# Model-Independent Schema and Data Translation

Paolo Atzeni[1], Paolo Cappellari[1], Philip A. Bernstein[2]

[1] Università Roma Tre, Italy
atzeni@dia.uniroma3.it, cappellari@dia.uniroma3.it
[2] Microsoft Research, Redmond,WA, USA
philbe@microsoft.com

**Abstract.** We describe MIDST, an implementation of the model management operator ModelGen, which translates schemas from one model to another, for example from OO to SQL or from SQL to XSD. It extends past approaches by translating database instances, not just their schemas. The operator can be used to generate database wrappers (e.g. OO or XML to relational), default user interfaces (e.g. relational to forms), or default database schemas from other representations. The approach translates both schemas and data: given a source instance $I$ of a schema $S$ expressed in a source model, and a target model TM, it generates a schema $S'$ expressed in TM that is "equivalent" to $S$ and an instance $I'$ of $S'$ "equivalent" to $I$. A wide family of models is handled by using a metamodel in which models can be succinctly and precisely described. The approach expresses the translation as Datalog rules and exposes the source and target of the translation in a generic relational dictionary. This makes the translation transparent, easy to customize and model-independent.

## 1 Introduction

### 1.1 The problem

To manage heterogeneous data, many applications need to translate data and their descriptions from one model (i.e. data model) to another. Even small variations of models are often enough to create difficulties. For example, while most database systems are now object-relational (OR), the actual features offered by different systems rarely coincide, so data migration requires a conversion. Every new database technology introduces more heterogeneity and thus more need for translations. For example, the growth of XML has led to such issues, including *(i)* the need to have object-oriented (OO) wrappers for XML data, *(ii)* the translation from nested XML documents into flat relational databases and vice versa, *(iii)* the conversion from one company standard to another, such as using attributes for simple values and sub-elements for nesting vs. representing all data in sub-elements. Other popular models lead to similar issues, such as Web site descriptions, data warehouses, and forms. In all these settings, there is the need to translate both schemas and data from one model to another. A requirement of an even larger set of contexts is to be able to translate schemas only. This is called the *ModelGen* operator in [7].

Given two models $M_1$ and $M_2$ and a schema $S_1$ of $M_1$, *ModelGen* translates $S_1$ into a schema $S_2$ of $M_2$ that properly represents $S_1$. If data is of interest, it should translate that as well. Given a database instance $I_1$ of $S_1$ we want to produce an instance $I_2$ of $S_2$ that has the same information content as $I_1$.

As there are many different models, what we need is an approach that is generic across models, and can handle the idiosyncrasies of each model. Ideally, one implementation should work for a wide range of models, rather than implementing a custom solution for each pair of models.

We illustrate the problem with some of its major features by means of a short example (additional ones appear in Sec. 5). Consider a simple OR model whose tables have system-managed identifiers and tuples contain domain values as well as identifier-based references to other tables. Fig. 1 shows a database for this model with information about employees and departments: values for attribute Dept in relation Employees contain a system managed identifier that refers to tuples of Departments. For example, E#1 in Employees refers to D#1 in Departments.

| | 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

To translate OR databases into the relational model, we can follow the well known technique that replaces explicit references by values. However, some details of this transformation depend upon the specific features in the source and target model. For example, in the object world, keys (or visible identifiers) are sometimes ignored; in the figure: can we be sure that employee numbers identify employees and names identify departments? It depends on whether the model allows for keys and on whether keys have actually been specified. If keys have been specified on both object tables, then Fig. 2 is a plausible result. Its schema has tables that closely correspond to the object tables in the source database. In Fig. 2 keys are underlined and there is a referential integrity constraint from the Dept attribute in Employees to (the key of) Departments.

If instead the OR model does not allow the specification of keys or allows them to be omitted, then the translation has to be different: we use an additional attribute for each table as an identifier, as shown in Fig. 3. This attribute is visible, as opposed to the system managed ones of the OR model. We still have the referential integrity constraint from the Dept attribute in Employees to Departments, but it is expressed using the new attribute as a unique identifier.

| 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

| Employees | | | |
|---|---|---|---|
| EmpID | EmpNo | Name | Dept |
| 1 | 134 | Smith | 1 |
| 2 | 201 | Jones | 2 |
| 3 | 255 | Black | 1 |
| 4 | 302 | Brown | NULL |

| Departments | | |
|---|---|---|
| DeptID | Name | Address |
| 1 | A | 5, Pine St |
| 2 | B | 10, Walnut St |

**Fig. 3.** A translation with new key attributes

The example shows that we need to be able to deal with the specific aspects of models, and that translations need to take them into account: we have shown two versions of the OR model, one that has visible keys (besides the system-managed identifiers) and one that does not. Different techniques are needed to translate these versions into the relational model. In the second version, a specific feature was the need for generating new values for the new key attributes.

More generally, we are interested in the problem of developing a platform that allows the specification of the source and target models of interest (including OO, OR, ER, UML, XSD, and so on), with all relevant details, and to generate the translation of their schemas and instances.

## 1.2 The MDM approach

Given the difficulty of this problem, there is no complete general approach available to its solution, but there have been a few partial efforts (see Sec. 6). We use as a starting point the MDM proposal [3]. In that work 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 [18] 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. In MDM, a metamodel is defined by these generic metaconstructs. Each model is defined by its constructs and the metaconstructs they refer to. The models in the examples in Sec. 1.1 could be defined as follows:

– 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;

– a simplified OR model has (i) abstracts (tables with system-managed identifiers); (ii) lexical attributes of abstracts (for example Name and Address), each of which can be specified as part of the key; (iii) reference attributes for abstracts, which are essentially functions from abstracts to abstracts (in the example, the Dept attribute in table EMPLOYEES).

A major concept in the MDM 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 and a schema in any model is also a schema in the supermodel, apart from the specific names used for constructs. The translation of a schema from one model to another is defined in terms of translations over the metaconstructs. The supermodel acts as a "pivot" model, so that it is sufficient to have translations from each model to and from the supermodel, rather than translations for every pair of models. Thus, a linear and not a quadratic number of translations is needed. Moreover, since every schema in any model is an instance of the supermodel, the only needed translations are those within the supermodel with the target model in mind; a translation is performed by eliminating constructs not allowed in the target model, and possibly introducing new constructs that are allowed.

Each translation in MDM is built from elementary transformations, which are essentially elimination steps. So, a possible translation from the OR model to the relational one is to have two elementary transformations (i) one that eliminates references to abstracts by adding aggregations of abstracts (i.e., replacing functions with relationships), and (ii) a second that replaces abstracts and aggregations of abstracts with aggregations of lexicals and foreign keys (the traditional steps in translating from the ER to the relational model). Essentially, MDM handles a library of elementary transformations and uses them to implement complex transformations.

The major limitation of MDM with respect to our problem is that it considers schema translations only and it does not address data translation at all.

### 1.3 Contribution

This paper proposes *MIDST (Model Independent Data and Schema Translation)* a framework for the development of an effective implementation of a generic (i.e., model independent) platform for schema and data translation. It is among the first approaches that include the latter. (Sec. 6 describes concurrent efforts.) MIDST is based on the following novel ideas:

– a visible *dictionary* that includes three parts (i) the meta-level that contains the description of models, (ii) the schema-level that contains the description of schemas; (iii) the data-level that contains data for the various schemas. The first two levels are described in detail in Atzeni et al. [2]. Instead, the focus and the novelty here are in the relationship between the second and third levels and in the role of the dictionary in the translation process;

– the elementary translations are also visible and independent of the engine that executes them. They are implemented by rules in a Datalog variant with Skolem functions for the invention of identifiers; this enables one to easily modify and personalize rules and reason about their correctness;
– the translations at the data level are also written in Datalog and, more importantly, are generated almost automatically from the rules for schema translation. This is made possible by the close correspondence between the schema-level and the data-level in the dictionary;
– mappings between source and target schemas and data are generated as a by-product, by the materialization of Skolem functions in the dictionary.

A demo description of a preliminary version of the tool considering only the schema level is in Atzeni et al. [1].

### 1.4 Structure of the paper

The rest of the paper is organized as follows. Sec. 2 explains the schema level of our approach. It describes the dictionary and the Datalog rules we use for the translation. Sec. 3 covers the major contribution: the automatic generation of the rules for the data level translation. Sec. 4 discusses correctness at both schema and data level. Sec. 5 presents experiments and more examples of translations. Sec. 6 discusses related work. Sec. 7 is the conclusion.

## 2 Translation of schemas

In this section we illustrate our approach to schema translation. We first explain how schemas are described in our dictionary using a relational approach. We then show how translations are specified by Datalog rules, which leverage the relational organization of the dictionary. Two major features of the approach are the unified treatment of schemas within the supermodel and the use of Skolem functors for generating new identifiers in the dictionary. We will comment on each of them while discussing the approach.

### 2.1 Description of schemas in the dictionary

A schema is described in the dictionary as a set of schema elements, with references to both its specific model and the supermodel [2]. For example, an entity of an ER schema is described both in a table, say ER_Entity, referring to the ER model and in a supermodel table SM_Abstract, corresponding to the abstract metaconstruct to which the entity construct refers. Similarly, a class of a UML diagram gives rise to a tuple in a specific table UML_Class and to one in SM_Abstract again, because classes also correspond to abstracts. As we will see in Sec. 2.2, our translation process includes steps ("copy rules") that guarantee the alignment of the two representations.

| SM_Abstracts | | |
|---|---|---|
| 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 |
| 204 | 1 | Address | F | F | 102 | String |
| ... | ... | ... | ... | ... | ... | ... |

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

**Fig. 4.** An object-relational schema represented in the dictionary

The supermodel's structure is relatively compact. In our relational implementation, it has a table for each construct. We currently have a dozen constructs, which are sufficient to describe a large variety of models. Translation rules are expressed using supermodel constructs. Therefore, they can translate any construct that corresponds to the same metaconstruct, without having to rewrite rules for each construct of a specific model. Therefore, we concentrate here on the portion of the dictionary that corresponds to the supermodel, as it is the only one really relevant for translations.

In the dictionary, each schema element has (i) a unique identifier (OID), (ii) a reference to the schema it belongs to (sOID), (iii) values of its properties and (iv) references to other elements of the same schema. Each schema element belongs to only one schema.

In the schema of Fig. 1 both Employees and Departments are object-tables with identifiers and therefore correspond to the *abstract* metaconstruct. Dept is a reference attribute (its values are system-managed identifiers) and in our terminology corresponds to *reference attribute of abstract.* The other attributes are value based and therefore correspond to the metaconstruct *attribute of abstract.* Fig. 4 shows how the description of the schema in Fig. 1 is organized in the dictionary of MIDST. To illustrate the main points, consider the table SM_RefAttributeOfAbstract. The tuple with OID 301 belongs to schema 1. It has two properties: Name, with value "Dept" and IsNullable with value true (it says that nulls are allowed in the database for this attribute). Finally, it has two references AbsOID and AbsToOID, which denote the element this attribute belongs to and the element it refers to, respectively: this attribute belongs to Employees (the abstract with OID 101) and points to Departments (the abstract with OID 102).

### 2.2 Rules for schema translation

As in the MDM approach, translations are built by combining elementary translations. The novelty here is that each elementary translation is specified by means of a set of rules written in a Datalog variant with Skolem functors for the generation of new identifiers. Elementary translations can be easily reused because they refer to the constructs in supermodel terms, and so each of them can be applied to all constructs that correspond to the same metaconstruct. The actual translation process includes an initial step for "copying" schemas from the specific source model to the supermodel and a final one for going back from the supermodel to the target model of interest. For the sake of space we omit the discussion of these two steps, as they are straightforward.

We illustrate the major features of our rules by means of an example, which refers to the translation from the OR to the relational models, specifically, mapping the database of Fig.1 to that of Fig.2. The following rule translates object references (attribute Dept in relation EMPLOYEES) into value based references:

```
 SM_ATTRIBUTEOFAGGREGATIONOFLEXICALS(
     OID:#attribute_4(refAttOid, attOid), sOID:target, Name:refAttName,
     IsKey: "FALSE", IsNullable:isN, AggOID:#aggregation_2(absOid))
← SM_REFATTRIBUTEOFABSTRACT(
     OID:refAttOid, sOID:source, Name:refAttName, IsNullable:isN,
     AbsOID:absOid, AbsToOID:absToOid),
   SM_ATTRIBUTEOFABSTRACT(
     OID:attOid, sOID:source, Name:attName,
     IsKey:"TRUE", AbsOID:absToOid)
```

The rule replaces each reference (SM_REFATTRIBUTEOFABSTRACT) with one column (SM_ATTRIBUTEOFAGGREGATIONOFLEXICALS) for each key attribute of the referenced table. The body unifies with a reference attribute and a key attribute (note the constant TRUE for IsKey in the body) of the abstract that is the target of the reference (note the variable *absToOid* that appears twice in the body). In our example, as DEPARTMENTS has only one key attribute (Name), the rule would generate exactly one new column for the reference.

Skolem functors are used to create new OIDs for the elements the rule produces in the target schema.[3] The head of the rule above has two functors: #attribute_4 for the OID field and #aggregation_2 for the AggOID field. The two play different roles. The former generates a new value, which is distinct for each different tuple of arguments, as the function associated with the functor is injective. This is the case for all the functors appearing in the OID field of the head of a rule. The second functor correlates the element being created with an element created by another rule, namely the rule that generates an aggregation of lexicals (that is, a relation) for each abstract (that is, an object table). The new SM_ATTRIBUTEOFAGGREGATIONOFLEXICALS being generated indeed belongs to the SM_AGGREGATIONOFLEXICALS generated for the SM_ABSTRACT denoted by variable *absOid*.

[3] A brief comment on notation: functors are denoted by the # sign, include the name of the construct whose OIDs they generate (here often abbreviated for convenience), and have a suffix that distinguishes the various functors associated with a construct.

| SM_InstOfAbstract | | |
|---|---|---|
| OID | dOID | AbsOID |
| 1001 | 1 | 101 |
| 1002 | 1 | 101 |
| 1003 | 1 | 101 |
| 1004 | 1 | 101 |
| 1005 | 1 | 102 |
| 1006 | 1 | 102 |
| ... | ... | ... |

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

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

**Fig. 5.** Representation of an object relational instance

As another example, consider the rule that, in the second translation mentioned in the Introduction (Fig. 3), produces new key attributes when keys are not defined in the OR-tables in the source schema.

SM_AttributeOfAggregationOfLexicals(
  OID:#attribute_5(*absOid*), sOID:*target*, Name:*name*+'ID',
  IsNullable:"FALSE", IsKey:"TRUE", AggOID:#aggregation_2(*absOid*))
← SM_Abstract(
  OID:*absOid*, sOID:*source*, Name:*name*)

The new attribute's name is obtained by concatenating the name of the instance of SM_Abstract with the suffix 'ID'. We obtain EmpID and DeptID as in Fig. 3.

## 3 Data translation

The main contribution of MIDST is the management of translations of actual data, derived from the translations of schemas. This is made possible by the use of a dictionary for the data level, built in close correspondence with the schema level one. Therefore, we first describe the dictionary and then the rules.

### 3.1 Description of data

Data are described in a portion of the dictionary whose structure is automatically generated and is similar to the schema portion. The basic idea is that all data elements are represented by means of internal identifiers and also have a value, when appropriate. A portion of the representation of the instance in Fig. 1 is shown in Fig. 5. Let us comment the main points:

– Each table has a dOID (for *database OID*) attribute, instead of the sOID attribute we had at the schema level. Our dictionary can handle various schemas for a model and various instances (or databases) for each schema. In the example, we show only one database, with 1 as the dOID.
– Each data element has a reference to the schema element it instantiates. For example, the first table in Fig. 5 has an AbsOID column, whose values are identifiers for the abstracts in the schema. The first four rows have a value 101 for it, which is, in Fig. 4, the identifier of object-table EMPLOYEES; in fact, the database (see Fig. 1) has four elements in EMPLOYEES.
– "Properties" of schema elements (such as IsKey and IsNullable) do not have a counterpart at the data level: they only are needed as schema information.
– All identifiers appearing at the schema level are replaced by identifiers at the data level. They include both the OID and the references to the OIDs of other tables. In the example, table SM_REFATTRIBUTEOFABSTRACT in Fig. 4 has columns (i) OID, the identifier of the row, (ii) AbsOID, the identifier of the abstract to which the attributes belong, and (iii) AbsToOID, the identifier of the abstract to which the attributes "point". In Fig. 5 each of them is taken one level down: (i) each row is still identified by an OID column, but this is the identifier of the data element; (ii) each value of i-AbsOID indicates the instance of the abstract the attribute is associated with (1001 in the first tuple of SM_INSTOFREFATTRIBUTEOFABSTRACT in Fig. 5 identifies employee Smith); (iii) i-AbsToOID indicates the instance of the abstract the attribute refers to (in the same tuple, 1005 identifies department A);
– If the construct is lexical (that is, has an associated value [18]), then the table has a Value column. In Fig. 5, SM_INSTANCEOFATTRIBUTEOFABSTRACT is the only lexical construct, and Value contains all the values for all the attributes of all the abstracts. Differences in type are not an issue, as we assume the availability of serialization functions that transform values of any type into values of a common one (for example strings).

The above representation for instances is clearly an "internal" one, into which or from which actual database instances or documents have to be transformed. We have developed import/export features that can upload/download instances and schemas of a given model. This representation is somewhat onerous in terms of space, so we are working on a compact version of it that still maintains the close correspondence with the schema level, which is its main advantage.

## 3.2 Rules for data translation

The close correspondence between the schema and data levels in the dictionary allows us to automatically generate rules for translating data, with minor refinements in some cases. The technique is based on the DOWN function, which transforms schema translation rules "down to instances." It is defined both on Datalog rules and on literals. If $r$ is a schema level rule with $k$ literals in the body, DOWN($r$) is a rule $r'$, where:

- the head of $r'$ is obtained by applying the DOWN function to the head of $r$ (see below for the definition of DOWN on literals)
- the body of $r'$ has two parts, each with $k$ literals:
  1. literals each obtained by applying DOWN to a literal in the body of $r$;
  2. a copy of the body of $r$.

Let us now define DOWN on literals. A literal is a possibly negated atom. An atom has the form $P(n_1 : a_1, \ldots, n_h : a_h)$, where $P$, the *predicate*, is the name of the table for a supermodel construct (therefore beginning with the prefix SM_), each $n_i$ (a *name*) is a column (property or reference) of $P$ and each $a_i$ is an *argument*, which can be a constant, a variable,[4] or a Skolem functor. In turn, a Skolem functor has the form $F(p_1, \ldots, p_m)$, where $F$ is the name of a Skolem function and each $p_j$ is a constant or a variable.

Given a schema level atom $l_S = P(n_1 : a_1, \ldots, n_h : a_h)$, DOWN produces a data level literal with a predicate name obtained from $P$ by replacing SM_ with SM_INSTANCEOF[5] and arguments as follows.

- Two pairs are built from the OID argument (OID: $a$) of $l_S$:
  - (OID: $a'$) where $a'$ is obtained from $a$ as follows, depending on the form of $a$: if $a$ is a variable, then $a'$ is obtained by prefixing i- to its name; if instead it is a Skolem functor, both the function name and its variable parameters are prefixed with i-;
  - ($P$-OID: $a$), where $P$-OID is the reference to the schema element in the dictionary (built as the concatenation of the name of $P$ and the string OID.
- For each pair ($n$:$a$) in $l_S$ where $n$ is a reference column in table $P$ in the dictionary (that is, one whose values are OIDs), DOWN($l_S$) contains a pair of the form ($n'$:$a'$), where $n'$ is obtained from $n$ by adding a "i-" prefix and $a'$ is obtained from $a$ as above with the additional case that if it is a constant then it is left unchanged.
- If the construct associated with $P$ is lexical (that is, its occurrences at the data level have values), then an additional pair of the form (Value:$e$) is added, where $e$ is an expression that in most cases is just a variable $v$ (we comment on this issue at the end of this section).

Let us consider the first rule presented in Sec. 2.2. At the data level it is as follows:

SM_INSTANCEOFATTRIBUTEOFAGGREGATIONOFLEXICALS(
    OID:#i-attribute_4(*i-refAttOid*, *i-attOid*),
    AttOfAggOfLexOID:#attribute_4(*refAttOid*, *attOid*), dOID:*i-target*,
    i-AggOID:#i-aggregation_2(*i-absOid*), Value:$v$)
$\leftarrow$ SM_INSTANCEOFREFATTRIBUTEOFABSTRACT(

[4] In general, an argument could also be an expression (for example a string concatenation over constants and variables), but this is not relevant here.
[5] In figures and examples we abbreviate names when needed.

```
    OID:i-refAttOid, RefAttOfAbsOID:refAttOid, dOID:i-source,
    i-AbsOID:i-absOid, i-AbsToOID:i-absToOid),
  SM_InstanceOfAttributeOfAbstract(
    OID:i-attOid, AttOfAbsOID:attOid, dOID:i-source, i-AbsOID:i-absToOid,
    Value:v),
  SM_RefAttributeOfAbstract(
    OID:refAttOid, sOID:i-source, Name:refAttName, IsNullable:isN,
    AbsOID:absOid, AbsToOID:absToOid),
  SM_AttributeOfAbstract(
    OID:attOid, sOID:source, Name:attName,
    IsKey:"TRUE", AbsOID:absToOid)
```

Let us comment on some of the main features of the rules generation.

1. schema level identifiers become data level identifiers: OID element;
2. data elements refer to the schema elements they instantiate;
3. references to schemas become references to databases, that is, instances of schemas: both in the head and in the second literal in the body, we have a dOID column instead of the sOID;
4. Skolem functors are replaced by "homologous" functors at the data level, by transforming both the name and the arguments; in this way, they generate new data elements, instead of schema elements;
5. "properties" do not appear in data level literals; they are present in the schema level literals in order to maintain the same selection condition (on schema elements) declared in the body of the schema level translation;
6. lexical constructs have a Value attribute.

The copy of the body of the schema-level rule is needed to maintain the selection condition specified at the schema level. In this way the rule translates only instances of the schema element selected within the schema level rule.

In the rule we just saw, all values in the target instance come from the source. So we only need to copy them, by using a pair (Value:*v*) both in the body and the head. Instead, in the second rule in Sec. 2.2, the values for the new attribute should also be new, and a different value should be generated for each abstract instance. To cover all cases, MIDST allows functions to be associated with the Value field. In most cases, this is just the identity function over values in the source instance (as in the previous rule). In others, the rule designer has to complete the rule by specifying the function. In the example, the rule is as follows:

```
 SM_InstanceOfAttributeOfAggregationOfLexicals(
    OID:#i-attribute_5(i-absOid), AttOfAggOfLex:#attribute_5(absOid),
    dOID:i-target, Value:valueGen(i-absOid),
    i-AggOID:#i-aggregation_2(i-absOid) )
← SM_InstanceOfAbstract(
    OID:i-absOid, AbsOID:absOid, dOID:i-source),
  SM_Abstract(
    OID:absOid, sOID:source, Name:name)
```

## 4 Correctness

In data translation (and integration) frameworks, correctness is usually modelled in terms of information-capacity dominance and equivalence (see Hull [16, 17] for the fundamental notions and results and Miller et al. [19, 20] for their role in schema integration and translation). In this context, it turns out that various problems are undecidable if they refer to models that are sufficiently general (see Hull [17, p.53], Miller [20, p.11-13]). Also, a lot of work has been devoted over the years to the correctness of specific translations, with efforts still going on with respect to recently introduced models: see for example the recent contributions by Barbosa et al. [5, 6] on XML-to-relational mappings and by Bohannon et al. [11] on transformations within the XML world. Undecidability results have emerged even in discussions on translations from one specific model to another specific one [5, 11].

Therefore, given the genericity of our approach, it seems hopeless to aim at showing correctness in general. However, this is only a partial limitation, as we are developing a platform to support translations, and some responsibilities can be left to its users (specifically, rule designers, who are expert users), with system support. We briefly elaborate on this issue.

We follow the initial method of Atzeni and Torlone [3] for schema level translations, which uses an "axiomatic" approach. It assumes the basic translations to be correct, a reasonable assumption as they refer to well-known elementary steps developed over the years. It is the responsibility of the rule's designer to specify basic translations that are indeed correct. So given a suitable description of models and rules in terms of the involved constructs, complex translations can be proven correct by induction.

In MIDST, we have the additional benefit of having schema level transformations expressed at a high-level, as Datalog rules. Rather than taking on faith the correctness of the signature of each basic transformation as in [3], we can automatically detect which constructs are used in the body and generated in the head of a Datalog rule and then derive the signature. Since models and rules are expressed in terms of the supermodel's metaconstructs, by induction, the same can be done for the model obtained by applying a complex transformation.

For correctness at the data level, we can reason in a similar way. The main issue is the correctness of the basic transformations, as that of complex ones would follow by induction. Again, it is the responsibility of the designer to verify the correctness of the rules: he/she specifies rules at the schema level, the system generates the corresponding data-level rules, and the designer tunes them if needed and verifies their correctness. It can be seen that our data-level rules generate syntactically correct instances (for example, with only one value for single-valued attributes) if the corresponding schema-level rules generate syntactically correct schemas.

The validity of the approach, given the unavailability of formal results has been evaluated by means of an extensive set of test cases, which have produced positive results. We comment on them in the next section.

## 5 Experimentation

The current MIDST prototype handles a metamodel with a dozen different metaconstructs, each with a number of properties. For example, attributes with nulls and without nulls are just variants of the same construct. These metaconstructs, with their variants, allow for the definition of a huge number of different models (all the major models and many variations of each). For our experiments, we defined a set of significant models, extended-ER, XSD, UML class diagrams, object-relational, object-oriented and relational, each in various versions (with and without nested attributes and generalization hierarchies).

We defined the basic translations needed to handle the set of test models. There are more than twenty of them, the most significant being those for eliminating n-ary aggregations of abstracts, eliminating many-to-many aggregations, eliminating attributes from aggregations of abstracts, introducing an attribute (for example to have a key for an abstract or aggregation of lexicals), replacing aggregations of lexicals with abstracts and vice versa (and introducing or removing foreign keys as needed), unnesting attributes and eliminating generalizations. Each basic transformation required from five to ten Datalog rules.

These basic transformations allow for the definition of translations between each pair of test models. Each of them produced the expected target schemas, according to known standard translations used in database literature and practice.

At the data level, we experimented with the models that handle data, hence object-oriented, object-relational, and nested and flat relational, and families of XML documents. Here we could verify the correctness of the Down function, the simplicity of managing the rules at the instance level and the correctness of data translation, which produced the expected results.

In the remainder of this section we illustrate some major points related to two interesting translations. We first show a translation of XML documents between different company standards and then an unnesting case, from XML to the relational model.

For the first example, consider two companies that exchange data as XML documents. Suppose the target company doesn't allow attributes on elements: then, there is the need to translate the source XML conforming a source XSD into another one conforming the target company XSD. Fig. 6 shows a source and a target XML document.

Let us briefly comment on how the XSD model is described by means of our metamodel. XSD-elements are represented with two metaconstructs: abstract for elements declared as complex type and attribute of aggregation of lexicals and abstracts for the others (simple type). XSD-groups are represented by aggregation of lexicals and abstracts and XSD-attributes by attribute of abstract. Abstract also represents XSD-Type.

The translation we are interested in has to generate: (i) a new group (specifically, a sequence group) for each complex-type with attributes, and (ii) a simple-element belonging to such a group for each attribute of the complex-type. Let's

```
...
<employee>
  <name>Cappellari</name>
  <address street="52, Ciclamini St"
     city="Rome"/>
  <company>
    <name>
       University "Roma Tre"
    </name>
    <address street="84, Vasca
       Navale St" city="Rome"/>
  </company>
</employee>
...
```

```
...
<employee>
  <name>Cappellari</name>
  <address>
    <street>52, Ciclamini St</street>
    <city>Rome</city>
  </address>
  <company>
    <name>University "Roma Tre"
    </name>
    <address>
      <street>84, Vasca Navale St</street>
      <city>Rome</city>
    </address>
  </company>
</employee>
...
```

**Fig. 6.** A translation within XML: source and target

see the rule for the second step (with some properties and references omitted in the predicates for the sake of space):

SM_AttributeOfAggregationOfLexAndAbs(
    OID:#attAggLexAbs_6(*attOid*), sOID:*target*, Name:*attName*,
    aggLexAbsOID:#aggLexAbs_8(*abstractOid*))
$\leftarrow$ SM_AttributeOfAbstact(
    OID:*attOid*, sOID:*source*, Name:*attName*, abstractOID:*abstractOid*)

Each new element is associated with the group generated by the Skolem functor #aggLexAbs_8(*abstractOid*). The same functor is used in the first step of the translation and so here it is used to insert, as reference, OIDs generated in the first step. The data level rule for this step has essentially the same features as the rule we showed in Sec. 3.2: both constructs are lexical, so a Value field is included in both the body and the head. The final result is that values are copied.

As a second example, suppose the target company stores the data in a relational database. This raises the need to translate schema and data. With the relational model as the target, the translation has to generate: (a) a table (aggregation of lexicals) for each complex-type defined in the source schema, (b) a column (attribute of aggregation of lexicals) for each simple-element and (c) a foreign key for each complex element. In our approach we represent foreign keys with two metaconstructs: (i) foreign key to handle the relation between the *from* table and the *to* table and (ii) components of foreign keys to describe columns involved in the foreign key.

Step (i) of the translation can be implemented using the following rule:

```
 SM_ForeignKey(
     OID:#foreignKey_2(abstractOid), sOID:target,
     aggregationToOID:#aggregationOfLexicals_3(abstractTypeOid),
     aggregationFromOID:#aggregationOfLexicals_3(abstractTypeParentOid))
← SM_Abstract(
      OID:abstractTypeOid, sOID:source, isType:“TRUE”),
   SM_Abstract(
      OID:abstractOid, sOID:source, typeAbstractOID:abstractTypeOid,
      isType:“FALSE”, isTop:“FALSE”),
   SM_AbstractComponentOfAggregationOfLexAndAbs(
      OID:absCompAggLexAbsOid, sOID:source,
      aggregationOID:aggregationOid, abstractOID:abstractOid),
   SM_AggregationOfLexAndAbs(
      OID:aggregationOid, sOID:source, isTop:“FALSE”,
      abstractTypeParentOID:abstractTypeParentOid)
```

In the body of the rule, the first two literals select the non-global complex-element (isTop=FALSE) and its type (isType=TRUE). The other two literals select the group the complex-element belongs to and the parent type through the reference abstractTypeParentOID.

Note that this rule does not involve any lexical construct. As a consequence, the corresponding data-level rule (not shown) does not involve actual values. However, it includes references that are used to maintain connections between values.

Fig. 7 shows the final result of the translation. We assumed that no keys were defined on the document and therefore the translation introduces a new key attribute in each table.

| Employees | | | |
|---|---|---|---|
| eID | EmpName | Address | Company |
| E1 | Cappellari | A1 | C1 |
| E2 | Russo | A2 | C1 |
| E3 | Santarelli | A3 | C2 |

| Address | | |
|---|---|---|
| aID | Street | City |
| A1 | 52, Ciclamini St | Rome |
| A2 | 31, Rose St | Rome |
| A3 | 21, Margherita St | Rome |
| A4 | 84, Vasca Navale St | Rome |

| Company | | |
|---|---|---|
| cID | CompName | Address |
| C1 | University “Roma Tre” | A4 |
| C2 | Quadrifoglio s.p.a | A4 |

**Fig. 7.** A relational database for the second document in Fig. 6

## 6 Related Work

Many proposals exist that address schema and data translation. However, most of them only consider specific data models. In this section we present related pieces of work that address the problem of model-independent translations.

The term *ModelGen* was coined in [7] which, along with [8], argues for the development of model management systems consisting of generic operators for solving many schema mapping problems. An example of using *ModelGen* to help solve a schema evolution problem appears in [7].

An early approach to ModelGen was proposed by Atzeni and Torlone [3, 4] who developed the MDM tool, which we have discussed in the introduction. The basic idea behind MDM and the similar approaches (Claypool and Rundensteiner et al. [13, 14] Song et al. [25], and Bézivin et al [10]) is useful but offers only a partial solution to our problem. The main limitation is that they refer only to the schema level. In addition, their representation of the models and transformations is hidden within the tool's imperative source code, not exposed as more declarative, user-comprehensible rules. This leads to several other difficulties. First, only the designers of the tool can extend the models and define the transformations. Thus, instance level transformations would have to be recoded in a similar way. Moreover, correctness of the rules has to be accepted by users as a dogma, since their only expression is in complex imperative code. And any customization would require changes in the tool's source code. The above problems are significant even for a tool that only does schema translation, without instance translation. All of these problems are overcome by our approach.

There are two concurrent projects to develop *ModelGen* with instance translations [9, 22]. The approach of Papotti and Torlone [22] is not rule-based. Rather, their transformations are imperative programs, which have the weaknesses described above. Their instance translation is done by translating the source data into XML, performing an XML-to-XML translation expressed in XQuery to reshape it to be compatible with the target schema, and then translating the XML into the target model. This is similar to our use of a relational database as the "pivot" between the source and target databases.

The approach of Bernstein, Melnik, and Mork [9] is rule-based, like ours. However, unlike ours, it is not driven by a relational dictionary of schemas, models and translation rules. Instead, they focus on flexible mapping of inheritance hierarchies and the incremental regeneration of mappings after the source schema is modified. A detailed description of their approach has not yet appeared.

Bowers and Delcambre [12] present Uni-Level Description (UDL) as a metamodel in which models and translations can be described and managed, with a uniform treatment of models, schemas, and instances. They use it to express specific model-to-model translations of both schemas and instances. Like our approach, their rules are expressed in Datalog. Unlike ours, they are expressed for particular pairs of models.

Data exchange is a different but related problem, the development of user-defined custom translations from a given source schema to a given target, not the automated translation of a source schema to a target model. It is an old database problem, going back at least to the 1970's [24]. Some recent approaches are in Cluet et al. [15], Milo and Zohar [21], and Popa et al. [23].

## 7 Conclusions

In this paper we showed MIDST, an implementation of the ModelGen operator that supports model-generic translations of schemas and their instances within a large family of models. The experiments we conducted confirmed that translations can be effectively performed with our approach.

There are many areas where we believe additional work would be worthwhile. First, as we mentioned earlier, there is a need for more compact and efficient representations of translated instances. Second, despite the obstacles explained in Sec. 4, it would be valuable to produce a practical way to validate the correctness of a set of complex transformations. Third, there is a need to support all of the idiosyncrasies of rich models and exotic models, and to support more complex mappings, such as the many variations of inheritance hierarchies. Fourth, it would be helpful for users to be able to customize the mappings.

**Acknowledgements**

We would like to thank Luca Santarelli for his work in the development of the tool and Chiara Russo for contributing to the experimentation and for many helpful discussions.

## References

1. P. Atzeni, P. Cappellari, and P. A. Bernstein. Modelgen: Model independent schema translation. In *ICDE, Tokyo*, pages 1111–1112. IEEE Computer Society, 2005.
2. P. Atzeni, P. Cappellari, and P. A. Bernstein. A multilevel dictionary for model management. In *ER 2005, LNCS 3716*, pages 160–175. Springer, 2005.
3. 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.
4. P. Atzeni and R. Torlone. Mdm: a multiple-data-model tool for the management of heterogeneous database schemes. In *SIGMOD*, pages 528–531. ACM Press, 1997.
5. D. Barbosa, J. Freire, and A. O. Mendelzon. Information preservation in XML-to-relational mappings. In *XSym 2004, LNCS 3186*, pages 66–81. Springer, 2004.
6. D. Barbosa, J. Freire, and A. O. Mendelzon. Designing information-preserving mapping schemes for XML. In *VLDB*, pages 109–120, 2005.
7. P. A. Bernstein. Applying model management to classical meta data problems. *CIDR*, pages 209–220, 2003.
8. P. A. Bernstein, A. Y. Halevy, and R. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.
9. P. A. Bernstein, S. Melnik, and P. Mork. Interactive schema translation with instance-level mappings. In *VLDB*, pages 1283–1286, 2005.

10. J. Bézivin, E. Breton, G. Dupé, and P. Valduriez. The ATL transformation-based model management framework. Research Report Report 03.08, IRIN, Université de Nantes, 2003.
11. P. Bohannon, W. Fan, M. Flaster, and P. P. S. Narayan. Information preserving XML schema embedding. In *VLDB*, pages 85–96, 2005.
12. S. Bowers and L. M. L. Delcambre. The Uni-Level Description: A uniform framework for representing information in multiple data models. In *ER 2003, LNCS 2813*, pages 45–58, 2003.
13. K. T. Claypool and E. A. Rundensteiner. Gangam: A transformation modeling framework. In *DASFAA*, pages 47–54, 2003.
14. K. T. Claypool, E. A. Rundensteiner, X. Zhang, H. Su, H. A. Kuno, W.-C. Lee, and G. Mitchell. Sangam - a solution to support multiple data models, their mappings and maintenance. In *SIGMOD Conference*, 2001.
15. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *SIGMOD Conference*, pages 177–188, 1998.
16. R. Hull. Relative information capacity of simple relational schemata. *SIAM J. Comput.*, 15(3):856–886, 1986.
17. R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *PODS, Tucson, Arizona*, pages 51–61. ACM Press, 1997.
18. R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, Sept. 1987.
19. R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *VLDB*, pages 120–133, 1993.
20. R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: bridging theory and practice. *Inf. Syst.*, 19(1):3–31, 1994.
21. T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB*, pages 122–133. Morgan Kaufmann Publishers Inc., 1998.
22. P. Papotti and R. Torlone. Heterogeneous data translation through XML conversion. *J. Web Eng.*, 4(3):189–204, 2005.
23. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating Web data. In *VLDB*, pages 598–609, 2002.
24. N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. Express: A data extraction, processing, amd restructuring system. *ACM Trans. Database Syst.*, 2(2):134–174, 1977.
25. G. Song, K. Zhang, and R. Wong. Model management though graph transformations. In *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 75–82, 2004.