entities. At the schema level, is as follows:

SM_AttributeOfAggregationOfLexicals
(OID:#attribute_2(*attOid*), sOID:*target*,
Name:*name*, isKey:*isK*, isNullable:*isN*,
AggOID:*#aggregation_2*(*absOid*)) $\longleftarrow$
SM_AttributeOfAbstract (OID:*attOid*,
sOID:*source*, Name:*name*, isKey:*isK*,
isNullable:*isN*, AbsOID:*absOid*)

The corresponding rule at the instance level is the following:

SM_InstanceOfAttributeOfAggr...OfLex...
(OID:#i-Attribute_2(*i-AttOid*), i-sOID:*target*,
i-AggOID:#i-Aggregation_2(*i-AbsOid*),
AttOID:#attribute_2(*attOid*),
Value:*value*) ) $\longleftarrow$
SM_AttributeOfAbstract (OID:*attOid*,
sOID:*sourceSchema*, Name:*name*, isKey:*isK*,
isNullable:*isN*, AbstractOID:*absOid*),
SM_InstanceOfAttributeOfAbstract
(OID:*i-AttOid*, i-sOID:*source*,
i-AbsOID:*i-AbsOid*, AttOID:*attOid*, Value:*value*)

Let us note that

– the head is obtained from the head of the schema level rule by applying transformations that take schema literals "down to instances" (see below);
– the body is composed of two parts: a copy of the body at the schema level and its transformation down to instances.

The transformations "down to instances" derive from the correspondences in the dictionary, as follows

1. references to schemas become references to instances: sOIDis replaced by i-sOID;
2. Skolem functors are replaced by "homologous" functors at the instance level, by transforming both the function name and the arguments;
3. instances refer to the constructs they instantiate: see the AttOID field;
4. "properties" (e.g., Name and isKey) of schema elements do not appear at the instance level;
5. identifiers at the schema level become identifiers at the instance level;
6. lexical constructs have a Value attribute.

## 4 Conclusions

In order to verify the ideas illustrated in this paper, we have implemented a prototype tool. We have used a metamodel with about twenty different constructs, each with a number of properties, so the number of different models we can define is huge. For our experiments, we have defined a set of significant models, each in various versions (including the possibility of having or not having nested attributes and generalization hierarchies): ER, UML class diagrams, OR, OO, and relational. At the instance level, we experimented with the models that handle data, hence OR, OO, and relational (in the nested and flat version). We have developed so far about fifteen basic translations, each involving five to ten Datalog rules, and have obviously generated automatically all copy rules. The experiments we conducted have confirmed that translations can be effectively performed with our approach. The tool has been effective, because of the easiness in the generation of the dictionary and of its visibility and for the declarativeness of the Datalog rules, independent of the engine that executes them.

Future work will concern additional experimentation, especially with complex nested models and the development of features that reason on properties of basic transformations and of complex translations; the declarative specification of rules in Datalog will be used as the basis for an automatic generation of descriptions for basic transformations.

## References

1. P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. *EDBT 1996*, 79–95.
2. P. Atzeni and R. Torlone. Mdm: a multiple-data-model tool for the management of heterogeneous database schemes. *SIGMOD 1997*, 291–301.
3. P. A. Bernstein. Applying model management to classical meta data problems. *CIDR'03*, 209–220.
4. P. A. Bernstein, A. Y. Halevy, and R. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.
5. R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, Sept. 1987.