MDM: a Multiple-Data-Model Tool for the Management of Heterogeneous Database Schemes*

Paolo Atzeni
Dipartimento di Informatica e Automazione
Università di Roma Tre
Via Vasca Navale, 84 — 00146 Roma, Italy
atzeni@inf.uniroma3.it

Riccardo Torlone
Dipartimento di Informatica e Automazione
Università di Roma Tre
Via Vasca Navale, 84 — 00146 Roma, Italy
torlone@iasi.rm.cnr.it

Abstract

MDM is a tool that enables the users to define schemes of different data models and to perform translations of schemes from one model to another. These functionalities can be at the basis of a customizable and integrated CASE environment supporting the analysis and design of information systems. MDM has two main components: the *Model Manager* and the *Schema Manager*. The Model Manager supports a specialized user, the *model engineer*, in the definition of a variety of models, on the basis of a limited set of *metaconstructs* covering almost all known conceptual models. The Schema Manager allows designers to create and modify schemes over the defined models, and to generate at each time a translation of a scheme into any of the data models currently available. Translations between models are automatically derived, at definition time, by combining a predefined set of elementary transformations, which implement the standard translations between simple combinations of constructs.

1 Introduction

With respect to the representation of data in the analysis phase (the conceptual design activity [5]), current CASE tools usually offer one specific data model. Although usually these models are presented as *the* "Entity-Relationship model" [6], it is in fact the case that there are many versions of it, and each tool adopts a different version. A similar observation can be made with respect to methodologies: each of them adopts a different conceptual model. As a consequence, it is common to have a context where different (maybe similar) data models have to be handled at the same time, for a number of different reasons: (i) a methodology is chosen and an independent tool is avalaible, and their models differ; (ii) the various designers of a complex project prefer to work with their favorite models, but their work has to be exchanged, reused and integrated; (iii) specific subproblems are handled with different models, suitable with their characteristics, and (iv) the results of independent design activities have to be integrated (a need that may arise when companies merge or get involved in a federated project).

We believe that a natural way to try to overcome this problem is the design of *extensible* systems that allow the user to customize the environment with the definition of specific models and the support in the translation between them. Recently, we have proposed a formal framework that allows the specification of conceptual data models by means of a suitable formalism called a *metamodel* [2, 3, 4]. Then, for any two models M_1 and M_2 defined in this way, and for each scheme S_1 (the *source scheme*) of M_1 (the *source model*), it is possible to obtain a scheme S_2 (the *target scheme*) that is the *translation* of S_1 into M_2 (the *target model*). The proposal is based on two major observations, as follows.

First, it has been noted that the constructs used in most known models can be classified by means of a rather limited set of categories [8] (lexical type, abstract type, aggregation, generalization, function and a few others). Therefore, a metamodel can be defined by means of a basic set of *metaconstructs*, corresponding to the above categories. In this framework, a model can be defined by indicating which metaconstructs (or versions thereof) its constructs correspond to. It can be observed that this approach is not universal, as it does not cover all possible models. However, it is easily extensible: should a model with a completely new construct be proposed, the corresponding type could be introduced in the metamodel, and then used in the definition of the model.

The second point is that there is no clear notion of when a translation is correct. In fact, a lot of research has been conducted in the last decades on scheme equivalence with reference to the relational model [1, 7, 12] or to heterogeneous frameworks [9, 10, 11], but there is no general, agreed definition. Therefore, we followed a pragmatic approach. We assume that the constructs that correspond to the same metaconstruct have the same semantics, and then we define translations that operate on individual constructs (or simple combinations thereof) as follows: for each construct x of the source scheme such that there is no construct of the same type in a target model M, we try to replace x by other constructs which are instead allowed in M. This work is supported by the use of a predefined set of elementary transformations, which implement the standard translations between con-

*This work was supported by Università di Roma Tre, MURST, and Consiglio Nazionale delle Ricerche.

To Appear in *ACM-SIGMOD Conference*, Tucson, Arizona, May 1997

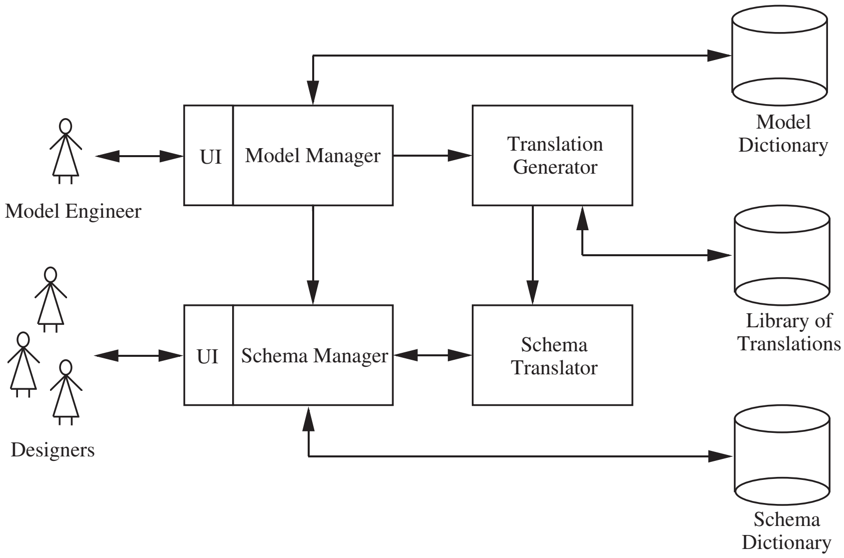

Figure 1: The architecture of the MDM tool

structs studied in the literature [5] (which we assume to be correct by definition). Thus, a complex translation can be obtained just as composition of elementary steps.

As a preliminary step, we have developed a graph-theoretic framework that allows us to define in an uniform way schemes and models [3]. Using this description we have been able to define and characterize desirable properties of translations, and to develop general methodologies for the automatic generation of translations that satisfy such properties.

On the basis of these results, we have defined and built a first prototype of the system, over which we are testing the various features of the approach in an important case which involves the various versions of the Entity-Relationship model and a few representatives of other categories of models. The tool is implemented in Visual C++ under Windows95, and it is currently being ported to Motif in a Unix environment. In the next section we describe the architecture of the tool and the functions of the variuos modules. In Section 3, we briefly discuss an example.

2 A description of the tool

On the basis of theoretical results and practical algorithms described in [3, 4], we have designed a tool, called MDM (Multiple Data Models), for the management of multiple models and the translation of schemes.

The architecture of the MDM tool is shown in Figure 1. Before examining its main components, it is useful to discuss the classes of specialists involved with the various activities. The main user of the tool would be a traditional *designer* (or analyst) interested in describing the data of interest for an application by means of a specific model, among those available. The current set of models is managed by a special user, called the *model engineer*, who has the responsibility of setting the framework. He/she defines (and modifies) models by using metaconstructs and takes care of the translations between models (with the support of tool, as we will see). Finally, we could envision the role of a *metamodel engineer*, responsible for extending the whole system, with the addition (or modification) of metaconstructs and basic translation steps. Given the specificity of this last role, the tool provides a direct support only for users in the first two classes.

The main operations offered by the tool to designers and model engineers are the following:

1. The definition of a model by means of a (menu-driven) "Model Definition Language". This language has been designed according to a metamodel that involves (at the moment) the following metaconstructs: lexical types, abstract types, functions, binary and n-ary aggregations and generalizations between abstracts. The task is to be conducted by the model engineer, whose work is supported by a number of menus (for choosing the appropriate type of construct between the available metaconstructs and specifying its features). When a new model M is defined, the system automatically generates a *default translation* from the "supermodel" to M (we elaborate on this aspect shortly). This translation is later used to translate any scheme to the model M.

2. The specification of a scheme belonging to a model by means of a menu-driven "Schema Definition Language". This language is automatically provided with the definition of a model. Specifically, after the definition of a new model, designers can select this model from a menu and build a new schema by choosing among the constructs available for it. This work is supported by a fexible graphical interface.

3. The request for a translation of a scheme into a specific target model. The system satisfies the request by applying a customized version of the default translation associated with the target model.

This task is also autonomously conducted by designers.

Let us briefly illustrate the main components of the MDM tool.

User Interface. This part allows the interaction with the system by means of a graphical (as well as textual) language. At the moment the interface is quite primitive but we are currently testing a tool that allows the editing and the automatic layout of complex diagrams. With this tool, it is possible to customize edges and nodes. This is very useful in our context since, using this feature, the users can also specify their preferred diagram style.

Model Manager. It takes as input data model specifications done with respect to the metamodel, and stores them in a Model Dictionary. The Model Dictionary contains all the data models defined by the model engineer together with a special model (the *supermodel*), which subsumes every other model. The supermodel is automatically generated by the Model Manager by finding the *least upper bound* of the models currently stored in the Model Dictionary [3]. This model is the model of reference for generating schema translations. The system is able to store, together with a model description, further informations like special constraints on the application of the constructs in a specific model.

Schema Manager. In a similar way as the Model Manager, this component takes as input the specification of a new scheme S of a model M stored in the Model Dictionary, checks whether S belongs to the model M and, if so, stores S in a Schema Dictionary. The Schema Dictionary is the repository of schemes and can store different versions of the same scheme obtained after modifications and/or translations of the original scheme. Again, additional information can be stored together with a scheme, such as integrity constraints that cannot be expressed with the scheme itself.

Translation Generator. This module generates new translations between pairs of models, on the basis of a set of predefined basic translations permanently stored in the Library of Translations. The computed translations can be modified by the model engineer. All the translations generated by this module can be stored (according to a request done by the Model Manager) in the Library of Translations (for later use).

Schema Translator. It actually executes translations of schemes, by applying the appropriate translation generated by the Translation Generator, to a source scheme received by the Schema Manager. The output scheme is returned to the Schema Manager to be stored in the Schema Dictionary or displayed through the user interface.

The various components of MDM co-operate as follows.

1. When the model engineer defines a new model M, the Model Manager first stores it in the Model Dictionary and then checks whether the supermodel subsumes the new model or not. In the first case, the Model Manager sends a request to the Translation Generator for the generation of the *default translation* from the supermodel to M, which will be stored in the Library of Translations. In the latter case, the Model Manager generates a new supermodel that replaces the previous one in the Model Dictionary. Then, a request is sent to the Translations Generator for generating translations from the new supermodel to every other model stored in the Model Dictionary. Those new translations replace the old default translations in the Library of Translations.

2. When a designer defines a new scheme S for a model M, the Schema Manager verifies whether S is allowed in M, by matching S with the definition of M, which is stored in the Model Dictionary. If the matching is successful, the scheme can be stored in the Schema Dictionary. Schemes can also be modified (by saving old versions if necessary) and deleted.

3. When a designer submits a request for the translation of a scheme S from a source model M_s to a target model M_t, the Schema Translator loads from the Library of Translations, through the Translation Generator, the default translation for the model M_t. This translation is certainly correct, but, because it is a translation from the supermodel to M_t and the supermodel is in general more complex than M_s, it may include redundant steps. Therefore, the translation is first optimized by deleting unnecessary steps and then applied to S. The result is displayed and eventually stored in the Schema Dictionary.

If, during the various activities, it turns out that the metamodel is not enough expressive for describing a new data model or that the basic translations used to build more complex translations are not sufficient or satisfactory, then the metamodel engineer has to be called for extending the tool.

We are currently testing the capabilities of the tool in a restricted but important case. We have taken in account the various versions of the Entity-Relationship model that can be found described in the literature or implemented in the systems (with or without generalizations, with binary or n-ary relationships, with simple or composite domains, and so on), a version of the Functional Model and the Relational Model. At the moment, we have stored 10 models and 25 basic translations (among them: the translation of n-ary relationships into binary ones, the translation of generalizations into binary relationships, the elimination of composite attributes, the translations of functions between abstracts into relationships, and so on). In this situation, the tool is able to translate schemes from each model to any other thus confirming that the provided set of basic translations is indeed complete.

3 An example of application

In this section we briefly present a practical example of application of the tool. We will consider two models M_s and M_t (both of them are indeed different versions of the E-R model) and derive a translation from M_s to M_t. Then, this translation will be applied to a specific schema of M_s. The model M_t is a version of the E-R model that involves binary relationships and entities with simple and/or multivalued attributes (that is, attributes whose instances are sets of values). The model M_s is instead

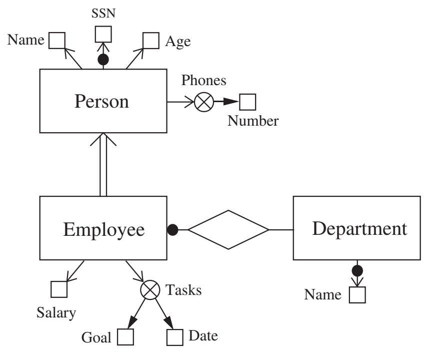

Figure 2: A schema for the model M_s

a version of the E-R model involving n-ary relationships, entities with simple and/or composite attributes (that is, attributes whose instances are sets of tuples of values), and is-a relationships between entities. The translation from M_s to M_t requires the following basic steps:

1. The translation of n-ary relationships into binary ones;
2. The translation of is-a relations between entities into relationships on entities (actually, other translations could be applied here);
3. The translation of composite attributes with only one component into multivalued attributes;
4. The translation of composite attributes into new entities;
5. The translations of functions between entities into relationships on entities (this function is needed to eliminate a side-effect produced by step 4 as clarified below).

Now consider the scheme of the model M_s in Figure 2. The scheme represents persons and employees. Employees have a salary and work in departments having a name. Tasks with specific goals, to be executed within a certain date, are assigned to employees. This is represented by means of a composite attribute of the entity Employee.

By applying the translation described above, we obtain the scheme reported in Figure 3. Actually, the first step does not produce any effect on the scheme since the relationships in original scheme are already binary (an optimization task before the execution of the translation in fact eliminates useless steps such as this). The second step translates the is-a relation between the entities Person and Employee in a one-to-one relationship on them. The third step translates the composite attribute Phones of the entity Person in a multivalued attribute, whereas the fourth step translates the composite attribute Tasks of the entity Employee in a new entity. This step generates an undesired side-effect: a function from the entity Employee to the entity Tasks, which is a construct not allowed in the target model. This construct is eliminated in the last step by replacing it with a one-to-many relationship between the involved entities.

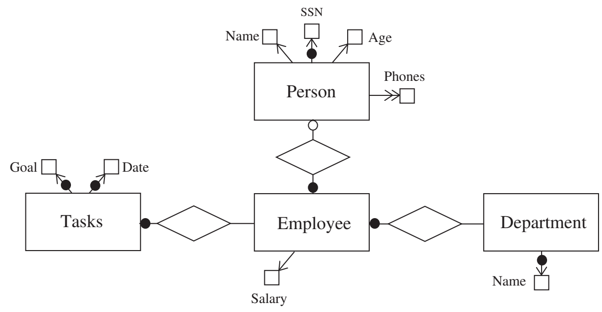

Figure 3: The result of a translation into the model M_t

References

[1] P. Atzeni, G. Ausiello, C. Batini, and M. Moscarini. Inclusion and equivalence between relational database schemata. *Theoretical Computer Science*, 19(2):267–285, 1982.

[2] P. Atzeni and R. Torlone. A metamodel approach for the management of multiple models and the translation of schemes. *Information Systems*, 18(6):349–362, 1993.

[3] P. Atzeni, R. Torlone. Schema Translation between Heterogeneous Data Models in a Lattice Framework. In *Sixth IFIP TC-2 Working Conference on Data Semantics (DS-6), Atlanta*, pages 218–227, 1995.

[4] P. Atzeni, R. Torlone. Management of Multiple Models in an Extensible Database Design Tool. In *EDBT '96, LNCS 1057*, Springer–Verlag, pages 79–95, 1996.

[5] C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach.* Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.

[6] P.P. Chen. The entity-relationship model: toward a unified view of data. *ACM Trans. on Database Syst.*, 1(1):9–36, March 1976.

[7] R.B. Hull. Relative information capacity of simple relational schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.

[8] R.B. Hull and R. King. Semantic database modelling: survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[9] L.A. Kalinichenko. Methods and tools for equivalent data model mapping construction. In *EDBT '90, LNCS 416*, pages 92–119, Springer-Verlag, 1990.

[10] Y.E. Lien. On the equivalence of database models. *Journal of the ACM*, 29(2):333–362, 1982.

[11] R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *Eighteenth International Conf. on Very Large Data Bases, Dublin*, 1993.

[12] J. Rissanen. On equivalence of database schemes. In *ACM SIGACT SIGMOD Symp. on Principles of Database Systems*, pages 23–26, 1982.