Technical Report No.2006-521

Model transformation via pull-backs: algebra vs. heuristics *

Zinovy Diskin

School of Computing, Queen's University,
Kingston, Ontario, Canada
{zdiskin,dingel}@cs.queensu.ca

Abstract. The paper presents a formal algebraic framework, where model transformation can be specified in a truly generic way: the source and the target metamodels are parameters of the entire procedure. In more detail, the operation is specified as a special diagram operation well known in category theory under the name of pull-back. At the heart of the approach is a mapping between the two metamodels, which governs the entire translation procedure.
An essential feature of the framework is that this mapping is allowed to map elements of one metamodel to elements that can be *derived* in the other metamodel (by posing suitable queries to it) but are *not immediately* present in it. This makes the pattern applicable to a wide range of practically interesting situations.

1 Introduction and motivating discussion

Model transformation (or translation), MT, is a key component of many activities in software development and integration. In databases, for example, it appears in data warehousing (the infamous extract/transofrm/load cycle) and in numerous scenarios of data and schema integration (from federated databases to web data). In programming, if we consider code artifacts as models whose metamodels are usually set by the corresponding grammars, model transformations are everywhere: from model-based code generation to compilation to reverse engineering. In fact, in the MDD vision, programming *is* model transformation. In a sense, software development is all about MT.

The MT-task is formulated as follows: a source model S in some (source) metamodel $\mathbf{M}_S$ is to be transformed into a target model T in another (target) metamodel $\mathbf{M}_T$. Note that we can treat the database (DB) and the programming (language, PL) cases uniformly by simply considering programs as data over grammars-generated schemas (metamodels). However, different use contexts in the two communities generated two essentially different approaches to the main

* Research supported by OCE Centre for Communications and Information Technology and IBM CAS Ottawa.

MT-problem: how to facilitate programming MT-tasks, which are notoriously laborious and error-prone.

In DB, a broad vision of generic model management (MMt) emerged [3] as an environment, where the programmer manipulates models as holistic entities without zooming into their internal elementwise structure (see[10] for a discussion). The idea of MMt is to offer the programmer a powerful arsenal of operations over models, where the operator of model transformation (called model generation, ModelGen, in [2]) is an important yet not the only one (amongst other MMt-operators are, e.g., Model Merge, Match and Extract [2]). A principal schema of ModelGen is shown in Fig. 1(a). Traceability mapping is a mapping from the new model to the old one, it can be considered optional.

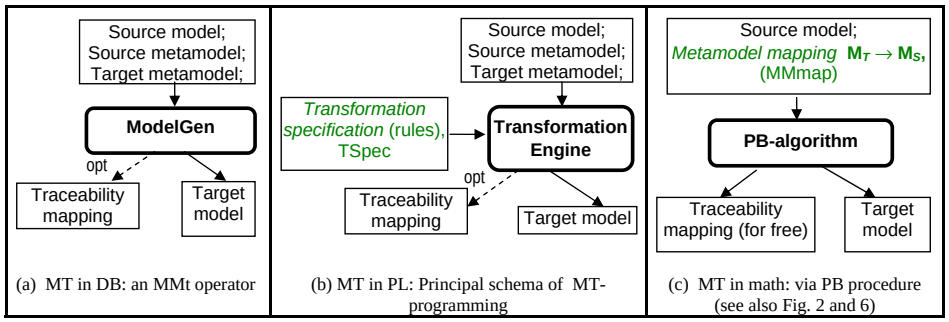


Fig. 1. Three approaches to MT

In PL, model transformation is viewed in a somewhat different way. First of all, it is the only MMt-operator considered in a majority of works, because transformation is the key component of MDD, MDA and other model-centric endeavors, see [5, 16, 11] for surveys and discussion. The entire MMt trend is also represented, see [4] and references therein, although has gained much less attention so far.

A principle schema of MT-programming (adapted from [11]) is presented in Fig. 1(b).The key block is Transformation specification, TSpec. Roughly, it says how each element, or a group of elements closed in some technical sense, in the source model are to be transformed into an element or a group in the target model. TSpec thus amounts to a set of pairs of elements (in the declarative MT-languages), or to a set of transformation rules (in the imperative MT-languages). In either way, the programmer needs to create an *elementwise* specification relating the source and the target models. Since a model usually comprises a big set of different elements and their structural links, the elementwise nature of TSpec makes MT-programming laborious and error-prone. It is these problems in metadata management that have driven the database community to the idea of generic MMt.

Thus, MMt suggests a radically different and seemingly a more efficient approach to programming model operation. However, the key condition of its realization is that all MMt-operators must be specified in a generic (metamodel independent) way. Genericness becomes especially non-trivial for the model translation operator. In [2], Phil Bernstein even questioned the very possibility of specifying this operation in a generic way. Indeed, the few attempts of approaching the task one can find in the database literature are, in fact, attempts to sidestep the problem rather than to solve it. The main idea employed in these works was to catalogue the constructs used in all the dialects of a modeling language and then try to extract a core generic subset from them. A typical example is the "the most general" *super-ER* format in [1]. Though the approach can be efficient within the space of ER-modeling (understood in a broad sense), it will not work beyond the ER-space.

Fortunately, mathematical category theory offers a suitable apparatus for designing generic specifications, particularly, for model translation, see [6] for an outline of its MMt-applications. The present paper illustrates how the machinery works for MT in greater detail.

Briefly, the idea is as follows. If we want to translate models in some (source) metamodel $\mathbf{M}_S$ to models in another (target) metamodel $\mathbf{M}_T$, then the elements of $\mathbf{M}_T$ should be somehow found in $\mathbf{M}_S$. The simplest case is when we have a mapping $m\colon \mathbf{M}_T \to \mathbf{M}_S$ interpreting $\mathbf{M}_T$-constructs by suitable $\mathbf{M}_S$-constructs. However, in practice we rarely have such a simple relationship between the metamodels. As a rule, $\mathbf{M}_S$-counterparts of $\mathbf{M}_T$-constructs are not basic (i.e., immediate) elements of $\mathbf{M}_S$ but can be derived from them by applying to $\mathbf{M}_S$ suitable algebraic operations (queries, in the database jargon). The metamodel relationship is then specified by a mapping $m\colon \mathbf{M}_T \to \mathsf{der}^Q\mathbf{M}_S$ into some augmentation $\mathsf{der}^Q\mathbf{M}_S$ of the source metamodel with derived elements (where Q refers to the set of operatiosn/queries used). (In category theory, such mappings are called *Kleisly* morphisms), see [6] for a popular presentation).

Given such a mapping (MMmap), it can be demonstrated that the result of translating $\mathbf{M}_S$-models into $\mathbf{M}_T$-models could be defined as the result of some algebraic (meta)operation over models and model mappings. This operation, well known in category theory under the name of *pull-back* (PB), is formally defined and appears in many different mathematical and applied contexts. For example, when we consider typed graphs and their transformations, the retyping procedure is given by the corresponding pull-back (see, e.g., [15]).

Model transformation also can be seen as a sort of retyping; however, does this retyping (MT-retyping) equal to retyping provided by the PB-operation (PB-retyping)? Equality that we mean here should be understood in some conventional sense: what is a proper result of model transformation is determined by an application dependant pragmatic context and hence is an informal notion (at least, in the situation when semantics is not taken into account!). In contrast, the result of the PB-operation is perfectly formal and mechanistic. Thus, equality *MT-retyping = PB-retyping* is just a definition, which we can accept (if we believe it is useful) or reject otherwise. The paper aims at careful

motivation of this definition. In other words, the goal is to show that defining model transformation to be the result of the corresponding PB-operation is an adequate algebraic model of MT.

Nevertheless, however reasonable this motivation could be, the final justification is the responsibility of the two Top Judges:

(a) a formal proof that MT-via-PB preserves sets of instances in some still-to-be defined sense, and

(b) an implementation of an MT-via-PB tool and checking its effectiveness.

As for the former, it is a big and largely unexplored issue; in the current state-of-the-art of MMt theory, we cannot even formulate what needs to be proved precisely. Nevertheless, in data modeling a few successful attempts can be mentioned, for example, [13] studies transformation between relational schemas and ER-diagrams with instances essentially taken into account, and [6] describes a general framework for data model and data instance transformations going *together*. Particularly, in [7] this framework is used for building a coherent syntax-semantics theory of model merge: a generic procedure for model merge is proposed and a result *semantically justifying* this procedure is proved. A "big goal" is to build similar semantic justifications for other MMt operators in both data modeling (cf. [14]) and behavior modeling.

As for (b), it is a project waiting for its realization; so far, we can loosely replace it by considering a number of MT-examples recognized as right and then checking that the PB-procedure also provides the right results.

The principle schema of the PB-approach is shown in Fig. 1(c). Note that the TSpec block crucial for MT is not needed: everything is provided by the mapping MMmap. In fact, the PB-algorithm itself generates all the necessary transformation rules from the mapping and then executes them. In this sense, MT-via-PB "programming" is somewhat similar to declarative MT-programming, but there are essential differences. Metamodels are much more compact than models and hence specifying relations between metamodels (mappings) is much less laborious. Also, a mapping between metamodels has a clear semantic meaning of interpreting constructs of one language by constructs of another language. This factor makes metamodel mapping design less error-prone. Finally, perhaps the most dramatic difference (at least, in the data management perspective) is that ordinary MT-programming sees MT as rewriting (updating) the source model, while MT-via-PB amounts to augmenting and retyping the source model (querying). Indeed, TSpec prescribes how to *change* the source model; in the database jargon, it is an update specification. In contrast, MMmap prescribes what derived information must be extracted from the source metamodel (querying), afterwards the PB-procedure operates only types of the source model elements rather than elements themselves. In the database jargon, MMmap is nothing but a view definition and the target model is the corresponding materialized view. We will return to this discussion in section 5.

The rest of the paper is organized as follows. In section 2 we discuss how to specify model translation generically, and how the PB-operation emerges there. Section 3 outlines some mathematical details. Section 4 demonstrates how it

works with two simple examples of extracting ER-diagrams from SQL-table definitions (the first one is unsuccessful but instructive). The culmination is in section 5: a generic algebraic pattern for model translation is presented and discussed.

2 Model translation as an arrow diagram operation

We begin with informal general considerations of what could be a generic pattern for model translation.

2.1 Models as typed graphs

We assume that a model (schema) S is a structure of elements typed by the corresponding elements of the metamodel $\mathbf{M}_S$. An important observation (made by many people but still not too familiar to the community) is that typing can be considered as a structure-preserving mapping (morphism) $\sigma\colon S \to \mathbf{M}_S$ from the model to the metamodel. For example, we can specify the metamodel $\mathbf{M}_S$ by a (directed) graph, and then a model over $\mathbf{M}_S$ is nothing but a graph S together with a graph mapping as above. Following some mathematical traditions, we will direct the typing mappings vertically from top to bottom and say that S is a model *over* $\mathbf{M}_S$.

The left half of Fig. 2 presents a simple relational schema (the left column) and its representation as a typed graph (the column on the right). In the lower part of the representation column, there is a metaschema of a simplified relational data model, presented as a directed graph. Those edges that are not directed are to be considered as syntactic abbreviations for two directed edges going into the opposite directions. The upper part of the left column shows a typed graph representing the relational schema.

Type labels are given after a colon (and in violet color on the display). Since for any two given nodes N and M of the model graph, there is at most one arrow between them typed by a label L, we can safely omit the names of the arrows but keep their labels. If necessary, we can unambiguously refer to such an arrow from M to N by the name *(M-N):L*.

It can be easily checked that typing actually amounts to a graph morphism, that is, a mapping sending nodes to nodes and arrows to arrows in such a way that the incidence relation between nodes and arrows is preserved. Moreover, it is easy to understand that the example is quite generic and any relational schema (with the similar simplifying assumptions) can be presented in this way, that is, by a graph S together with a graph morphism $\sigma\colon S \to \mathbf{M}_S$ into the graph specifying the relational data model.

Similarly, the right half of Fig. 2 presents a ER-diagram as a typed graph (in the rightmost column). The ER-diagrams we consider are first-order: relationships (R-Nodes) are defined only over entity (E-)nodes. The metamodel node *Node* is abstract in the sense that any of its instances is either E-node or R-node. It can be readily checked that the typed graph on the right specified a graph

morphism $\tau: T \to \mathbf{M}_T$ into the corresponding metamodel. These two examples are quite generic and demonstrate how models in different languages can be presented by mappings between graphs. We will return to this discussion later, in section 5.3.

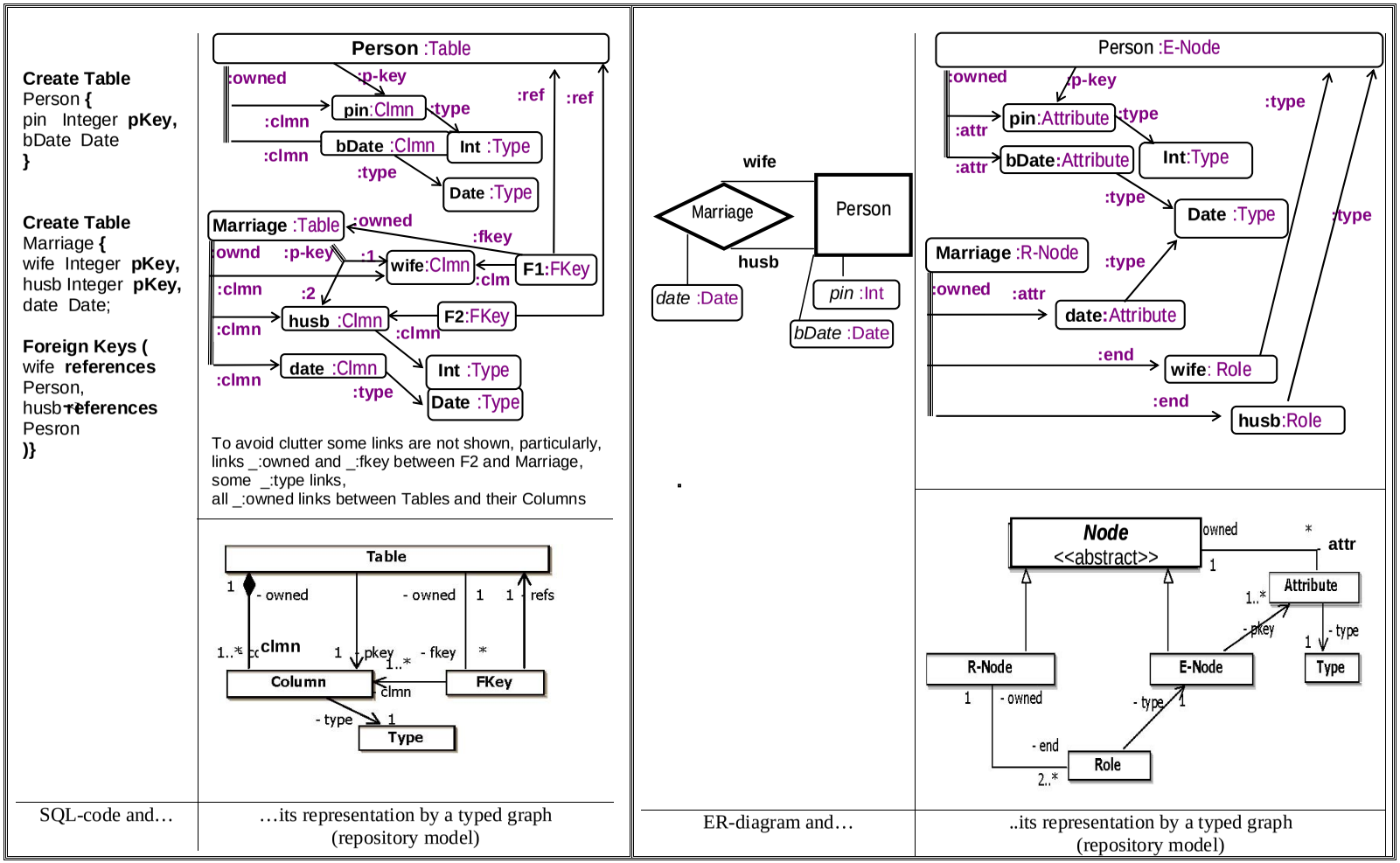


Fig. 2. Models as typed graphs

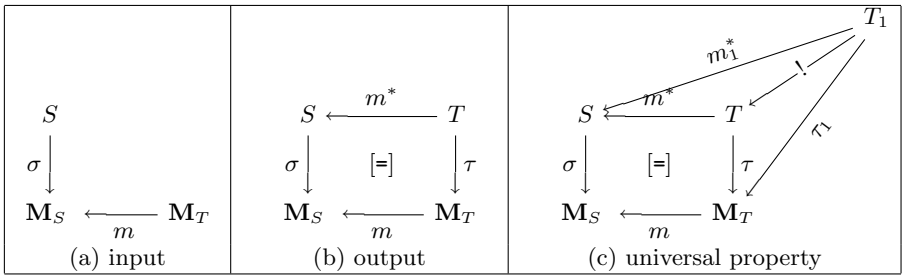


Fig. 3. Pull-back operation

2.2 Model translation generically: informal discussion

Suppose we have a model S over metamodel $\mathbf{M}_S$, which we want to translate into a model over another metamodel $\mathbf{M}_T$. Clearly, to do that in a reasonable way, we first need to specify relationships between the metamodels $\mathbf{M}_S$ and $\mathbf{M}_T$. A very simple case of such a relationship is when both metamodels are presented by similar structures (say, by graphs), and are related by a structure-preserving mapping (maybe, partially defined) $m\colon \mathbf{M}_T \to \mathbf{M}_S$. The reservation about preserving the structure is important and ensures model *translation* rather than messy mixing. Thus, the input data for the translation procedure appear as a pair of mapping with a common target as shown in diagram (a) of Fig. 3. We will call such a configuration a *sink*.

The result of translation must be a model over metamodel $\mathbf{M}_T$, hence, we should have a mapping $\tau\colon T \to \mathbf{M}_T$. In addition, each element of the new model T should appear there from some element of the original model. It is reasonable to assume that this *traceability* relationship should be structure-preserving and, hence, traceability appears as a morphism between models $m^*\colon T \to S$, see diagram (b) in Fig. 3. Thus, the result of translation appears as a pair of mappings with a common source; we will call such a configuration a *span*. In addition, if an element e in model T has a type $e.\tau$ and is traced back to element $e.m^*$, then the type of the latter should be $e.\tau.m$. That is, $e.m'.\sigma = e.\tau.m$ for all elements in model T and the diagram (b) is commutative (note the marker [=]).

The properties just listed do not necessarily characterize a unique model and we can well imagine another model T_1 together with mappings τ_1 and m'_1 making the outer "square" diagram in column (c) of Fig. 3 commutative. What should distinguish the desired translation T among other possible translations T_1, T_2, ... is that the former must not lose information and hence be maximal amongst all models T_i in some sense. In other words, model T should be considered as a union of all possible "partial" translations T_i. This suggests to specify maximality of T by the existence of a unique mapping ! to the model T from any model T_i that makes the outer diagram commutative.

These considerations motivate the following algebraic construction.

3 The pull-back operation

Let $\boldsymbol{C}$ be some universe of sets with structure (objects, nodes) and structure preserving mappings between them (morphisms, arrows).

3.1 Definition and construction. Let (σ, m) be a couple of mappings with a common target (*sink*) as shown in Fig. 3(a). A square diagram (b) is called *pull-back (PB)* if it is commutative and possesses the universal property specified by diagram (c). It can be easily proven that if (T, m^*, τ) and $(T', m^{*\prime}, \tau')$ are two arrow spans making PB-squares with the same input sink then objects T and T' are canonically isomorphic (and this isomorphism "switches" between τ and τ', and m^* and $m^{*\prime}$). Then we can consider pull-back as a *diagram operation*: given an arrow sink on its input, it produces one (up to isomorphism) arrow span on its output. We will also say that the diagram (b) is the pull-back of the diagram (a) and write $(T, \tau, m^*) = \mathsf{PB}(S, \sigma, m)$.

The pull-back operation is well-known in mathematical category theory; it works well in different contexts and appears in many applications without any relation to model translation. Thus, we can summarize our considerations in the previous section as a motivation to *define* the model translation as the PB-operation in the universe of graphs and graph morphisms. Of course, however reasonable this motivation may sound, for a more solid justification of the definition we need to consider a few examples, where we well understand what the result of the translation is, and then check whether it is indeed given by the pull-back or not.

This is the goal and contents of the next section yet before taking this endeavor, we need a constructive definition of the PB-operation. Indeed, the definition given above is entirely declarative: it explains *what* the PB is but does not say *how* to compute it. Fortunately, a well-known result of category theory says that if our universe of objects and morphisms has Cartesian products, then the pull-back object T can be computed as a specific relation over S and $\mathbf{M}_T$:

$$T = \{(a, X) \in S \times \mathbf{M}_T \mid a.\sigma = X.m\}, \tag{1}$$

and mappings m^* and τ are the projection mappings of this relation.

For example, if our universe consists of directed graphs and their mappings, the pull-back can be computed by applying the definition (1) twice: for nodes and for arrows. Example in Fig. 4 shows how it works. The lower part presents a mapping $m\colon G_T \to G_S$ between two graphs (the base mapping). The left upper quadrant presents another graph mapping $\sigma\colon S \to G_S$ specified by labeling (names of the arrows in this graph are omitted but the labels are kept; these arrows can be identified with ordered pairs of nodes they connect). The result of the PB-operation is presented by the right-upper quadrant, where we have a typed graph, and by the trace mapping between the right and left upper graphs. In specifying the right upper graph, we have used the following notation: a pair $(a, X) \in S \times G_T$ is denoted by $a \bullet X$.

3.2 Some useful mechanisms. The example demonstrates some mechanisms the PB-operation exhibits. The first one is the removal of elements: note that

elements $c1, c2$ and the two respective arrows to them have disappeared in the PB-result because their type labels C and g are out of range of the base mapping. This well fits in the model translation context: being out of range of the base mapping means that these constructs of the source metamodel are not interpretable or useless from the viewpoint of the target metamodel, and the latter does not need them.

Another important mechanism is duplication of elements. Because two nodes, Y, Z, of the graph G_T are mapped to the same node A in G_S, and correspondingly two arrows u, v are mapped to the same arrow f, the corresponding part of the source graph S is duplicated in the result. This is also a quite reasonable property in the context of model translation (see, e.g., also in [16]). Indeed, if two constructs Y, Z of the target metamodel are interpreted by the same construct A of the source metamodel, then for any source model containing elements of type A, these elements must be duplicated because they should play two roles, Y and Z, in the translated model.

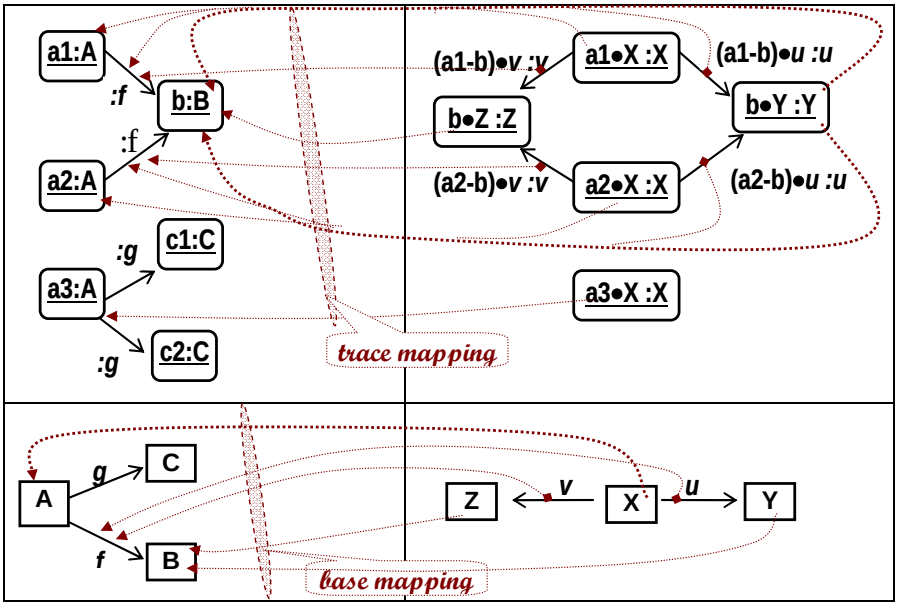


Fig. 4. Pull-back operation over graphs

3.3 The inverse pull-back problem. Sometimes, given the result of the translation, we need to find the original sink whose pull-back provides this result. In more detail, we are given the arrows τ and m as shown in Fig. 3(b), and we need to find arrows m^* and σ such that the entire square would be pull-back. We will call this task the *inverse pull-back* or *inverse translation* problem.

4 Two examples: Extracting ER-diagrams from SQL-table definitions

4.1 The first try and the lessons of the failure

It is evident that the ER-diagram in the right half of Fig. 2 is a precise counterpart of the relational schema in the left half. The question is whether it is possible to obtain this ER-diagram as a result of some algebraic operation with the relational schema. Our discussion above suggests to specify a suitable base mapping $\mathsf{er2rel} : \mathbf{M}_{\mathsf{ER}} \to \mathbf{M}_{\mathsf{Rel}}$ between metamodels and then compute the ER-model by pulling-back the corresponding arrow sink.

Indeed, if we believe that ER-diagrams can be extracted from SQL-table definitions, then elements of the ER-metamodel $\mathbf{M}_{\mathsf{ER}}$ (the target metamodel) should be found in the relational (source) metamodel, $\mathbf{M}_{\mathsf{Rel}}$. Thus, there should be a suitable mapping $\mathsf{er2rel} : \mathbf{M}_{\mathsf{ER}} \to \mathbf{M}_{\mathsf{Rel}}$ sending ER-diagram constructs to the *respective* relational constructs. The adjective "respective" is essential and means that the mapping must be semantically meaningful. This latter condition suggests to map nodes E-Node, Attribute and Type in the metamodel $\mathbf{M}_{\mathsf{ER}}$ to, respectively, nodes Table, Column and Type in the metamodel $\mathbf{M}_{\mathsf{Rel}}$, and arrows 'attr' and 'type' in $\mathbf{M}_{\mathsf{ER}}$ to arrows 'clmn' and 'type' in $\mathbf{M}_{\mathsf{Rel}}$ (see the lower part of Fig. 5 and disregard the right-most extra part of $\mathbf{M}_{\mathsf{Rel}}$ for a while; with a color display, this part is shown by blue, in the black-white printing, the extra nodes are blank and edges are thinner). Further, R-Nodes are somehow related to foreign keys and we may try to map R-Node in $\mathbf{M}_{\mathsf{ER}}$ to FKey node in $\mathbf{M}_{\mathsf{Rel}}$. The question is where to map $\mathbf{M}_{\mathsf{ER}}$'s node Role?

Roles are links that connect R-nodes (FKeys in $\mathbf{M}_{\mathsf{Rel}}$) with corresponding E-nodes (Tables in $\mathbf{M}_{\mathsf{Rel}}$). In metamodel $\mathbf{M}_{\mathsf{Rel}}$, such links are realized with arrows 'owned' and 'refs' and, thus, we would need to map the *node* Role to *arrows*. Evidently, it would violate the basic structure-preserving property of graph morphisms and make such a mapping illegal.

To manage the difficulty, we can reify the arrows in question by treating them as mappings and building their graphs. Consider the arrow 'refs'. It denotes a mapping from the set of FKey elements to the set of Table elements (find in the model graph the two arrows labeled by ':refs', one of them is not shown).

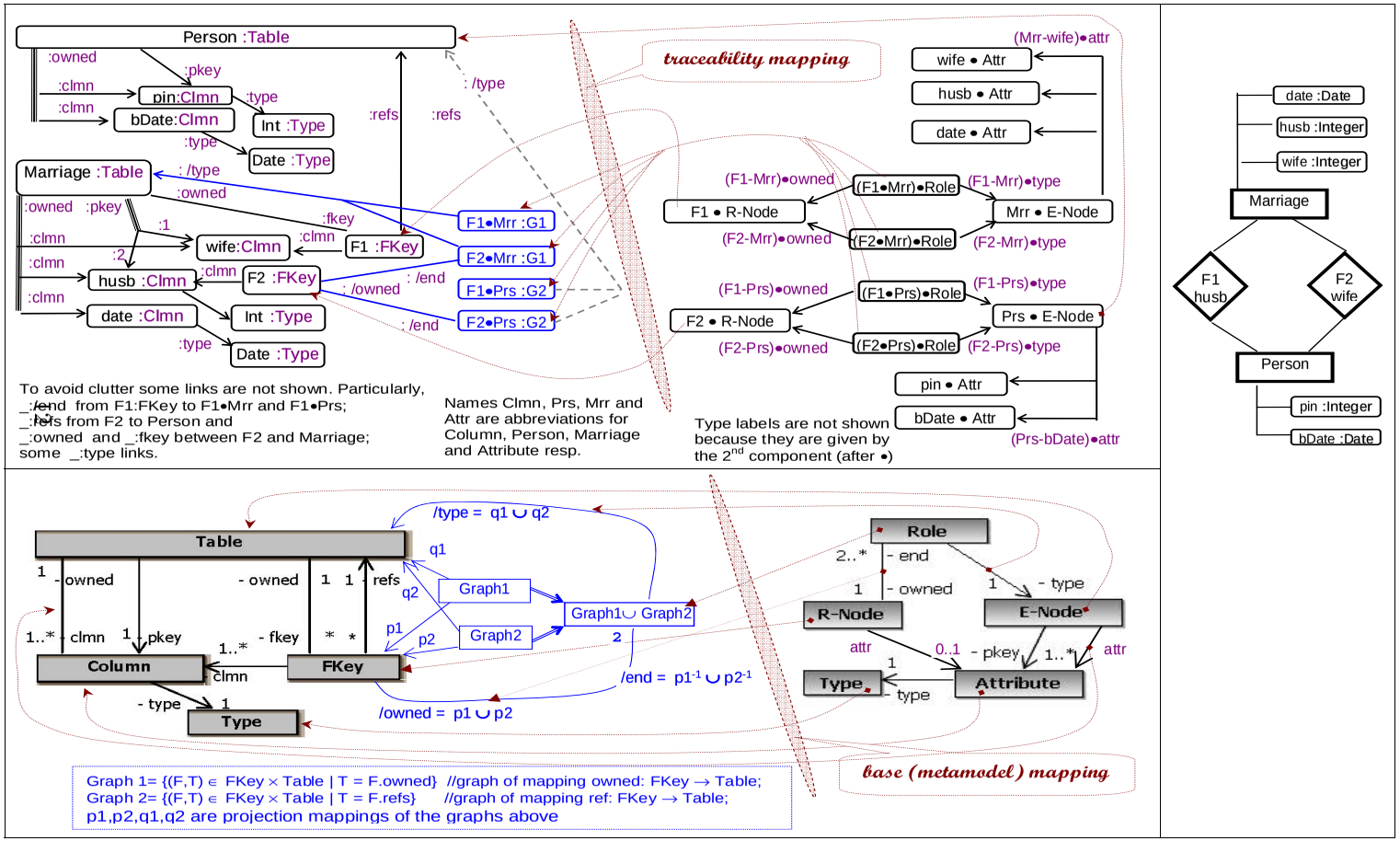


Fig. 5. Model translation via pull-back [$(X \bullet Y$ denotes the pair (X, Y)]

The graph of this mapping is a set of pairs (F:FKey, T:Table) such that T=refs(F). The metamodel of the construct is given by a node Graph2 together with two projection arrows $(p2, q2)$. Correspondingly, in the model we have two new nodes F1•*Person* and F2•*Person* typed by Graph2. Similarly, we augment the metamodel $\mathbf{M}_{\mathsf{Rel}}$ with arrow span (Graph1,p1,q1) denoting the graph of the mapping 'owned'.
Then we take the union of these two graphs – note the node Graph1∪Graph2 together with the two double arrows denoting the corresponding inclusions. Finally, we define mappings /type and /owned from the union node by taking the union of the respective projection mappings. Note also that since mappings 'refs' and 'owned' from FKey to Table have multiplicity one (are functional), the projection mappings p1 and p2 are one-one. It follows then that mapping /end $\stackrel{\text{def}}{=} p_1^{-1} \cup p_2^{-1}$ has exactly multiplicity 2. Now we can complete our metamodel mapping by sending node Role to node Graph1∪Graph2, and the arrows 'end' and 'owned' in $\mathbf{M}_{\mathsf{ER}}$ to arrows '/owned' and '/type' in $\mathbf{M}_{\mathsf{Rel}}$. Note also that our base mapping is not defined on arrow 'attr' from R-Node to Attribute.

The final step is entirely automatic: having the typed graph S and the base mapping specified in Fig. 5, we perform the PB-operation and get the typed graph shown in the upper right quadrant together with a trace mapping. Since the base mapping is injective (no two elements of $\mathbf{M}_{\mathsf{ER}}$ are mapped to the same element in $\mathbf{M}_{\mathsf{Rel}}$), computing the pull-back is fairly easy (see explanations in sect. 3 for details). It results in the typed graph presented in the right upper quadrant of Fig. 5. The ER-diagram in the right column of the figure presents this typed graph in the conventional ER-diagram syntax.

This translation is almost strait-forward but perhaps the names of the two diamonds need some explanation. For simplicity, in our ER-metamodel we assumed that names of the elements are their identifiers. In a more accurate setting, almost each of the nodes in the metamodels (besides FKey and Type) should have an attribute 'name' (presented by an arrow going out of the node to node String). Then we could define that 'name' attribute (arrow) of the node R-Node is mapped to the composition of arrows 'clmn';'name' from FKey to String (via node Column) in $\mathbf{M}_{\mathsf{Rel}}$.

Thus, the PB-operation has produced a syntactically valid ER-diagram but, semantically, the result is disappointing: compare it with a a compact and semantically transparent ER-diagram in Fig. 2. The cause of the failure is in the improper definition of the base mapping: as soon as we map $\mathbf{M}_{\mathsf{ER}}$'s nodes R-Node and Role to, respectively, nodes FKey and Graph1∪Graph2 in $\mathbf{M}_{\mathsf{Rel}}$, *each* foreign key is interpreted as a binary diamond. To obtain a proper translation, we need a deeper analysis of relational schemas, where the relation between primary and foreign keys is taken into account.

4.2 The second try: a proper metamodel mapping is a key to success

Our first attempt to translate relational schemas into ER-diagrams failed because the mapping $m\colon \mathbf{M}_{\mathsf{ER}} \to \mathbf{M}_{\mathsf{Rel}}$ we used did not make a distinction between the

two essentially different types of tables. One, we may call it E-tables, is when the primary key does not contain any foreign keys like, say, the attributes of social security or personal identification numbers. The other, let us call them R-tables, is when the primary key is composed from two or more foreign keys like in our sample relational schema in Fig. 2. For simplicity, we will exclude from consideration other cases but they can be treated as well in our framework (see below). We must also consider the partition of foreign keys into those occurring into the primary key of some table (p-foreign keys), and the others (np-foreign keys). In the former case, the table is necessarily an R-table and the pf-keys are the roles of the corresponding relationship. As for npf-keys, they are merely references to other tables and thus, in the ER-model, are themselves modeled by relationships. The roles attached to these latter relationships are just pairs of the npf-key in question and its owning and referencing table names. Thus, R-elements of the relational model are R-tables and np-foreign keys, and the roles attached to them are pf-keys and pairs mentioned above. Having defined these new E-, R- and Role-elements in the relational metamodel, we can build another mapping from $\mathbf{M}_{\mathsf{ER}}$ to $\mathbf{M}_{\mathsf{Rel}}$, which hopefully will determine a better translation algorithm.

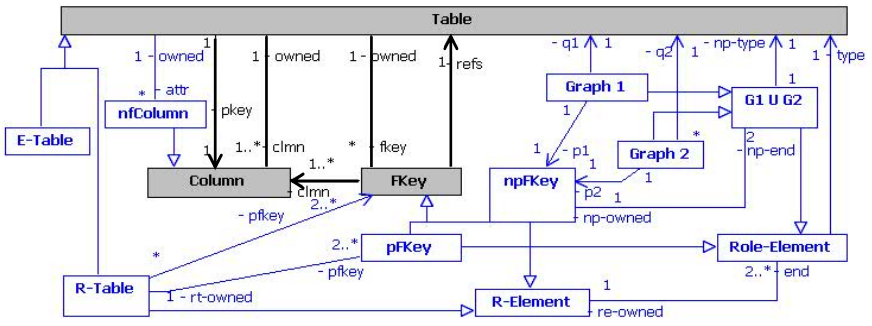
A precise description of the required manipulations with the relational metamodel is presented in Fig. 6. The upper part shows a part of the relational metamodel extended with new elements – blank nodes and thin edges (pairs of arrows), each of which is provided with a definition of its semantic meaning in the lower part of the figure.

A precise interpretation of these definitions is as follows. Any relational schema over the metamodel $\mathbf{M}_{\mathsf{Rel}}$ assigns sets $[\![\, \mathit{Tables} \,]\!]$, $[\![\, \mathit{Column} \,]\!]$ and $[\![\, \mathit{FKey} \,]\!]$ to the basic nodes and mappings $[\![\, \mathit{clmn} \,]\!]$, $[\![\, \mathit{refs} \,]\!]$ and so on to the arrows. The definition assigned to a new element E (a node or arrow) says how to compute its semantic meaning $[\![\, E \,]\!]$ (a set or mapping) from sets and mappings assigned to either basic or new elements introduced *prior* to E. (By the abuse of notation, in these definitions we write E instead of $[\![\, E \,]\!]$). In other words, the new elements are *derived* by applying the corresponding operations in contrast to the initial elements of the metamodel, which we call *basic*. Thus, Fig. 6(a) presents a derivable augmentation of (a part of) the relational metamodel $\mathsf{der}^{Q2}\mathbf{M}_{\mathsf{Rel}}$, where $Q2$ refers to the set of operations (queries) used in the derivations (in our second attempt, hence, $Q2$). Now we can define a better relational interpretation of the ER-metamodel by defining the mapping $\mathsf{er2rel}_2 \colon \mathbf{M}_{\mathsf{ER}} \to \mathsf{der}^{Q2}\mathbf{M}_{\mathsf{Rel}}$ between the metamodels as specified in the following table (only nodes are shown):

$\mathbf{M}_{\mathsf{ER}}$	E-Node	R-Node	Role	Attribute	Type
$\mathsf{der}^{Q2}\mathbf{M}_{\mathsf{Rel}}$	E-Table	R-Element	Role-Element	nfColumn	Type

Any relational schema, i.e., a model S over $\mathbf{M}_{\mathsf{Rel}}$, can be extended to a model $\mathsf{der}^{Q}S$ over the metamodel $\mathsf{der}^{Q}\mathbf{M}_{\mathsf{Rel}}$ in a unique way by actually performing operations specified in Q. That is, given a set of derivations/queries Q, any mapping $\sigma \colon S \to \mathbf{M}_{\mathsf{Rel}}$ can be uniquely extended to a mapping in $\overline{\sigma}^{Q} \colon \mathsf{der}^{Q}S \to \mathsf{der}^{Q}\mathbf{M}_{\mathsf{Rel}}$. In fact, we have already performed such a procedure in sect. 4.1, Fig. 5, where we used another set of operations $Q1 \subset Q2$. Now, having the sink of graph mor-

phisms $\overline{\sigma}^{Q2}\colon \mathsf{der}^{Q2}S \to \mathsf{der}^{Q2}\mathbf{M}_{\mathsf{Rel}}$ and $\mathsf{er2rel}_2\colon \mathbf{M}_{\mathsf{ER}} \to \mathsf{der}^{Q2}\mathbf{M}_{\mathsf{Rel}}$, it is just an exercise in computing the pull-back to show that the PB-image of the relational model in Fig. 2 on the right is exactly the ER-diagram in that Figure on the left. Thus, given a proper ER-to-Rel metamodel interpretation, the pull-back operation computes the desired result.



(a) The extended metamodel (derived elements are blue, nodes are blank and edges are thin)

E-Table = { T∈Table | T.pkey ∩ (∪ T.fkey) = ∅

R-Table = {T ∈Table | T.pkey = ∪ $\mathcal{F}$ for some $\mathcal{F} \subset$ T.fkey with card($\mathcal{F}$) > 1 }

for RT∈R-Table, RT.pfkey = $\mathcal{F}$ used in the definition above

pFkey = ∪{ RT.pfkey | RT∈ R-Table} with evident rt-owned and pfkey mappings

npFKey = FKey – pFKey

nfColumn = {C∈ Column | C∉ F.clmn for all F∈ FKey }

Graph 1={(F,T) ∈ FKey × Table | T = F.owned} (graph of mapping owned: FKey → Table)

Graph 2={(F,T) ∈ FKey × Table | T = F.refs} (graph of mapping ref: FKey → Table)

p1,p2,q1,q2 are projection mappings of the graphs above

G1∪G2 = Graph1 ∪ Graph2

for G∈G1∪G2, G.np-type=G.q1 if G∈Graph1 and G.q2 otherwise

for G∈G1∪G2, G.np-owned = G.p1 if G∈Graph1 and G.p2 otherwise

for F∈npFKey, F.np-end = {F.p1^{-1}, F.p2^{-1}}

Role-Element = pFKey ∪ (G1∪G2)

for Ro∈Role-Element, Ro.type = Ro.refs if Ro∈pFKey and Ro.np-type otherwise

R-Element = R-Table ∪ npFKey

for Re∈R-Element, Re.end = Re.pfkey if Re∈R-Table and Re.end=np-end otherwise

(b) Definitions of derived elements

Fig. 6. Metamodel of relational schemas extended with derived elements to map to it the ER-metamodel

5 Algebra of reverse engineering

5.1 Divide and conquer: algebra vs. heuristics.

The examples we considered suggest the following general pattern for model translation and reverse engineering. We need to translate models over some (*source*) metamodel $\mathbf{M}_S$ to models over another (*target*) metamodel $\mathbf{M}_T$. The key to the entire process is in a suitable interpretation of $\mathbf{M}_T$-elements by $\mathbf{M}_S$-elements. However, setting a proper interpretation may need augmenting the source metamodel with new derived elements corresponding to those elements in $\mathbf{M}_T$, whose $\mathbf{M}_S$-counterparts are not immediately recorded in $\mathbf{M}_S$ but can be derived from them by applying suitable algebraic operations. The latter are nothing but queries to the metamodel $\mathbf{M}_S$, if we understand models as data and metamodel as their schema. Thus, we need to find a set Q of queries against metamodel $\mathbf{M}_S$ such that the corresponding extension $\mathsf{der}^Q\mathbf{M}_S$ allows a semantically justified mapping $m\colon \mathbf{M}_T \to \mathsf{der}^Q\mathbf{M}_S$.

This is the most non-trivial part of the problem: it needs a solid understanding of the semantics of both metamodels and their relationships, of the goals of the translation and of the relevant pragmatic aspects. Algebraic (like any other formal) procedures provide a proper output only when they are supplied with a proper input (for example, an adequate and semantically meaningful metamodel mapping mentioned above). Specifying such an input could be non-trivial and require heuristic efforts; often, this *is* the main issue in the problem in question. Yet having an algebraic formal model helps here too in that it provides a clear specification of what should be the output of the heuristic procedures.

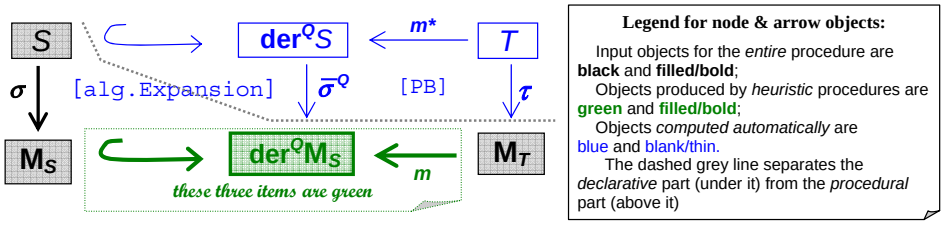


Fig. 7. Algebra and heuristics in model translation

However complex the heuristic initial phase of the process could be, we assume that it is accomplished and its results are presented by a pair (Q, m) with Q a set of queries to $\mathbf{M}_S$ and $m\colon \mathbf{M}_T \to \mathsf{der}^Q\mathbf{M}_S$ a metamodel mapping. After this pair is set, everything else in translating any $\mathbf{M}_S$-model to an $\mathbf{M}_T$-model is automatic. Given an arbitrary model $\sigma\colon S \to \mathbf{M}_S$ over the source metamodel, we extend it with derived elements by executing queries specified by $\mathsf{der}^Q\mathbf{M}_S$. The result is an extended model $\overline{\sigma}^Q\colon \mathsf{der}^Q S \to \mathsf{der}^Q\mathbf{M}_S$. The next step is to apply the PB-algorithm to the arrow sink $(\mathsf{der}^Q\mathbf{M}_S, \overline{\sigma}^Q, m)$. The algorithm returns an arrow span (T, τ, m^*), which we interpret as the translated model $\tau\colon T \to \mathbf{M}_T$

together with the traceability mapping $m^*: T \to \mathsf{der}^Q S$. Our discussion is summarized in Fig. 7.

5.2 Model translation as view computation.

Figure 7 shows that there are two algebraic procedures embodied into model translation: algebraic augmentation/expansion of models (querying) and pull-back (retyping), and they both need convenient and effective mechanisms to be implemented in RE-tools. An important observation in this respect is that together the two steps amount to a quite ordinary database procedure called view computation. The model S is data over the schema $\mathbf{M}_S$, metamodel $\mathbf{M}_T$ is a view schema and the model T is the (materialized) view. Hence, the entire procedure can be well implemented with a DBMS having an effective engine of complex query evaluation. It seems that this possibility of employing the database theory and tools for RE is not well explored. Of course, an important issue is how expressive the query language should be in order to provide proper interpretations/mappings between metamodels in practically interesting cases.[1]

5.3 Scope of applicability.

How universal is model representation by typed directed graphs? Though applicability of this pattern is surprisingly broad, there are two important limitations. The first is structural: we can imagine reasonable cases of graph-based metamodels, whose graphical structure is richer than simple graphs. Typical examples are 2- and n-graphs (where in addition to arrows between nodes there are arrows between arrows, *2-arrows*, and so on); *reflexive* graphs (where each node in the metamodel is supplied with one or more special arrows with a fixed meaning, e.g., *identity* arrow or *idle* transitions and the like); *hypergraphs* or, say, *attributed graphs*. The notion of pull-back and the corresponding machinery can be readily expanded for these and similar graph-based structures via the notion of presheaf topos, see [8] for some details.

The second "beyond-graphs" case is when not all morphisms $\sigma: S \to \mathbf{M}_S$ represent models (though each legal model is still a morphism). To exclude unwanted morphisms, we need to add constraints to the graph-based structure $\mathbf{M}_S$. Such constraints can be also treated diagrammatically in a special language of *diagram predicates*, and in this way we come to a structure called *generalized sketch* [9]. An important (and easy) result is that if a morphism $\sigma: S \to \mathbf{M}_S$ is a legal model, the base mapping $m: \mathbf{M}_T \to \mathbf{M}_S$ is a sketch morphism (i.e., is compatible with the constraints embodies into $\mathbf{M}_T$ and $\mathbf{M}_S$), and $(T, \tau, m^*) = \mathsf{PB}(S, \sigma, m)$, then morphism τ is also a legal model. Thus, if the base mapping is compatible with constraints, the PB-procedure transforms legal models into legal models. To summarize, the PB-pattern works well far beyond the modeling framework of simple typed graphs.

[1] It is closely related to the data exchange problem, which lately has been actively studied by the database community [12].

References

[1] Paolo Atzeni, Paolo Cappellari, and Philip A. Bernstein. Model-independent schema and data translation. In *EDBT*, pages 368–385, 2006.

[2] P. Bernstein. Applying model management to classical metadata problems. In *Proc. CIDR'2003*, pages 209–220, 2003.

[3] P. Bernstein, A. Halevy, and R. Pottinger. A vision for management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[4] Jean Bézivin, Salim Bouzitouna, Marcos Didonet Del Fabro, Marie-Pierre Gervais, Frédéric Jouault, Dimitrios S. Kolovos, Ivan Kurtev, and Richard F. Paige. A canonical scheme for model composition. In Arend Rensink and Jos Warmer, editors, *ECMDA-FA*, Lecture Notes in Computer Science, pages 346–360. Springer, 2006.

[5] K. Czarnecki and S. Helsen. Classification of model transformation approaches. In K. Czarnecki, editor, *2nd OOPSLA03 Workshop on Generative Techniques in the Context of MDA*, 2003.

[6] Z. Diskin. Mathematics of generic specifications for model management. In Rivero, Doorn, and Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 351–366. Idea Group, 2005.

[7] Z. Diskin. Metamodel-independent schema and data merge: Towards syntax-semantics integration in generic model management. Technical Report 2006-522, School of Computing, Queen's University, Kingston, ON, Canada, 2006. http://www.cs.queensu.ca/TechReports/reports2006.html.

[8] Z. Diskin, J. Dingel, and H. Liang. Scenario integration via higher-order graphs. Technical Report 2006-517, School of Computing, Queen's University, Kingston, ON, Canada, 2006. http://www.cs.queensu.ca/TechReports/reports2006.html.

[9] Z. Diskin and B. Kadish. Variable set semantics for keyed generalized sketches: Formal semantics for object identity and abstract syntax for conceptual modeling. *Data & Knowledge Engineering*, 47:1–59, 2003.

[10] Z. Diskin and B. Kadish. Generic model management. In Rivero, Doorn, and Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 258–265. Idea Group, 2005.

[11] K. Ehrig, E.Guerra, J. de Lara, L. Lengyel, T.Levendovszky, U.Prange, G.Taentzer, D.Varró, and S.Varró-Gyapay. Model transformation by graph transformation: A comparative study. In *MTiP 2005, Int.Workshop on Model Transformations in Practice (Satellite Event of MoDELS 2005)*, 2005.

[12] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1), 2005.

[13] Martin Gogolla. Exploring ER and RE Syntax and Semantics with Metamodel Object Diagrams. In Uffe K. Wiil, Peter J. Nürnberg, and David L. Hicks, editors, *Metainformatics Symposium (MIS'2005)*. Springer, Berlin, LNCS, 2006.

[14] S. Melnik, P. Bernstein, A. Halevy, and E. Rahm. Supporting executable mappings in model management. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 167–178, New York, NY, USA, 2005. ACM Press.

[15] M.Grosse-Rhode, F.Presicce, and M. Simeoni. Formal software specification with refinements and modules of typed graph transformation systems. *J. Comput. Syst. Sci.*, 64(2):171–218, 2002.

[16] Laurence Tratt. Model transformations and tool integration. *Software and System Modeling*, 4(2):112–122, 2005.