Management of Multiple Models in an Extensible Database Design Tool[⋆]

Paolo Atzeni[1] and Riccardo Torlone[2]

[1] Terza Università di Roma, c/o DIS, Via Salaria 113, 00198 Roma, Italy
[2] Terza Università di Roma, c/o IASI–CNR, Viale Manzoni 30, 00185 Roma, Italy

Abstract. We describe the development of a tool, called MDM, for the management of multiple models and the translation of database schemes. This tool can be at the basis of an integrated CASE environment, supporting the analysis and design of information systems, that allows different representations for the same data schemes. We first present a graph-theoretic framework that allows us to formally investigate desirable properties of schema translations. The formalism is based on a classification of the constructs used in the known data model into a limited set of types. Then, on the basis of formal results, we develop general methodologies for deriving "good" translations between schemes and, more in general, between models. Finally, we define the architecture and the functionalities of a first prototype that implements the various features of the approach.

1 Introduction

During the past decade, the availability and use of automated tools for the analysis and development of information systems have rapidly increased. It has been observed however that, although these tools provide significant benefits to their users with productivity gains, current systems still present various limitations. Hence, a new generation of database design tools is currently under definition and development with the goal of extending functionalities and improving usability [6].

One important reason for the gap between user expectations and reality is the so-called *impedance mismatch* between methodologies and tools [6, Chapter 15], that is, the differences between the model for a given methodolgy and the model effectively supported by a specific CASE tool. A natural way for overcoming this problem is the design of *extensible* systems that support multiple data models and manage the translations of schemes from one model to another. The possibility of customizing the environment (with the definition of a model suitable for a given methodology) is a big step towards the solution of the impedance mismatch problem. Moreover, the availability of different, custom-defined models and their interaction is useful for a number of reasons: (i) to let each designer work with his/her favorite model, yet allowing the exchange, reuse

[⋆] This work was partially supported by Consiglio Nazionale delle Ricerche and by MURST.

and integration of their work, (ii) to tackle different subproblems with different models, suitable with the specific aspects of each, and (iii) to integrate the results of independent design activities (a need that may arise when companies merge or get involved in a federated project).

The goal of our research is the definition of an environment that allows the specification of conceptual data models by means of a suitable formalism called a *metamodel*. 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 should be possible to obtain a scheme S_2 (the *target scheme*) that be the *translation* of S_1 into M_2 (the *target model*). The solution we have proposed in our preliminary study [3] is based on the following points:

- Since all the constructs used in most known models fall in a rather limited set of categories [11] (lexical type, abstract type, aggregation, generalization, function and a few others) a metamodel can be defined by means of a basic set of *metaconstructs*, corresponding to the above categories. Then, a model can be described by defining its constructs by means of the metaconstructs in the metamodel. It can be argued that this approach is not "complete", as it does not cover all possible models, but it is however easily extensible: should a model with a completely new construct be proposed, the corresponding type could be introduced in the metamodel.
- Since there is no clear notion of when a translation is correct (a lot of research has been conducted in the last decades on scheme equivalence with reference to the relational model [2, 10, 15] or to heterogeneous frameworks [1, 12, 13, 14], but there is no general, agreed definition) we follow 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 constructs studied in the literature [6] (which we assume to be correct by definition)[3]. Thus, a complex translation can be obtained just as composition of elementary steps.

To the best of our knowledge, there is not much literature related to the problem we tackle and the goal we set. Some work exists on the idea of a metamodel for the representation of models [5, 12], but the goal is more on the integration of heterogeneous databases in a federated environment [16] than on the translation of schemes to generic target models.

The approach has been studied within a graph-theoretic framework that allows us to define in an uniform way schemes and models [4]. In this framework a model M is defined by means of a set of *patterns* $\mathcal{P}$, directed graphs whose

[3] The approach could be called "axiomatic": this is coherent with the difficulty in defining correct translations.

nodes have different types (corresponding to the basic metaconstructs we mentioned above, such as lexical, abstract, aggregation and function) and edges have *ranges* as labels. A partial order can be introduced on patterns, which, suitably extended, becomes a lattice on sets of patterns. Elementary transformations are described on the basis of the patterns they eliminate and the patterns they introduce: clearly, this is only part of their description (we say this is the *signature* of a basic translation, as opposed to its *implementation*), but it is sufficient for studying general properties of translations. Using this description we are able to define and characterize desirable properties of translations, and to develop general methodologies for the automatic generation of translations that satisfy such properties. The results are obtained in an elegant way by means of the lattice framework on patterns.

On the basis of these results, we have defined functionalities and architecture of a first prototype of the system which is currently under development at University of Rome "La Sapienza". On this system, we are testing the various features of the approach in an important case which involves the various versions of the Entity-Relationship model.

The paper is organized as follows. In Section 2 we informally describe our graph-theoretic framework (presented in [4]). In Section 3, we develop practical methods for deriving translations between models and between schemes of different models. In Section 4, we discuss operational issues and present the architecture of a first prototype of the system. In Section 5 we show a brief example of application of our methodology. Finally, in Section 6, we draw some conclusions.

2 A formal approach to the problem

2.1 Structures and Patterns

We have introduced a graph-theoretic formalism that allows us to define in an uniform way schemes and models [4]. In this framework, the components of the metamodel are represented by a fixed set of node types $\mathcal{N}$ and a fixed set of edge types $\mathcal{E}$. In the following, we will refer to a (simple) metamodel that consists of three types of nodes, corresponding to *abstracts* (denoted by the symbol $\triangle$), *aggregations* ($\otimes$), and *lexicals* ($\square$); and six types of edges, corresponding to *functions* (denoted by $\rightarrow$), *multivalued functions* ($\twoheadrightarrow$), *components of aggregation* ($\longrightarrow\!\!\triangleright$), *keys of aggregation* ($-\!\!\bullet\!\!\triangleright$), *keys of abstract* ($-\!\!\bullet\!\!\rightarrow$) and *subset relations between abstracts* ($\Rightarrow$). We point out however that the approach can handle a variety of metamodels [4].

Two main notions have been introduced for describing schemes and models: the notion of a *structure*, a directed acyclic graph whose nodes and edges are elements of the metamodel, and the notion of a *pattern*, a tree whose nodes and edges are elements of the metamodel and where the edges have *ranges* as labels. Roughly speaking, a range denotes the number of times a certain edge can appear in a structure. Thus, a pattern describes a collection of structures that involve a specific composition of metaconstructs. A mapping between structures and sets of patterns can be easily defined, so that, given a set of patterns $\mathcal{P}$ and a structure S, we can verify whether S is an *instance* of (that is, can be described by) $\mathcal{P}$. The set of all instances of a set of patterns $\mathcal{P}$ is denoted by $Inst(\mathcal{P})$.

Figure 1 shows an example of a set of patterns and Figure 2 one of its instances (n is a parameter denoting a fixed integer). The instances of this set of patterns may be: (1) isolated abstracts with key combined with (monovalued or multivalued) functions from abstracts to lexicals (pattern P_1), and (2) (one-to-many or many-to-many) binary aggregations of abstracts combined as above (patterns P_2 and P_3).

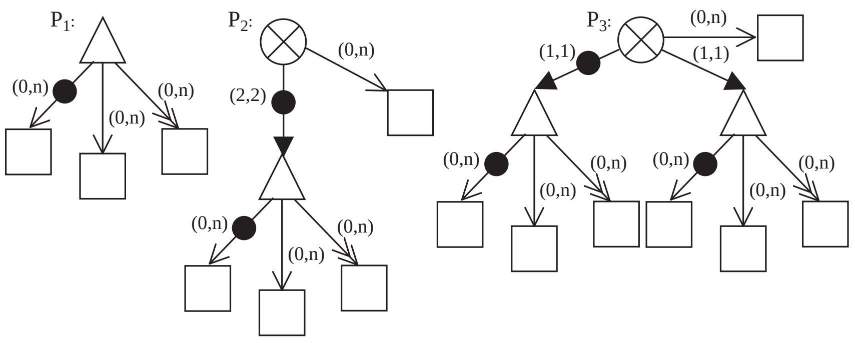


Fig. 1. A set of patterns describing a version of the E-R model.

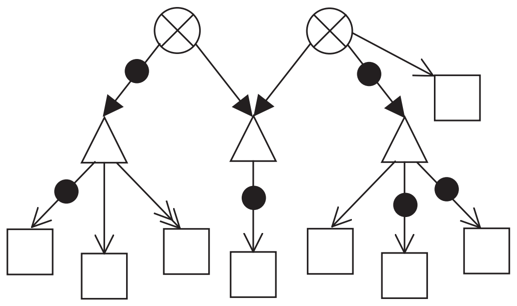


Fig. 2. A structure that is an instance of the set of patterns in Figure 1.

A natural partial order relationship $\preceq$ can be defined on sets of patterns, which yields a practical way to test whether a set of patterns $\mathcal{P}_1$ is *subsumed* by another set of patterns $\mathcal{P}_2$, that is, whether the set of all instances of $\mathcal{P}_1$ is contained in the set of all instances of $\mathcal{P}_2$ (in symbols $Inst(\mathcal{P}_1) \subseteq Inst(\mathcal{P}_2)$). For instance, the set of patterns in Figure 3 subsumes (and therefore describes at least all the structures described by) the set of patterns in Figure 1.

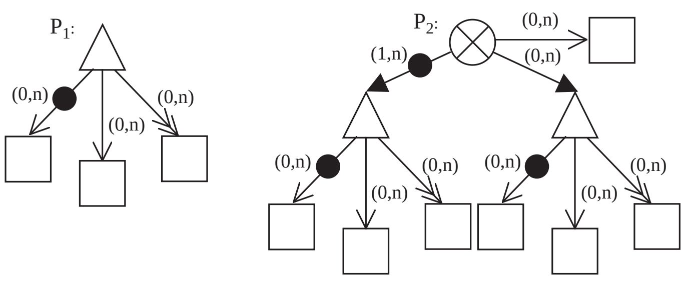


Fig. 3. A set of patterns that subsumes the set of patterns in Figure 1.

Finally, we have shown that the partial order relation $\preceq$ induces a *lattice* on the set of sets of patterns, that is, every finite collection $\mathbf{P}$ of sets of patterns has both a *greatest lower bound* and a *least upper bound*.

2.2 Models and Schemes

In the framework above, a *model* can be defined by a set of patterns $\mathcal{P}$ and by a labeling function γ that maps each element of $\mathcal{N} \cup \mathcal{E}$, occurring in $\mathcal{P}$, to a label. These labels corresponds to the names given to the constructs in a specific model.

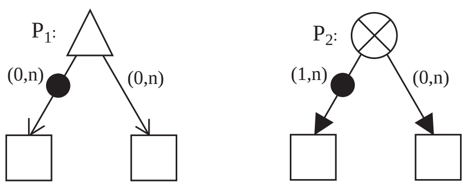
Let us consider for instance the patterns in Figure 1. If we associate the label "Entity" to the node $\triangle$, the label "Relationship" to the node $\otimes$, the label "Domain" to the node $\square$, the label "simple attribute" to the edge $\rightarrow$ and the label "multivalued attribute" to the edge $\twoheadrightarrow$, we define a version of the Entity-Relationship model (E-R for short) involving binary relationships on entities which can have simple and/or multivalued attributes.

Similarly, a *scheme* is defined by a structure S and by a labeling function λ that maps each node and each edge of S to a label. The labels associated with the components of a scheme correspond to names of the various concepts represented in the scheme (e.g., *persons*, *books* and so on).

It is important to note that in our approach, the definition of scheme is completely independent of the notion of model. Clearly, it is possible to establish a correspondence between schemes and models: a scheme $\$ = (S, \lambda)$ is *allowed* in a model $M = (\mathcal{P}, \gamma)$ if $S \in \mathit{Inst}(\mathcal{P})$. It is easy to see that, on the basis of the subsumption relationship, we can always verify whether a scheme $\$$ is allowed in a model M.

2.3 Schema translations

A *schema translation* τ is a function that operates on structures by replacing constructs with other constructs. The behavior of a translation can be effectively described in our framework by a pair of patterns $\sigma = (P_1, P_2)$ (we say this is the *signature* of a basic translation, as opposed to its implementation or *body*). Intuitively, a translation signature represents: (1) the constructs eliminated by τ and (2) the constructs introduced by τ, as effect of its execution. For instance, the following translation signature:



represents a translation that replaces abstracts and (optional) functions from abstracts to lexical (e.g., an entity of the E-R model with its attributes), with an aggregation on lexicals (e.g., a relation of the relational model). Note that, also in this case, a translation signature is independent of a specific model. A *translation rule* has the form $\sigma[\tau]$, where τ is a translation function and σ is a translation signature for τ.

A nice property of translation signatures is that they can be used to characterize translations in terms of sets of patterns, that is, we can compute the *effect* of σ on a set of patterns $\mathcal{P}$, denoted by $\sigma(\mathcal{P})$ (intuitively, $\sigma(\mathcal{P}) = \mathcal{P} - \{P_1\} \cup \{P_2\}$), such that, for each $S \in \mathit{Inst}(\mathcal{P})$, it is the case that: $\tau(S) \in \mathit{Inst}(\sigma(\mathcal{P}))$.

In our approach, a complex translation T can be obtained as a composition of a number of predefined *basic* translation rules: $T = \sigma_1[\tau_1], \ldots, \sigma_k[\tau_k]$. These basic translations implement the standard translations between the constructs present in the traditional data models (e.g., from an entity of the Entity-Relationship model to a relation of the relational model, or from a n-ary relation to a set of binary ones) [3]. The effect of the execution of a complex translation T on a set of patterns $\mathcal{P}$, denoted by $\sigma_T(\mathcal{P})$, can be easily computed as the composition of the effects of the components of T.

2.4 Formal properties of schema translations

Using the results described above, we can formally verify the *correctness* of a translation T from a model $M_s = (\mathcal{P}_s, \gamma_s)$ to a model $M_t = (\mathcal{P}_t, \gamma_t)$, that is, the fact that the application of T to any scheme of M_s always generates a scheme allowed in M_t. Indeed, T is a correct translation from M_s to M_t if and only if $\sigma_T(\mathcal{P}_s) \preceq \mathcal{P}_t$.

One of the major result in [4] is that, as a consequence of the lattice framework, given a *set* of models $\mathcal{M}$, there is no need to specify a translation for each pair of models in $\mathcal{M}$ since, for each model $M_t = (\mathcal{P}_t, \gamma_t) \in \mathcal{M}$, it is sufficient to look a translation from $\mathcal{P}_\top$ (the least upper bound of the sets of patterns describing the models in $\mathcal{M}$) to $\mathcal{P}_t$: this translation will be correct from any model in $M_s \in \mathcal{M}$ to M_t. Thus, the number of required translations is linear in terms of the number of involved models, rather than quadratic. It could be said that the set of patterns $\mathcal{P}_\top$ represents a *supermodel* containing all the possible combination of constructs used in the various models in $\mathcal{M}$.

Intuitively, a set of basic translation rules $\mathcal{R}$ is *complete* with respect to a set of models $\mathcal{M}$ if it is possible to find a correct translation T, using the rules in $\mathcal{R}$, from any pair of models in $\mathcal{M}$. Another important result in [4] is that we can test completeness by verifying the existence of translations from the supermodel (see above) to the *minimal models* (the models whose description is subsumed by the description of any other model).

We can also formally define a natural measure of the "quality" of a translation from one model to another. Given two different correct translations T_1 and T_2 from a source model $M_s = (\mathcal{P}_s, \gamma_s)$ to a target model $M_t = (\mathcal{P}_t, \gamma_t)$, T_1 *is preferable* than T_2 if $\sigma_{T_2}(\mathcal{P}_s) \preceq \sigma_{T_1}(\mathcal{P}_s) \preceq \mathcal{P}_t$. Intuitively, a translation is preferable than another if the effect of its execution is "closer" to the target model. For instance, a translation towards a version of the E-R model with n-ary relationships that is able to generate both binary and ternary relationships is preferable than another translation that generates only binary relationships.

Finally, two reasonable notions of "optimization" can be defined for translations. A correct translation T from a source model M_s to a target model M_t is *minimal* if there is no rule R in T such that $T - \{R\}$ is correct and preferable to T. A correct translation T from a source model M_s to a target model M_t is *optimal* if there is no other correct translation T' from M_s to M_t, such that T' is preferable to T.

3 Generating translations

On the basis of the formal results on translations, we present in this section a number of practical algorithms for deriving correct and (possibly) optimal translations between models. We will refer to a set of basic translations $\mathcal{R}_b$, which we assume to be predefined, but we will not assume that this set is complete. Then, the algorithm we propose can be also used to test for completeness as described in the previous section.

3.1 Preliminaries

We will start by proposing a method for deriving *reductions*, that is translation between models described by sets of patterns $\mathcal{P}_s$ and $\mathcal{P}_t$ such that $\mathcal{P}_t \preceq \mathcal{P}_s$.[4] Indeed, this is not a restrictive hypothesis since, by the results on the lattice framework, for any pair of models M_1 and M_2 which are not comparable, a translation from their least upper bound (which, by definition, subsumes both

[4] Actually, a reduction may contain steps introducing new patterns, but at the end, it always generates a set of patterns that is subsumed by the original set.

of them) to M_1 (M_2) is also a correct translation from M_2 to M_1 (from M_1 to M_2).

A very general method for generating a reduction from $\mathcal{P}_s$ to $\mathcal{P}_t$ consists in selecting rules that eliminate patterns of $\mathcal{P}_s$ which are not allowed in $\mathcal{P}_t$. Unfortunately, this cannot be done naively since the order in which the rule are selected is crucial. In fact, it may happen that a rule that eliminates a certain pattern P is selected before a rule that eliminates another pattern but, as a side effect, introduces P again. A *monotonic* reduction is one in which if it is never the case that a pattern is eliminated in one step and introduced again in a subsequent step. It turns out that this property can be verified locally, by analyzing the set of rules at disposal. Given a set of rules $\mathcal{R}$, the analysis needs the construction of a graph $G_{\mathcal{R}}$, called *precedence graph* of $\mathcal{R}$, such that the nodes represent the rules in T and there is an edge from a rule R_i to a rule R_j if R_i introduce a pattern which is deleted by R_j. If the graph $G_{\mathcal{R}}$ does not contain any cycle, the set $\mathcal{R}$ is *serializable*. Then, it is possible to show that a *serialization* of $\mathcal{R}$ (that is, a translation T based on an order of the rules in $\mathcal{R}$ that satisfies the partial order induced by $G_{\mathcal{R}}$) is monotonic.

3.2 Reductions

The first algorithm we propose is based on the assumption that the set of rule of reference $\mathcal{R}_b$ is serializable. We will later relax this hypothesis by refining the algorithm. This assumption is indeed very useful since we have proved in [4] that if a set of rules $\mathcal{R}$ is serializable, then any serialization based on $\mathcal{R}$ produces the same effect and therefore, in this case, the order in which the rule are selected is immaterial. It follows that the basic algorithm is quite simple.

```
Algorithm REDUCTION-SET-1
Input: A pair of models $M_s = (\mathcal{P}_s, \gamma_s)$ and $M_t = (\mathcal{P}_t, \gamma_t)$ such that $\mathcal{P}_t \preceq \mathcal{P}_s$
            and a set of basic rules $\mathcal{R}_b$.
Output: A set of rules $\mathcal{R}_{out} \subseteq \mathcal{R}_b$.
begin
{Part 1: search for a correct translation}
    $\mathcal{R}_{out} := \emptyset; \mathcal{P} := \mathcal{P}_s$;
    for each $R \in \mathcal{R}_b$ do
        if $R$ deletes patterns in $\mathcal{P}$ that are not in $\mathcal{P}_t$
        then $\mathcal{R}_{out} := \mathcal{R}_{out} \cup \{R\}$; $\mathcal{P} := \mathcal{P} \cup$ {patterns introduced by $R$} endif;
    until $\mathcal{P} \preceq \mathcal{P}_t$ or all the rules in $\mathcal{R}_b$ have been selected;
    if $\mathcal{P} \not\preceq \mathcal{P}_t$ then return($\emptyset$) and exit;
{Part 2: search for a minimal rule set}
    for each rule $R \in \mathcal{R}_{out}$ do
        $\mathcal{P}' :=$ the effect of $\mathcal{R}_{out} - \{R\}$ on $\mathcal{P}_s$;
        if $\mathcal{P} \preceq \mathcal{P}' \preceq \mathcal{P}_t$
        then $\mathcal{R}_{out} := \mathcal{R}_{out} - \{R\}; \mathcal{P} := \mathcal{P}'$ endif;
    endfor
end.
```

Given a pair of models $M_s = (\mathcal{P}_s, \gamma_t)$ and $M_t = (\mathcal{P}_t, \gamma_t)$, in the first step the above algorithm selects every rule of $\mathcal{R}_b$ whose effect deletes patterns in $\mathcal{P}_s$, which are not in $\mathcal{P}_t$. At each step, new patterns eventually introduced by selected rules are added to $\mathcal{P}_s$. At the end, if the effect of the selected rules on $\mathcal{P}_s$ (we can speak of effect of a *set* of rules for the property mentioned above) is a set of patterns subsumed by $\mathcal{P}_t$, then the set $\mathcal{R}_b$ is not complete, the algorithm interrupted and the empty set is returned. In the second step, a minimal translation is derived from the set of rules selected in the first step by deleting "redundant" rules (if any), that is, rules whose elimination produce a preferable translation.

Now, assume that the set of rule $\mathcal{R}_b$ is not necessarily serializable. The algorithm can be slightly modified by assuming, in searching for a correct translation, that the selected set of rules is serializable (and so the first two steps of the algorithm are not modified) and then verifying, a posteriori, the serialization of the obtained set of rules. If this set is not serializable, the algorithm is recursively executed over the set of rules $\mathcal{R}_b - \{R\}$, where R is one of the rules that causes the set to be non-serializable. This is summarized in the following general algorithm.

```
**Algorithm** REDUCTION-SET-2
**Input:** *A pair of models* $M_s = (\mathcal{P}_s, \gamma_s)$ *and* $M_t = (\mathcal{P}_t, \gamma_t)$ *such that* $\mathcal{P}_t \preceq \mathcal{P}_s$
        *and a set of rules* $\mathcal{R}_b$.
**Output:** *A set of rules* $\mathcal{R}_{out} \subseteq \mathcal{R}_b$.
**begin**
**{Part 1: search for a correct translation}**
**{Part 2: search for a minimal rule set}**
**{Part 3: search for a serializable rule set}**
    $\mathcal{R}_c := \{$rules in $\mathcal{R}_{out}$ involved in a cycle in $G_P\}$;
    **while** $\mathcal{R}_{out}$ *is not serializable* **and** $\mathcal{R}_c \neq \emptyset$ **do**
        *pick a rule* $R$ *from* $\mathcal{R}_c$;
        $\mathcal{R}'_{out}$ := REDUCTION-SET-2($\mathcal{P}_s, \mathcal{P}_t, \mathcal{R}_c - \{R\}$);
        **if** $\mathcal{R}'_{out} \neq \emptyset$ **then** $\mathcal{R}_{out} := \mathcal{R}'_{out}$
    **endwhile**;
    **if** $\mathcal{R}_{out}$ *is not serializable* **then return**($\emptyset$) **and exit**;
**end.**
```

Finally, the algorithm can be further refined for achieving optimality. Similarly to the second algorithm, this can be done by applying the algorithm recursively to the set of rules $\mathcal{R}_b - \{R\}$, where R is a rule that deletes patterns which are indeed in the target model. The rationale under this choice is that there could be "finer" functions which are able to replace the work done by R and which do not require the deletion of patterns in the target model.

3.3 Model translations

According to the properties on translations between models described in Section 2, the general algorithm for deriving model translations is the following:

```
**Algorithm** MODEL-TRANSLATION
**Input:** A pair of models $M_s = (\mathcal{P}_s, \gamma_s)$ and $M_t = (\mathcal{P}_t, \gamma_t)$
        and a set of rules $\mathcal{R}_b$.
**Output:** A correct translation from $M_s$ to $M_t$.
**begin**
    $\mathcal{P}_\top :=$ the least upper bound of $\mathcal{P}_s$ and $\mathcal{P}_t$;
    **if** $\mathcal{R}_b$ is serializable
    **then** $\mathcal{R}_{out} :=$ REDUCTION-SET-1$(\mathcal{P}_\top, \mathcal{P}_t, \mathcal{R}_b)$
    **else** $\mathcal{R}_{out} :=$ REDUCTION-SET-2$(\mathcal{P}_\top, \mathcal{P}_t, \mathcal{R}_b)$;
    **if** $\mathcal{R}_{out} \neq \emptyset$
    **then** $T_{out} :=$ a serialization of $\mathcal{R}_{out}$;
    **return**$(T_{out})$ **endif**;
**end.**
```

3.4 Schema translations

Let $\$$ be a scheme of a certain model M_s and assume we want to translate $\$$ in another model M_t. Also, let T be a correct and optimal translation from M_s to M_t. Actually, before applying T to $\$$, we can *refine* T adapting the translation to the scheme, by deleting basic steps of T_M that are "useless", that is, steps operating on constructs allowed in M_s, but not used in $\$$. This can be easily done by comparing the constructs of $\$$ and the signatures of the basic translations occurring in T (cf. Section 2). Then, we have the following general algorithm for translating schemes.

```
**Algorithm** SCHEMA-TRANSLATION
**Input:** A scheme $\$ = (S, \lambda)$ of a model $M_s = (\mathcal{P}_s, \gamma_s)$
        and a model $M_t = (\mathcal{P}_t, \gamma_t)$;
**Output:** A scheme $\$_{out} = (S_{out}, \lambda)$ allowed in $M_t$.
**begin**
    $T :=$ MODEL-TRANSLATION$(M_s, M_t, \mathcal{R}_b)$;
    **for each** rule $R$ occurring in $T$ **do**
    **if** $R$ has a null effect on $S$
    **then** $T := T - \{R\}$;
    $S_{out} := \tau_T(S)$;
    **return**$(\$_{out})$;
**end.**
```

Clearly, from a practical point of view, the translation between models are computed once for all, at definition time, as described in the next section.

4 Implementation issues

On the basis of the theoretical results and the practical algorithms described in the previous sections, we have designed a tool (whose architecture is reported in

Figure 4), called MDM (Multiple Data Models), for the management of multiple models and the translation of schemes. More specifically, the operations offered by this tool 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 of defining models should not be done by any user, but rather by a specialist that we call *model engineer*. His work is supported by a number of menus (for choosing the appropriate type of construct between the available metaconstructs) and forms. When a new model M is defined, the system automatically generates a *default translation* from the supermodel to M (see below).
2. *The specification of a scheme belonging to a model (previously defined) by means of a graphical "Schema Definition Language"*. This language is automatically provided with the definition of a model. More specifically, there is a predefined graphical language for describing schemes that is expressive enough for any scheme allowed in the metamodel. Then, the SDL for a certain model M is obtained by tailoring this general language to the features of M. The work of defining schemes is supported by a flexible graphical interface.
3. *The request for an ad-hoc translation from a source model M_s to a target model M_t*. The translation is permanently stored and can be later applied to any scheme belonging to M_s.
4. *The request for a translation of a scheme into a specific target model*. The system satisfies the request by applying the default translation for the target model, or an ad-hoc translation previously computed if one exists.

The MDM tool consists of the following main components (see Figure 4).

– A Graphical User Interface. It allows the interaction with the system by means of a graphical (as well as textual) language. We have used for this component *Diagram Server* [8, 9], a tool developed at the University of Rome "La Sapienza" 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. The GUI also transforms schemes and models from their *external representation* (diagrams) into an *internal representation* (and vice versa) whose structures have been designed according to the notions of *structure* and *pattern*, respectively.
– A Model Manager. It takes as input data model specifications done with respect to the metamodel, and store them in a Model Dictionary. The Model Dictionary contains all the data models defined by the model engineer together with a special model, called *supermodel* (SM), that subsumes each other model. The supermodel is automatically generated by the Model Manager by finding the least upper bound of the sets of patterns describing the

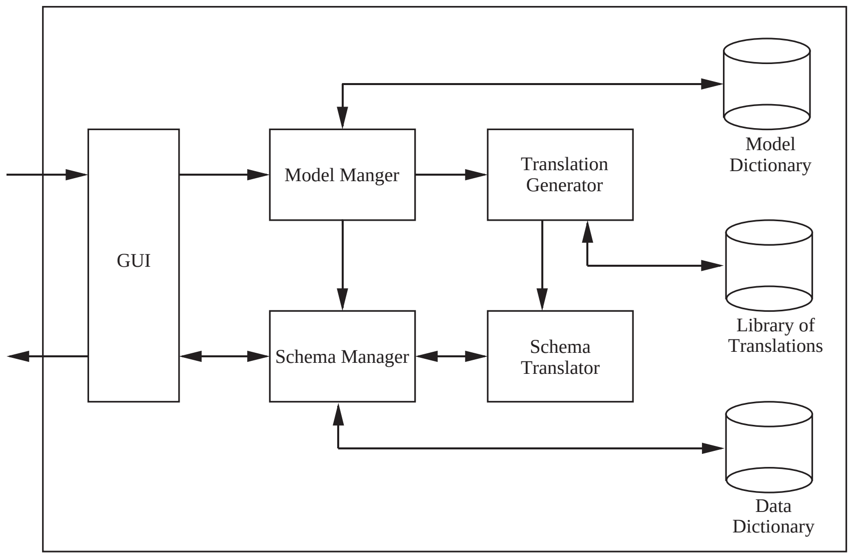


Fig. 4. The architecture of the MDM tool.

models in the Model Dictionary (cf. Section 2). According to the results in Section 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 of the model that cannot be described with the notion of pattern.

- A Schema Manager. Similarly to the Model Manager, this component takes as input the specification of a new scheme $\mathbb{S}$ of a model M stored in the Model Dictionary, checks whether $\mathbb{S}$ belongs to the model M (according to the definition of Section 2) and, if so, stores $\mathbb{S}$ 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 (this relationship between schemes is implemented by means of suitable *scheme identifiers*). Also in this case, a number of information can be stored together with a scheme like integrity constraints that cannot be expressed with the scheme itself.
- A Translation Generator. This module generates new translations between pairs of models, on the basis of a set of predefined basic translations $\mathcal{R}_b$ permanently stored in the Library of Translations. More specifically, it implements Algorithms Reduction-set and Model-translation described in Section 3. The computed translations can be modified by the model engineering. 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).

– A 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. Thus, the module implements Algorithm SCHEMA-TRANSLATION described in Section 3. The output scheme is returned to the Schema Manager to be stored in the Schema Dictionary or displayed through the GUI. Also in this case, the users can modify the generated scheme.

The various components of MDM co-operate as follows.

1. When a new model M is defined, the Model Manager first checks whether SM subsumes M or not. In the former case, the Model Manager stores M in the Model Dictionary and sends a request to the Translation Generator for the generation of the *default translation* (a reduction in this case) from the SM to M which will be stored in the Library of Translations. In the latter case, the Model Manager stores M and generates a new supermodel SM' that replaces SM in the Model Dictionary. Then, a request is sent to the Translations Generator for translations from the new supermodel SM' to *each* other model stored in the Model Dictionary. Those new translations replace the old default translations in the Library of Translations. This is indeed a quite complex task that however should not be very frequent. Actually, this work can be avoided by permanently storing in the Model Dictionary a predefined supermodel that is the *most general model* we can define with the given metamodel. However, with this approach, the quality of translations is surely degraded (since they are generated with respect to a model that is, in many cases, too general).
2. When a new scheme S for a model M is defined, the Schema Manager checks whether S is allowed in M (cf. Section 2) by matching S with the definition of M, which is stored in the Model Manager. If the matching is successful, the scheme can be stored in the Data Dictionary. The schemes can also be modified (by saving the old versions if necessary) and deleted.
3. When a user submits a request for a translation of a scheme S to a model M, the Schema Translator loads from the Library of Translations, through the Translation Generator, either the default translation T_M for the model M or an ad-hoc translation (if any). Then, the translation is applied according to the algorithm described in Section 3.

During the various activities performed by the tool, some problems may arise. First, it may happen that the metamodel is not enough expressive for describing a new data model. However, metaconstructs and other features of the metamodel are stored in special files accessible by the system only. This files can be updated quite easily without modifying the code of the components of the tool. Moreover, it can be the case that the basic translations used to build more complex translations are not sufficient for a certain translation. This can be solved by adding new basic translations in the Library of Translations, as well as modifying the old once. It turns out that MDM is easily adaptable and provides a very flexible framework for the development of more complex and general environments.

5 An example of application

In this section we briefly present a practical example of application of methodologies and tools described in the previous sections.

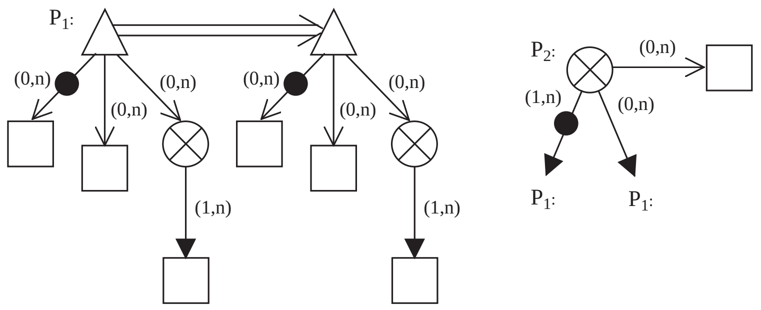


Fig. 5. A set of patterns describing M_s: a version of the E-R model.

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 the one described by the set of patterns in Figure 1. We recall that this model is a version of the E-R model that involves binary relationships on entities which can have simple and/or multivalued attributes (that is, attributes whose instances are sets of values). The model M_s is instead described by the set of patterns reported in Figure 5. It is possible to see that this model is a version of the E-R model involving n-ary relationships on entities, which can have 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:

- The translation of n-ary relationships in binary ones;
- The translation of is-a relations between entities in relationships on entities (actually, other translations could be applied here);
- The translation of composite attributes in new entities;
- The translations of functions between entities with relationships on entities (this function is needed to eliminate a side-effect produced by step 3 as described in the following).

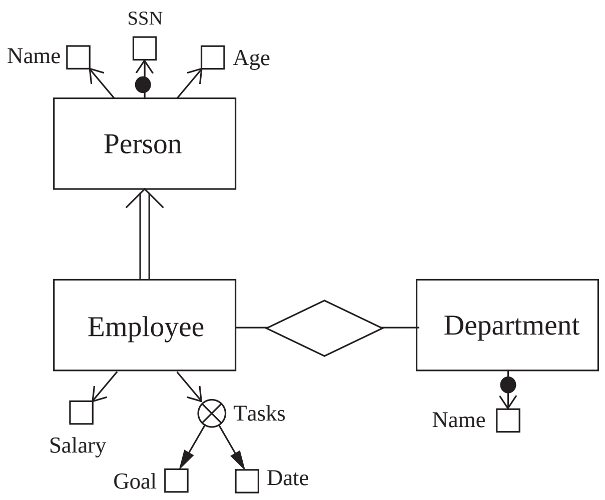


Fig. 6. A schema for the model M_s described by the patterns in Figure 5.

Now consider the scheme of the model M_s in Figure 6. Note that this scheme uses a notation that is slightly different from the notation used to describe the model (specifically, entities are represented by rectangles and relationships between entities are represented by rhombs) but this is coherent with the possibility, offered by the tool, of customizing the diagrammatic notation for the model constructs. The scheme represents persons and employees. The 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 7. Actually, the first step does not produce any effect on the scheme since the relationships in original scheme are already binary (this step can be indeed deleted before the execution of the translation as described in Section 3). The second step translates the is-a relation between the entities Person and Employee in a relationship on them. The third 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 relationship between the involved entities.

6 Conclusions

In this paper we have presented theoretical and practical aspects of the development of MDM: a tool for the management of different data models and the

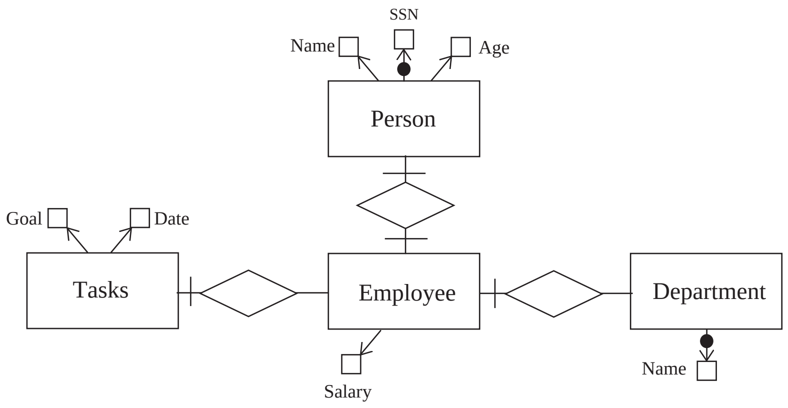


Fig. 7. The result of the application, to the scheme in Figure 6, of the translation from the model M_s (described by the patterns in Figure 5) to the model M_t (described by the patterns in Figure 1).

translations of schemes from one model to another. We have started by presenting a graph-theoretic framework for the description of models and schemes. This formal framework allows us to compare different data models and to define and characterize various interesting properties of schema translations. We have then derived general methodologies for deriving translations that satisfy those desirable properties. On the basis of these results, we have designed architecture and functionalities of a tool that supports the desired features, showing the feasibility of the approach and the practical impact of the formal results.

We believe that this research brings a contribution not only to the development of new generation database design tools, but also to several problems related to cooperative activities within heterogeneous database environments. The formal basis and the prototypical tool yield promising contexts for further theoretical and practical investigations, on these and related issues. For instance, it could be very interesting to extend the approach presented in this paper to behavioral models (e.g. DFD, SADT). Currently, from a theoretical point of view, we are working on extending the results to more general cases. From a practical point of view we are working on the development of a solid prototype of the system by testing its capabilities in complex cases.

References

1. S. Abiteboul and R. Hull. Restructuring hierarchical database objects. *Theoretical Computer Science*, 62(3):3–38, 1988.
2. P. Atzeni, G. Ausiello, C. Batini, and M. Moscarini. Inclusion and equivalence between relational database schemata. *Theoretical Computer Science*, 19(2):267–285, 1982.

3. 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.
4. 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), Atalanta*, pages 218–227, 1995.
5. T. Barsalou and D. Gangopadhyay. M(DM): An open framework for interoperation of multimodel multidatabase systems. In *International Conference on Data Engineering*, pages 218–227, Tempe, AZ, February 1992.
6. C. Batini, S. Ceri, and S.B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach.* Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.
7. P.P. Chen. The entity-relationship model: toward a unified view of data. *ACM Trans. on Database Syst.*, 1(1):9–36, March 1976.
8. G. Di Battista, G. Liotta, and S. Vargiu. Diagram Server. *Journal of Visual Languages and Computing*, 1995. To appear.
9. G. Di Battista et al. A tailorable and extensible automatic layout facility. *IEEE Workshop on Visual Languages*, pages 68–73, 1991.
10. R.B. Hull. Relative information capacity of simple relational schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.
11. R.B. Hull and R. King. Semantic database modelling: survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.
12. L.A. Kalinichenko. Methods and tools for equivalent data model mapping construction. In *EDBT'90 (Int. Conf. on Extending Database Technology), Venezia, Lecture Notes in Computer Science 416*, pages 92–119, Springer-Verlag, 1990.
13. Y.E. Lien. On the equivalence of database models. *Journal of the ACM*, 29(2):333–362, 1982.
14. 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.
15. J. Rissanen. On equivalence of database schemes. In *ACM SIGACT SIGMOD Symp. on Principles of Database Systems*, pages 23–26, 1982.
16. A.P. Sheth and J.A. Larson. Federated database systems for managing distributed database systems for production and use. *ACM Computing Surveys*, 22(3):183–236, 1990.
17. D. Tsichritzis and F.H. Lochovski. *Data Models.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

This article was processed using the LaTeX macro package with LLNCS style