A multilevel dictionary for model management

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

Abstract. We discuss the main features of a multilevel dictionary based on a metamodel approach. The application is an implementation of ModelGen, the model management operator that translates schemas from one model to another, for example from ER to relational or from XSD to object. The dictionary manages schemas and, at a metalevel, a description of the models of interest. It describes all models in terms of a limited set of metaconstructs. It describes all the schemas in a unifying model, called the supermodel, which generalizes all the others. The dictionary is composed of four parts, based on the combination of two features: schema level or model level, and model specific or model generic. We also show how such a dictionary can be the basis for a model independent approach to reporting, that provides a detailed textual and XML description of schemas.

1 Introduction

The need for handling design artifacts corresponding to different models arises in many different application settings. In the database world, we often have different systems that we need to use to handle our data, which use different (data) models, or we might need to exchange or integrate schemas expressed in different models. Small variations of models are often enough to create difficulties: for example, while most designers use the ER model, the actual features adopted by different methodologies and tools almost never coincide and so any integration requires a conversion. The introduction of new technology often introduces more heterogeneity and more need for translations. This happened with respect to analysis and design settings, where the growth of UML has not really simplified things, as the formalism is very complex and most people use just a subset of it—but each uses a different subset! XML has also contributed to increased heterogeneity, as data sources are now often described by means of XML Schemas (possibly in simplified versions).

In all these situations, there is the need to translate design artifacts from one model to another. There is also a need to translate data, but we focus only on design artifacts here. The models of interest can be significantly different, including those traditionally used in databases, as well as many others, such as those for XML documents, Web site structure descriptions, data warehouses, and restrictions or variants of each of them. The requirement, in all these cases,

is the capability to handle different models and to be able to translate schemas from one model to another: given a source scheme S_1 in a model M_1 and a target model M_2, the need is to be able to generate the translation of S_1 into M_2.

These translation problems have always been tackled in practical settings by means of ad-hoc solutions, for example by writing a program for each specific application. This is clearly very expensive, as it is laborious and hard to maintain. Bernstein et al. [8, 9] have recently argued for generic solutions for all problems that require the management of descriptions of application artifacts. They 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 model to a target model, exactly as we required above. This paper reports on some new features of a recent development for ModelGen.

An early approach to ModelGen was proposed by Atzeni and Torlone [4, 5] who developed a tool, called MDM, to manage heterogeneous schemas based on a notion of *metamodel*. A metamodel is a set of constructs (the *metaconstructs*) that can be used to define models. 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. In this approach, 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. Other authors have proposed similar approaches, including Claypool and Rundensteiner et al. [11, 12], Song et al. [19], Bézivin et al [10]. The ideas at the basis of the MDM tool and of similar approaches are interesting and useful. But they do have one weakness, namely that they hide the representation of the models and transformations within the source code of the tool. So any extension of the models or customization of the translations would be very complex.

We have recently started a completely new development for ModelGen, with various novel features. One important feature is that it makes the description of models and the specification of translations visible and easily modifiable. In this paper, we give a detailed account of the dictionary that enables this feature, with its structure and the consequent benefits. A preliminary discussion of the overall development is available in [2].

The main contribution of this paper is the structure of the dictionary, which allows the integrated management of the descriptions of models and schemas for the various models of interest. The dictionary has a relational structure, which allows for the effective development of translation steps by means of Datalog rules [2]. It also makes it easy to produce model-specific reports in a model-independent way.

The paper is organized as follows. Section 2 illustrates the background of the approach, its main features and the relationship with some related literature. The next two sections describe the dictionary in some detail: Section 3 concentrates on the lower level, which describes schemas and Section 4 concentrates on the

metalevel, which describes models and thus the structure of the lower level. Then, in Section 5 we show how the approach can be the basis for effective, model-independent reporting. Section 6 is the conclusion.

2 Background, contribution and related work

The starting point for our work is the MDM proposal [4], whose principles are as follows. 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 [14] 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, and function. Each model is defined by its constructs and the metaconstructs they refer to. Simple versions of popular models are as follows:

- a simplified version of the ER model involves entities (which correspond to the *abstract* metaconstruct), attributes for them (corresponding to the metaconstruct *attribute of abstract*), and relationships (*aggregations of abstracts*);
- a simplified version of the object-oriented (OO) model involves classes (which also correspond to abstracts), fields (also attributes of abstracts), and references from classes to classes (the metaconstruct *reference to abstract*).

The translation of a schema from one model to another is defined in terms of translations over the metaconstructs. A major concept in the approach is the *supermodel*, a model that has constructs corresponding to all the 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, apart from the specific names used for constructs. 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 each pair of models. Thus, only $2n$ translations are needed between n models, not n^2 translations. 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 models in mind; a translation is performed by eliminating constructs not allowed in the target model, and possibly introducing new constructs.

In our new ModelGen development effort, we wanted to automate as many activities as possible, and to support the rapid construction and maintenance of the others. A visible dictionary turned out to be a major contribution in this direction. We realized that a database structure for it would be effective, especially a relational one, as we felt that the translation steps could be effectively implemented in Datalog. To handle models and schemas effectively and to coordinate the individual models with the supermodel, we organized the dictionary in four parts, which can be characterized along two coordinates: the first corresponding to whether they describe models or schemas and the second depending on whether they refer to specific models or to the supermodel (see Figure 1).

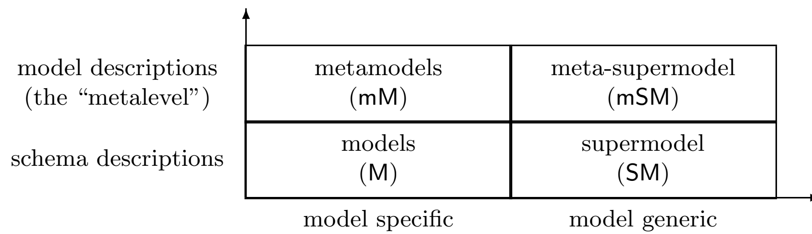

Fig. 1. The four parts of the dictionary

The most abstract part is at the model level and is model generic. It describes the supermodel, that is, the set of constructs used by the tool for building schemas. We refer to this part as mSM, the "meta-supermodel," as it is about the supermodel. The second part at the model level is model specific. It describes the individual models. It includes metamodels, referred to as mM. For each of them, mM describes the specific constructs used, each corresponding to a construct in the supermodel. We refer to these first two parts as the "metalevel" of the dictionary, as it contains the description of the structure of the lower level, whose content describe schemas. The lower level is also composed of two parts, one referring to the supermodel constructs (therefore called the SM part) and the other to model-specific constructs (the M part). The structure of the schema level is, in our system, automatically generated out of the content of the metalevel: so we can say that the dictionary is self-generating out of a small core.

In the next two sections we discuss the details of the two levels of the dictionary, proceeding bottom up. In Section 3 we concentrate on the schema level. In Section 4 we illustrate the metalevel and its relationship to the lower one.

There are many proposals for dictionary structure in the literature. The use of dictionaries to handle metadata has been popular since the early database systems of the 1970's, initially in systems that were external to those handling the database (see Allen et al. [1] for an early survey). With the advent of relational systems in the 1980's, it became possible to have dictionaries be part of the database itself, within the same model. Today, all DBMSs have such a component. Extensive discussion was also carried out in even more general frameworks, with proposals for various kinds of dictionaries, describing various features of systems (see for example [6, 13, 15]) within the context of industrial CASE tools and research proposals. More recently, a number of metadata repositories have been developed [17]. They generally use relational databases for handling the information of interest. There are other significant recent efforts towards the description of multiple models, including the Model Driven Architecture (MDA) and, within it, the Common Warehouse Metamodel (CWM) [18], and Microsoft Repository [7]; in contrast to our approach, these do not distinguish metalevels, as the various models of interest are all specializations of a most general one, UML based.

The description of models in terms of the (meta-)constructs of a metamodel was proposed by Atzeni and Torlone [4]. But it used a sophisticated graph language, which was hard to implement. The other papers that followed the same or similar approaches [10–12, 19] also used specific structures.

We know of no literature that describes a dictionary that exposes schemas and instances in a highly correlated way, in both model-specific and model-independent ways. Only portions of similar dictionaries have been proposed. None of them offer the rich interrelated structure we have here.

3 The schema level

Let us proceed with the description of the dictionary starting from the most natural parts, those that describe schemas. The M component has the structure of traditional dictionaries: a table for each construct, with tuples corresponding to the elements in the schemas. In Figure 3 we show the dictionary for a version of the ER model, containing the descriptions of the two schemas in Figure 2.

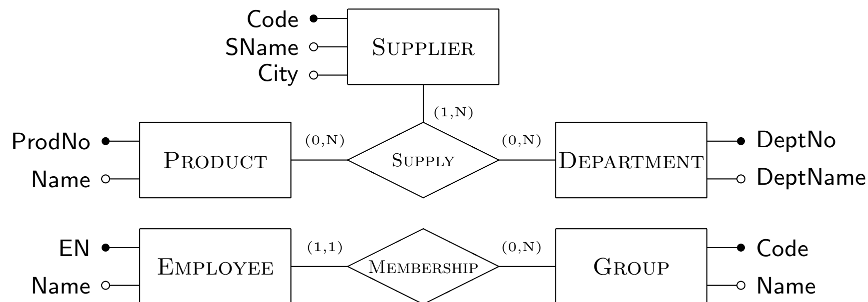

Fig. 2. Two simple ER schemas

The structure of the dictionary is rather standard and follows that often used in textbooks for describing the model (for example, Atzeni et al. [3, p.178]). We consider a simple version of the model, with n-ary relationships: this requires the separate table ComponentOfRelationship, used to specify the participation of an entity to a relationship. The version of the ER model we use includes a few of the basic features, which we use as representative properties. For example, minimum cardinalities are represented by the boolean IsOptional (abbreviated as IsOpt in the figure), so that the allowed minimum cardinality is 0 ("true" as the value for IsOpt) or 1 ("false" for IsOpt) and the maximum cardinality is represented by IsFunctional (IsFunct), with 1 and "N" as possible values. Similarly, for attributes, we have the boolean property isKey, used to represent whether they belong to the (main) identifier. Many other features are omitted here for the sake of space, such as optionality or nullability of attributes and external identification of entities. But they can be easily included and have indeed been

Schema	
OID	Name
s1	1st ER Schema
s2	2nd ER Schema

Entity		
OID	Name	Schema
e1	Supplier	s1
e2	Product	s1
e3	Department	s1
e4	Employee	s2
e5	Group	s2

Relationship		
OID	Name	Schema
r1	Supply	s1
r2	Membership	s2

AttributeOfEntity					
OID	Entity	Name	Type	isKey	Sch.
a1	e1	Code	int	true	s1
a2	e1	SName	string	false	s1
a3	e1	City	string	false	s1
a4	e2	ProdNo	int	true	s1
a5	e2	Name	string	false	s1
a6	e3	DeptNo	int	true	s1
a7	e3	DeptName	string	false	s1
a8	e4	EN	int	true	s2
...	...	...	...	...	...

ComponentOfRelationship					
OID	Rel'ship	Entity	IsOpt	IsFunct	Sch.
c1	r1	e1	false	false	s1
c2	r1	e2	true	false	s1
c3	r1	e3	true	false	s1
c4	r2	e4	false	true	s2
c5	r2	e5	true	false	s2

Fig. 3. The dictionary for a simple ER model

implemented in our prototype tool. A Schema column in each table specifies the schema the constructs refer to. We could have Schema only in the Entity table and omit it from the others. But the redundancy is not a big issue and would have consequences requiring further discussion.

In this M portion of the dictionary, we would like to be able to handle different models. Therefore, we have a portion of the dictionary like the one shown in Figure 3 for each model we are interested in. For example, if we want to handle a simple version of OODB schemas, such as that in Figure 4, we need a dictionary like the one shown in Figure 5. The OO model we use here is very simple, with just classes with scalar fields (each with a type, but with no other property; for example, we assume there is no way to specify they are optional nor to define keys) and reference attributes (fields that are references to other classes, representing 1:N relationships). The Schema table is the same as the one in Figure 3: the dictionary includes one table which refers to schemas for all the models of interest. It has an additional column that specifies the model to which the schema belongs, a reference to a Model table we will see shortly in the mM part of the dictionary.

Model-specific dictionaries are pretty standard. If we represented just the relational model, we would have something similar to some of the system tables of many DBMSs. The other parts of the dictionary are original, together with the overall structure. If we refer to Figure 1 again, starting from the M quadrant, two directions of abstraction are possible: we could consider the various models altogether, rather than each of them independently, thus moving to the SM quadrant in Figure 1. Or we can go from the level of schemas to its metalevel,

Fig. 4. A simple OODB schema

Schema	
OID	Name
s3	OODB Schema

Class		
OID	Name	Schema
cl1	Employee	s3
cl2	Department	s3

Field				
OID	Class	Name	Type	Sch.
f1	cl1	EmpNo	int	s3
f2	cl1	Name	string	s3
f3	cl1	Salary	int	s3
f4	cl2	DeptNo	int	s3
f5	cl2	DeptName	string	s3

ReferenceAttribute				
OID	Name	Class	ClassTo	Schema
ref1	Membership	cl1	cl2	s3

Fig. 5. The dictionary for a simple OODB model

that of models—quadrant mM. We consider the horizontal extension here, and the other in the next section.

As we mentioned in Section 2, a crucial issue in our approach is the use of the *supermodel*, a model that includes all the others as special cases, and can therefore be used to describe schemas in all models. The supermodel includes a rather limited set of constructs, to which the various model-specific constructs correspond. In the cases we just showed, Entity in the ER model and Class in OODB correspond to the same metaconstruct, called Abstract. The supermodel unifies in its tables all the dictionaries for the various models, by merging the tables whose constructs refer to the same metaconstruct. Entity and Class are merged, as are Attribute and Field. The schemas we saw in Figures 2 and 4 are represented in the SM portion of the dictionary by the tables shown in Figure 6. Here we see six tables rather than the nine in Figures 3 and 5. It is important to note that the number stays limited. In our current implementation we have ten tables for the SM part and more than fifty in the M part. The former number is stable whereas the latter could grow if new models are added.

The main benefit of this dictionary organization is that we can specify our translations in a simpler way, by referring to metaconstructs rather than to constructs. This is especially effective given the relational structure of the dictionary, as we write our rules in a variant of Datalog [2]. Notice that some properties in the supermodel are not used in some of the models. Also, properties need not have the same name in different models. For example, the IsId property of AttributeOfAbstract is not used in the Field table of the OO model and is used, but with the name IsKey, in the ER model. In fact, we have a null value,

Schema		
OID	Name	Model
s1	1st ER Schema	m1
s2	2nd ER Schema	m1
s3	OODB Schema	m2

Abstract		
OID	Name	Schema
e1	Supplier	s1
e2	Product	s1
e3	Department	s1
e4	Employee	s2
e5	Group	s2
cl1	Employee	s3
cl2	Department	s3

AttributeOfAbstract					
OID	Abstract	Name	Type	IsId	Sch.
a1	e1	Code	int	true	s1
a2	e1	SName	string	false	s1
...	...	...	...	...	...
a8	e3	EN	int	true	s2
...	...	...	...	...	...
f1	cl1	EmpNo	int	?	s3
f2	cl1	Name	string	?	s3
...	...	...	...	...	...

AggrOfAbstract		
OID	Name	Schema
r1	Supply	s1
r2	Membership	s2

ComponentOfAggrOfAbstract					
OID	AggrOfAbs	Abs	isOpt	isFunct	Sch.
c1	r1	e1	false	false	s1
c2	r1	e2	true	false	s1
c3	r1	e3	true	false	s1
c4	r2	e4	false	true	s2
c5	r2	e5	true	false	s2

ReferenceAttributeOfAbs				
OID	Name	Abs	AbsTo	Schema
ref1	Membership	cl1	cl2	s3

Fig. 6. A portion of the SM part of the dictionary

denoted by a question mark '?', for column IsId in tuples that correspond to fields of our OO model. We revisit this point in the next section.

4 The metalevel

The second direction of abstraction of the basic dictionary is its metalevel, that is, the description of the structure of the dictionary itself. Essentially, this level stands with respect to the dictionary in the same way as the dictionary stands with respect to the actual data. As we have two components at the schema level (M and SM), we also have two components at the metalevel, the mM part, which describes the structure of M, and the mSM part, which describes the structure of SM. These components of the dictionary are shown in Figures 7 and 8, respectively. The structure that is shown is complete. The content refers to the specific, simplified dictionaries of the previous section, shown in Figures 6 (for the supermodel) and 3 and 5 (for the models).

In Figure 8, each row in the MSM_Construct table describes a metaconstruct, by means of a unique identifier (the OID column), a Name for the metaconstruct (which is also unique, but not really used as an identifier), and a boolean property isLexical, used to indicate whether the constructs corresponding to the

MM_Model	
OID	Name
m1	ER
m2	OODB

MM_Construct				
OID	Name	Model	MSM-Constr	IsLex
co1	Entity	m1	mc1	false
co2	AttributeOfEntity	m1	mc2	true
co3	Relationship	m1	mc3	false
co4	ComponentOfRelationship	m1	mc4	false
co5	Class	m2	mc1	false
co6	Field	m2	mc2	true
co7	ReferenceAttribute	m2	mc5	false

MM_Property				
OID	Name	Constr	Type	MSM-Pr
pr1	Name	co1	string	mp1
pr2	Name	co2	string	mp2
pr3	IsKey	co2	bool	mp3
pr4	Name	co3	string	mp4
pr5	IsOpt	co4	bool	mp5
pr6	IsFunct	co4	bool	mp6
pr7	Name	co5	string	mp1
pr8	Name	co6	string	mp2
pr9	Name	co7	string	mp7

MM_Reference					
OID	Name	Constr	IsPartOf	ConstrTo	MSM-Ref
ref1	Entity	co2	true	co1	mr1
ref2	Rel'p	co4	true	co3	mr2
ref3	Entity	co4	false	co1	mr3
ref4	Class	co6	true	co5	mr1
ref5	Class	co7	true	co5	mr4
ref6	Cl.To	co7	false	co5	mr5

Fig. 7. The mM part of the dictionary

MSM_Construct		
OID	Name	IsLex
mc1	Abstract	false
mc2	AttributeOfAbstract	true
mc3	AggregationOfAbstract	false
mc4	ComponentOfAggrOfAbstract	false
mc5	ReferenceAttributeOfAbs	false

MSM_Precedence	
Predecessor	Successor
mc1	mc2
mc1	mc3
mc3	mc4
mc1	mc5

MSM_Property			
OID	Name	Constr	Type
mp1	Name	mc1	string
mp2	Name	mc2	string
mp3	IsId	mc2	bool
mp4	Name	mc3	string
mp5	IsOpt	mc4	bool
mp6	IsFunct	mc4	bool
mp7	Name	mc5	string

MSM_Reference				
OID	Name	Constr	IsPartOf	ConstrTo
mr1	Abstract	mc2	true	mc1
mr2	Aggregation	mc4	true	mc3
mr3	Abstract	mc4	false	mc1
mr4	Abstract	mc5	true	mc1
mr5	AbstractTo	mc5	false	mc1

Fig. 8. The mSM part of the dictionary

metaconstruct have (visible) values or not. Hull and King [14] define as *lexical* the constructs that have values. In Figure 8, the rows of MSM_CONSTRUCT list some of the main constructs we have defined, specifically, those needed for the constructs shown in the previous examples. The metamodel we are experimenting with contains a few additional metaconstructs, referring to aggregations of lexicals and their components, foreign keys over them (for handling value based models, especially the relational model), generalizations and their components, and a few others for handling nested structures. Note that "Abstract" is not lexical, as instances of the corresponding constructs (for example "Entity"), have no actual values directly associated with them. By contrast, "AttributeOfAbstract" is lexical, because the instances of its constructs, such as "AttributeOfEntity", do have values.

The rows in MM_CONSTRUCT (in Figure 7) correspond closely to those in MSM_CONSTRUCT. They describe constructs in the various models, each with a unique identifier (the OID column), a reference to the model (Model, a foreign key referencing the MM_MODEL table, which lists all the models currently stored in the system), a reference to the corresponding metaconstruct (MSM-Constr, a foreign key to MSM_CONSTRUCT) and a Name for the construct, unique within the model. The first row in MM_CONSTRUCT states that there is a construct with name "Entity," belonging to the "ER" model (as "m1" is the OID for such a model in MODEL), corresponding to the "Abstract" metaconstruct (as "mc1" is the OID for "Abstract" in MSM_CONSTRUCT).

We can see here how mSM and mM belong to a metalevel: each tuple in MSM_CONSTRUCT corresponds to a table in SM and vice versa and the same is the case for MM_CONSTRUCT and the tables in M.

The other two tables in mSM, MSM_PROPERTY and MSM_REFERENCE, describe some details of metaconstructs (and, as a consequence, of constructs), which in turn correspond to the structure of the tables in SM. The properties describe the value columns in the tables of the dictionary: each property has a Name and a Type, plus a unique OID and a reference to the construct it belongs to, Construct. For example, the second row in MSM_PROPERTY says that each "Attribute of Abstract" ("mc2" is the OID for "Attribute of Abstract") has property "Name," of type string. The third row says that each "Attribute of Abstract" also has a boolean one, "IsId." Thus, table ATTRIBUTEOFABSTRACT in SM has a string column Name and a boolean column IsId, corresponding to these properties.

References in table MSM_REFERENCE describe the columns in the dictionary tables that contain identifiers of other constructs. Specifically, each reference has a name, a construct it belongs to (Construct), a constructs it refers to (ConstructTo) and a boolean IsPartOf, which we will discuss soon. The second and third row in MSM_REFERENCE say that each "Component of Aggregation of Abstracts" ("mc4" for Construct) has a reference named "Aggregation" to an "Aggregation of Abstract" ("mc3" for ConstructTo in the second row) and a reference named "Abstract" to an "Abstract" ("mc1" for ConstructTo in the third row). Again, this describes features of the schema level dictionary. The

table ComponentOfAggregationOfAbstract has two references, one to AggregationOfAbstract and the other to Abstract.

The boolean IsPartOf specifies whether the construct which is the source of the reference is a component of the target and has no autonomous existence. This notion is used as a criterion for the automatic construction of reports for schemas in the various models. For example, the first and the second tuple have value "true" for IsPartOf, to specify that each "Attribute Of Abstract" is part of an "Abstract" and that each "Component of Aggregation of Abstract" is part of an "Aggregation of Abstract." The third row has "false," to specify that each "Component of Aggregation of Abstracts" has a reference to an "Abstract," without being part of it. Clearly, if a construct has multiple references, at most one of them can have "true" for IsPartOf. Otherwise, the occurrences of this construct would be required to be "physical" components of two different objects at the same time.

The table MSM_Precedence specifies that a Successor construct can appear in a model only if a Predecessor one also appears. For example, a model can have "Aggregation of Abstract" only if it also has "Abstract." Even if most precedences follow from references, this need not be the case. In the cases shown in the figure, we have four precedences. The one we just commented on could not be inferred from references. In the mathematical sense, MSM_Precedence is a partial order, so a construct cannot be an indirect predecessor of itself. This is used by our model definition tool to allow for the definition of constructs in the proper order: we can introduce a concept only after its predecessor(s) have been defined.

In summary, the structure of the SM dictionary is completely described by the content of the mSM dictionary and can therefore be generated automatically out of it. Apart from the Schema table, SM contains a table for each row of MSM_Construct. Let c be the metaconstruct in one such row. The table for c has the following columns:

- the "service" columns OID and Schema
- one column for each row of MSM_Property that has c as the value for Construct
- one column for each row of MSM_Reference that has c as the value for Construct; this column is a foreign key reference to the table for the ConstructTo construct in the same row
- a Type column if the row of MSM_Construct has the value "true" for IsLexical

In the tool we are developing [2], the structure of SM is automatically generated once mSM is defined. Suitable features have been defined for changing it when needed, without losing the contents.

Tables MM_Property and MM_Reference in mM play the same role as MSM_Property and MSM_Reference play in mSM, in the sense that they describe the properties and references of the constructs in the various models. Each table of the M dictionary can be generated from mM in the same way as the tables of SM can be generated from mSM: one table for each row in

MM_CONSTRUCT, with “service” columns plus those indicated in MM_PROPERTY and MM_REFERENCE.

Notice the relationships between the tables in mM (Figure 7) and mSM (Figure 8). A conceptual schema for the mM and mSM components of the dictionary is shown in Figure 9. A row in MM_CONSTRUCT (Figure 7) states that there is a construct, with a given Name, in a certain Model (foreign key to MM_MODEL), corresponding to a MSM-Construct (foreign key to MSM_CONSTRUCT). A row in MM_PROPERTY states that a Construct has a property with a Name and a Type. Each property here is a specialization of a property in mSM. This can be expressed by the following constraint: for each row p in MM_PROPERTY, there is a row p' in MSM_PROPERTY such that p.Construct is a construct whose metaconstruct is p'.Construct (that is, there is a row c in MM_CONSTRUCT such that c.OID=p.Construct and c.MSM-Construct=p'.Construct), and p.Type=p'.Type. A similar constraint holds between MM_REFERENCE and MSM_REFERENCE.

The various conditions on the tables in mM are enforced by the process used for defining models and populating the tables in mM. A model is defined by means of a set of constructs each of which is associated with a metaconstruct. At the same time, there cannot be two constructs in a model corresponding to the same metaconstruct. For each construct, one can define a property corresponding to each of the properties of the metaconstruct and a reference for each of the metaconstruct's references. However, in general, properties of metaconstructs need not all be used in the constructs that correspond to them. For example, “Attribute of Abstract” has a “IsKey” property, which is not used by “Field” in the OO model, whereas it is used in the ER model. This is the metalevel description of what we already commented on at the end of Section 3 regarding the schema level. Both properties and references can change their name from mSM to mM. For properties, the case we have in the examples seen so far is “IsId” which becomes “IsKey.” In the examples all references change their names, as they correspond to the names of the constructs rather that those of the metaconstructs.

Notice that the structure and the content of both MM_PROPERTY and MM_REFERENCE are redundant. Some of the information in them could be derived from that in MSM_PROPERTY and MSM_REFERENCE, respectively. This is clearly the case for columns Construct and Type in MM_PROPERTY and Construct, IsPartOf, and ConstructTo in MM_REFERENCE. However, in this way they can be used directly to generate the structure of the tables for the M dictionary, which is indeed their main use.

Also, it is worth noting that most of the tables in mM and mSM could be pairwise merged, as they are very similar: for example MSM_CONSTRUCT and MM_CONSTRUCT could be merged. This would lead to a “self-describing type system,” so that code developed in the tool to operate on user-defined types would also operate on system-defined types. This would be true for example for display methods in an interactive tool and for reporting features. Here we preferred to show them separately, to emphasize the symmetry with the schema level. Another reason is that the process for defining a model reads values in

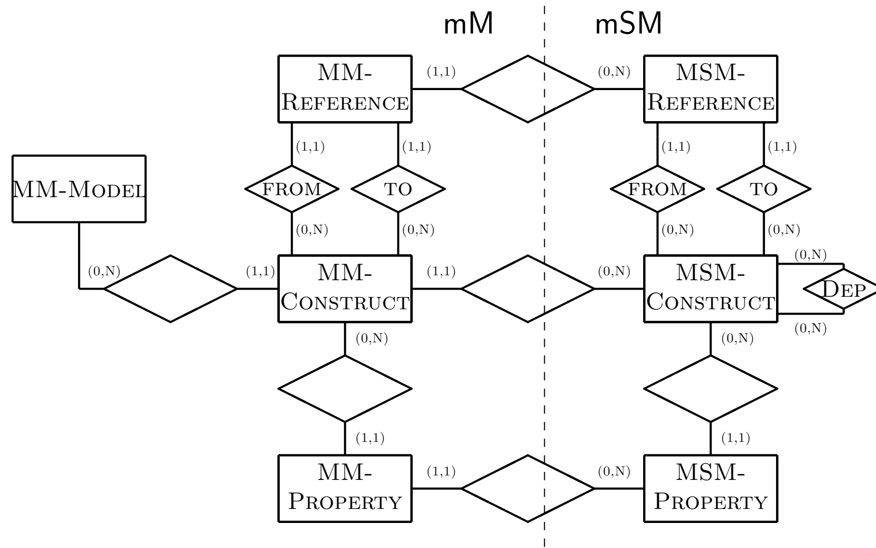

Fig. 9. A simplified ER-schema of the metalevel of the dictionary

mSM and inserts new ones into mM, and we wanted to emphasize the different roles. At the implementation level, the two tables could be merged, or even with separate tables, the code that accesses them could be shared.

5 Reporting

Whenever we have schemas in a model, we are interested in producing *reports* for them. Such reports give detailed textual documentation of the organization of schemas, in a readable and machine-processable way. This issue is delicate here, because we handle many different models, each with its own special features. Each model has its own specific aspects that it could be important to highlight. Probably, in order to provide a suitable emphasis for the specific aspects of each model, we would need a reporting facility expressly defined for it. Still, in a framework that allows for the definition of many different models, a general, flexible way of producing reports would be highly desirable.

Our metamodel approach gives the basis for a model independent reporting feature. Each model involves a set of constructs, each with properties and references. Properties are directly associated with constructs. For example, the Name of an Entity in the ER model or the Type of a Field in the OO model. References relate constructs, in at least two different ways. In some cases, the reference specifies that a given construct is so tightly associated with another construct that it is indeed a component. For example, in the ER model, each AttributeOfEntity is a component of an Entity and each ComponentOfRelationship is a component of Relationship. In other cases, the connection is looser, essentially an external reference to another construct. This is the case

for the reference between ComponentOfRelationship and Entity. As we discussed in Section 3, we have the boolean IsPartOf exactly to distinguish these two types of references.

Using this idea, we have developed a simple, effective algorithm for producing reports which associates components with the constructs they belong to. It takes into account a topological order over constructs induced by MSM_Precedence and presents occurrences of constructs according to that order. A topological order over constructs always exists, as MSM_Precedence is a partial order (as we said in Section 3). For example, constructs that are not components of other constructs (i.e., *base* constructs) are presented first. Precedences are actually defined in mSM but can be extended in a straightforward way to each model in mM. Occurrences of constructs that are incomparable according to the topological order are clustered. In principle, the presentation order of such clusters is arbitrary. In the current implementation, they are presented in the order in which the constructs were defined in the model.

The reports are produced in XML, so that they are both self-documenting and machine processable if needed, for example, for better presentation by means of style sheets. Let $c_1, \ldots, c_k$ be a topological ordering for the base constructs in the model. The algorithm considers the constructs in order, $c_1, \ldots, c_k$, and, for each c_i outputs the description of each of its occurrences o, with the form:

> element named c_i with attributes including OID and properties of o,
> with the following optional subelements
>   if o has components, an element `<components>`, with subelements
>   that are the (recursive) descriptions of the component occurrences
>   if o has references with "false" for IsPartOf
>   the referenced object with its properties (but not the components)

The names of the constructs themselves come from the metamodel and therefore the generation is indeed model-independent.

The structure of the report for an ER schema, according to the simple version of the model we saw in Section 3, would be the following, with some syntactic sugar to make it more readable:

```
<schema name="schemaName" model="modelName">
  <constructs>
    <constructName listOfProperties>
      <components> (if any)
        <constructName listOfProperties>
          ...
        </constructName>
        ...
      </components>
    </<constructName>
    <constructName listOfProperties>
    ...
  </constructs>
</schema>
```

Therefore, the report for the first ER schema shown in Figure 2 would be the following:

```
<schema OID="s1" name="1st ER Schema" model="ER">
 <constructs>
  <entity OID="e1" name="Supplier">
   <components>
    <attributeOfEntity OID="a1" name="Code" isKey="true" type="int"/>
    <attributeOfEntity OID="a2" name="SName" isKey="false"
         type="string"/>
    <attributeOfEntity OID="a3" name="City" isKey="false" type="string"/>
   </components>
  </entity>
  <entity OID="e2" name="Product">
   <components>
    <attributeOfEntity OID="a4" name="ProdNo" isKey="true" type="int"/>
    <attributeOfEntity OID="a5" name="Name" isKey="false" type="string"/>
   </components>
  </entity>
  <entity OID="e3" name="Department">
   <components>
    <attributeOfEntity OID="a6" name="DeptNo" isKey="true" type="int"/>
    <attributeOfEntity OID="a7" name="DeptName" isKey="false"
          type="string"/>
   </components>
  </entity>
  <relationship OID="r1" name="Supply">
   <components>
    <componentOfRelationship OID="c1" isOpt="false" isFunct="false">
     <entity OID="e1" name="Supplier"/>
    </componentOfRelationship>
    <componentOfRelationship OID="c2" isOpt="true" isFunct="false">
     <entity OID="e2" name="Product"/>
    </componentOfRelationship>
    <componentOfRelationship OID="c3" isOpt="true" isFunct="false">
     <entity OID="e3" name="Department"/>
    </componentOfRelationship>
   </components>
  </relationship>
 </constructs>
</schema>
```

We experimented with stylesheets for reports generated in this way. They are reasonably effective, even when produced in a completely model-independent way. For example, a convenient way to produce reports is hypertext based, where a single HTML page is built for a schema, with internal links that make it easy to navigate from one construct to another. Lists of constructs of the various types can also be easily generated.

6 Conclusions

The structure of the dictionary we have shown here is being used in the tool under development. It supports a variety of activities in a model independent way. Beside the generation from the core, it supports reporting, as we illustrated, and generation of parametric formats for import and export of schemas. Overall, the availability of a dictionary with a visible structure is very useful in the development of translation rules and in the maintenance of the tool itself. In particular, changes to the model do not require changes to the tool's engine.

These are consequences of the main novelty of our approach which offers an integrated and highly correlated representation of schemas and models, both in a model specific way and within a unified framework, the supermodel.

References

1. F. W. Allen, M. E. S. Loomis, and M. V. Mannino. The integrated dictionary/directory system. *ACM Comput. Surv.*, 14(2):245–286, 1982.
2. P. Atzeni, P. Cappellari, and P. A. Bernstein. Modelgen: Model independent schema translation. In *ICDE*, IEEE Computer Society, pages 1111-1112, 2005.
3. P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Databases: concepts, languages and architectures.* McGraw-Hill, 1999.
4. P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. *EDBT, Lecture Notes in Computer Science 1057*, Springer, pages 79–95, 1996.
5. P. Atzeni and R. Torlone. Mdm: a multiple-data-model tool for the management of heterogeneous database schemes. *SIGMOD*,ACM, pages 291–301, 1997.
6. C. Batini, G. D. Battista, and G. Santucci. Structuring primitives for a dictionary of entity relationship data schemas. *IEEE Trans. Software Eng.*, 19(4):344–365, 1993.
7. P. Bernstein, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, and D. Shutt. Microsoft repository version 2 and the open information model. *Information Systems*, 22(4):71–98, 1999.
8. P. A. Bernstein. Applying model management to classical meta data problems. *CIDR*, pages 209–220, 2003.
9. P. A. Bernstein, A. Y. Halevy, and R. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.
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. K. T. Claypool and E. A. Rundensteiner. Sangam: A framework for modeling heterogeneous database transformations. In *ICEIS (1)*, pages 219–224, 2003.
12. 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.
13. C. Hsu, M. Bouziane, L. Rattner, and L. Yee. Information resources management in heterogeneous, distributed environments: A metadatabase approach. *IEEE Trans. Software Eng.*, 17(6):604–625, 1991.
14. R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, Sept. 1987.
15. B. K. Kahn and E. W. Lumsden. A user-oriented framework for data dictionary systems. *DATA BASE*, 15(1):28–36, 1983.
16. S. Melnik. *Generic Model Management: Concepts and Algorithms.* Springer-Verlag, 2004.
17. E. Rahm and H. Do. On metadata interoperability in data warehouses. Technical report, University of Leipzig, 2000.
18. R. Soley and the OMG Staff Strategy Group. Model driven architecture. White paper, draft 3.2, Object Management Group, November 2000.
19. 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.