

# A metamodel independent framework for model transformation: Towards generic model management patterns in reverse engineering <sup>\*</sup>

Zinovy Diskin and Juergen Dingel

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

**Abstract.** The paper presents a formal algebraic framework, where model translations can be specified in a truly generic way: the source and the target metamodels are parameters of the entire procedure. In more detail, we specify the operation of model translation 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.

## 1 Introduction and motivating discussion

**1.1 Reverse engineering (RE) as a stage for the generic model management (MMt).** In the narrow technical sense, this paper is about a generic pattern for model translation: a (source) model  $S$  in some metamodel  $\mathbf{M}_S$  is to be transformed into a (target) model  $T$  in another metamodel  $\mathbf{M}_T$  (we use the terms *translation* and *transformation* interchangeably). The term “generic” means that we are looking for a translation procedure in which the source and the target metamodels would be parameters. Then, for example, translations of SQL-tables into ER-diagrams or Java code into UML class diagrams or into UML-sequence diagrams all will be particular instances of the same algorithmic pattern. We believe that such a pattern would be of real value for reverse engineering (RE): for its theoretical foundations and for tool design as well.

With respect to the latter, and here we come to a broader perspective of this paper goals, what we have in mind is some integrated environment, where a RE-engineer can manipulate models of software artifacts in different ways: translate them from one presentation to another, refine and abstract them, extract different views of the same model, compare and relate them between themselves, and integrate (merge) them.<sup>1</sup> We believe that the task of implementing such an environment for RE requires the full power of the *generic model management* framework as it was conceived in the database community [2]. Indeed, RE-tools

---

<sup>\*</sup> Research supported by OCE Centre for Communications and Information Technology and IBM CAS Ottawa.

<sup>1</sup> See, e.g., [15] for the role of model merge in RE.

operate software artifacts (programs) as data, and the size and complexity of the modern software make analysis and manipulation of these data close to a fully-fledged DBMS task. In the database view, repositories in which RE-tools store the source code are databases: they are populated by fact extractors at the front-ends of RE-tools, and are queried and analyzed at the back ends. In this sense, RE-scenarios of model operation present a special (and really complex yet just a) particular case of the general "MMt-play". The latter is fashionable in different research communities (but in different incarnations, cf.[11]) and is in active development. Although the field is still in its infancy, availability of a working prototype of MMt environments [13], and of a mathematical framework providing basic generic algorithms for MMt [4], demonstrate feasibility of the approach. A conjecture underlying the present paper is that the generic MMt view on RE can be fruitful theoretically and useful practically.

**1.2 MMt- vs. MT-programming.** In comparison to the database community, software engineering has undertaken a different approach to model operation. It mainly concentrates on model translation (here usually called transformation), MT, which is the key components of MDE/MDA and other recent model-centric endeavors. The idea is to design a specific programming language, where particular MT-tasks can be specified and then executed (we will call it *MT-programming*). The field is flourishing and an enormous amount of MT-languages were designed (including an OMG standard QVT [16]) and tried in practice: see [3] for a survey and [17, 12] for recent updates.

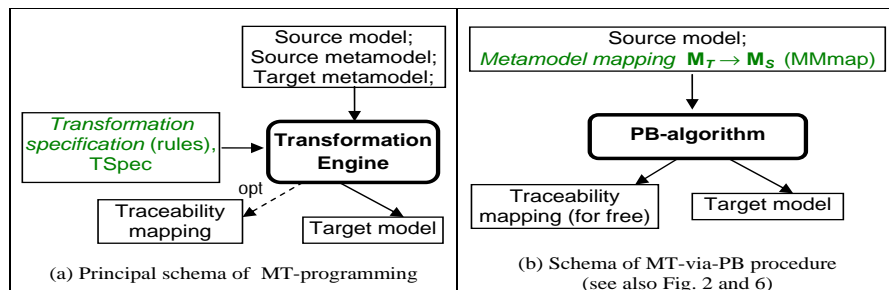


Fig. 1. Comparison of PB- and MT- (including GT-)programming

A principle schema of MT-programming (adapted from [9]) is presented in Fig. 1(a). 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. As a model usually comprises a big set

of different elements and their structural links, the elementwise nature of TSpec makes MT-programming notoriously laborious and error-prone. It is these problems in metadata management that have driven the database community to the idea of *generic model management*. The essence of the latter is to offer the programmer languages and tools for manipulating models as holistic entities without zooming into their elementwise internal structure (see [8] for a survey and discussion).

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 [1], Bernstein even questioned the very possibility of specifying this operation in a generic way. Fortunately, mathematical category theory offers a suitable apparatus for designing generic specifications, see [4] for an outline of its MMT-applications. In the present paper we apply the machinery to the model translation operator in the RE-context.

**1.3 Model transformation via pull-backs (queries vs. updates).** 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  $m: \mathbf{M}_T \rightarrow \mathbf{M}_S$  interpreting  $\mathbf{M}_T$ -constructs by suitable  $\mathbf{M}_S$ -constructs. However, in practice (particularly, in RE), 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  $m: \mathbf{M}_T \rightarrow \text{der}^Q \mathbf{M}_S$  into some augmentation  $\text{der}^Q \mathbf{M}_S$  of the source metamodel with derived elements (where  $Q$  refers to the set of operations/queries used).

Given such a mapping (MMmap), we then argue 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., [14]).

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  $MT\text{-retyping} = PB\text{-retyping}$  is just a definition, which we can accept (if we believe it is useful) or reject otherwise. What we will try to do in the paper is to carefully *motivate* this definition, i.e., to motivate that defining model trans-

formation to be the result of the corresponding PB-operation is an adequate algebraic model of MT.

The principle schema of the PB-approach is shown in Fig. 1(b). 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 (mappings) between metamodels 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 obtain the target model by changing (updating, in the database jargon) the source model. 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 language, MMmap is nothing but a view definition and the target model is the corresponding materialized view. We will return to comparison of the two approaches, particularly, in the context of RE, 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.

**Acknowledgement.** We appreciate critical yet stimulating remarks of the anonymous reviewers. Special thanks go to Jean-Marie Favre for the encouragement to put these ideas on paper.

## 2 Model translation as an arrow diagram operation

**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) is that typing can be considered as a structure-preserving mapping (morphism)  $\sigma: S \rightarrow \mathbf{M}_S$  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*  $\mathbf{M}_S$ .

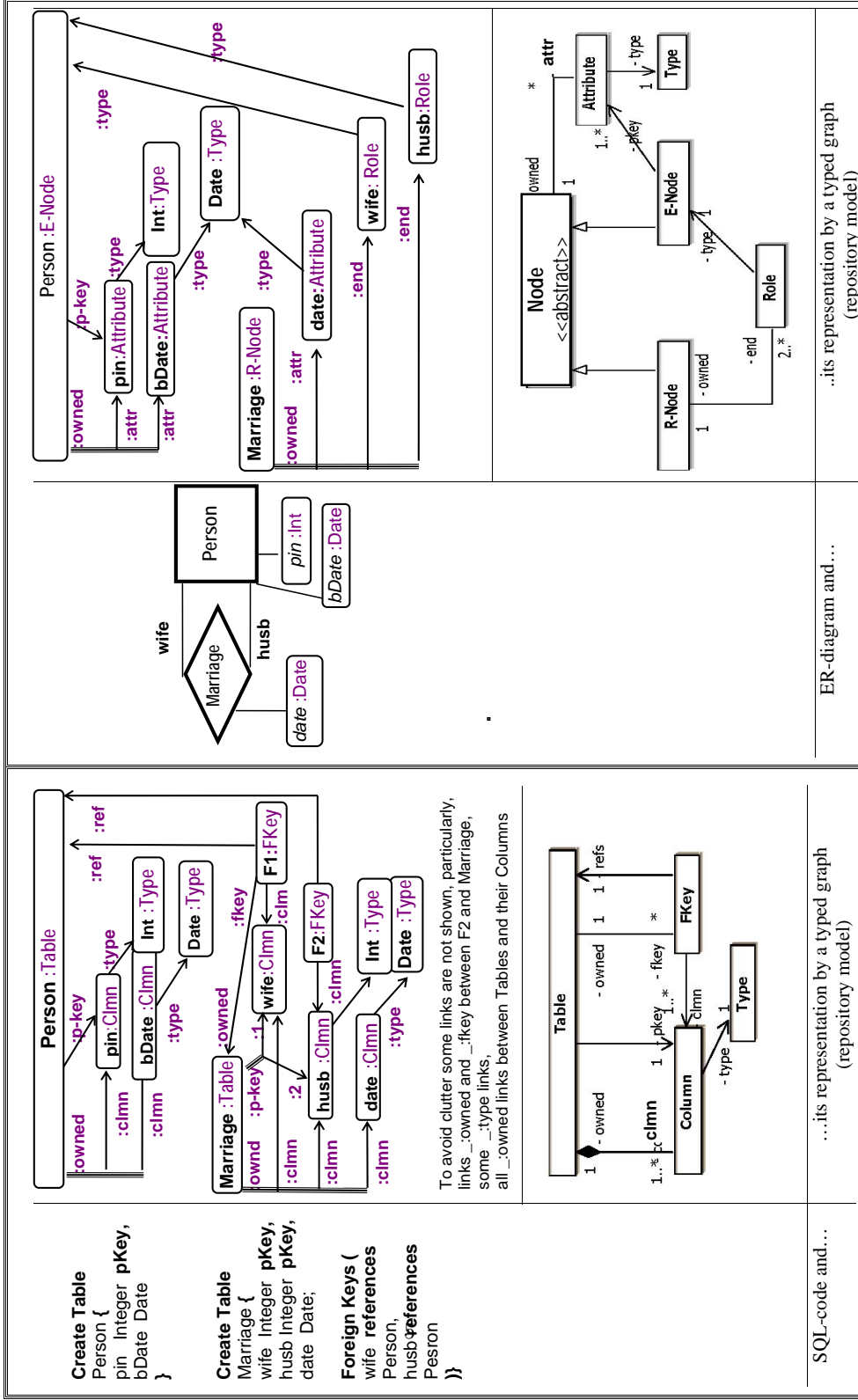


Fig. 2. Models as typed graphs

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.

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. 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  $\sigma: S \rightarrow \mathbf{M}_S$  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. Again, it is easy to see that the typed graph on the right specified a graph morphism  $\tau: T \rightarrow \mathbf{M}_T$  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.

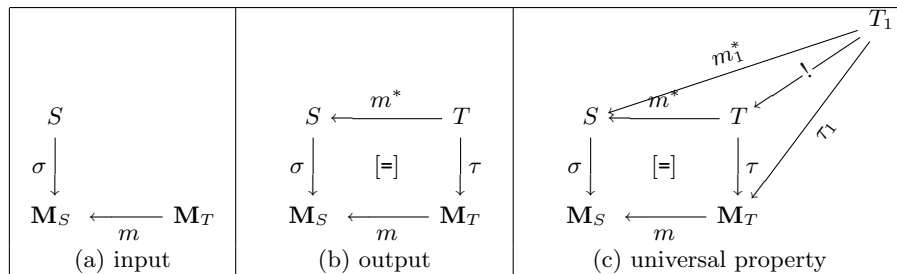


Fig. 3. Pull-back operation

**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)  $m: \mathbf{M}_T \rightarrow \mathbf{M}_S$ . 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  $\tau: T \rightarrow \mathbf{M}_T$ . 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  $m^*: T \rightarrow S$ , 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  $\tau_1$  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  $T_1, T_2, \dots$  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  $\mathcal{C}$  be some universe of sets with structure (objects, nodes) and structure preserving mappings between them (morphisms, arrows).

**3.1 Definition and construction.** Let  $(\sigma, m)$  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  $(T, m^*, \tau)$  and  $(T', m'^*, \tau')$  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  $\tau$  and  $\tau'$ , and  $m^*$  and  $m'^*$ ). 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  $(T, \tau, m^*) = \text{PB}(S, \sigma, m)$ .

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  $\mathbf{M}_T$ :

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

and mappings  $m^*$  and  $\tau$  are the projection mappings of this relation. the details how PB-works for graph objects, and why it is relevant for RE can be found in [5]. We are forced to skip them here due to space limitations

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  $m: G_T \rightarrow G_S$  between two graphs (the base mapping). The left upper quadrant presents another graph mapping  $\sigma: S \rightarrow G_S$  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, hence, model elements over them are not translated.

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 (cf. [17]). Indeed, if two constructs  $Y, Z$  of the target metamodel are interpreted by the same construct  $A$  of the source metamodel, then any source model elements of type  $A$  play two roles,  $Y$  and  $Z$ , in the translated model and hence must be duplicated.

## 4 Extracting ER-diagrams from SQL-table definitions: a proper metamodel mapping is a key to success

**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

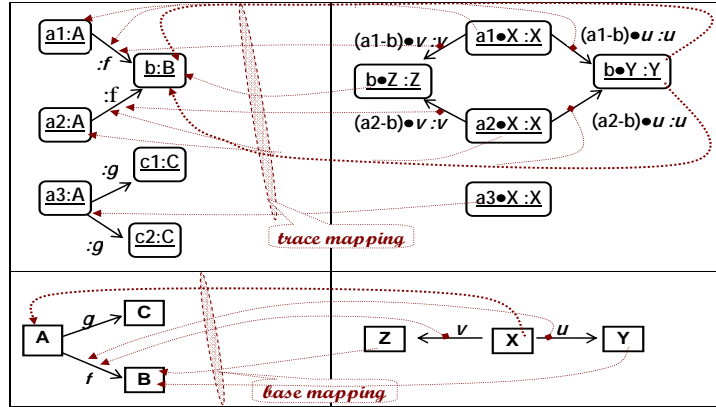


Fig. 4. Pull-back operation over graphs

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  $\text{er2rel} : \mathbf{M}_{\text{ER}} \rightarrow \mathbf{M}_{\text{Rel}}$  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  $\mathbf{M}_{\text{ER}}$  (the target metamodel) should be found in the relational (source) metamodel,  $\mathbf{M}_{\text{Rel}}$ . Thus, there should be a suitable mapping  $\text{er2rel} : \mathbf{M}_{\text{ER}} \rightarrow \mathbf{M}_{\text{Rel}}$  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  $\mathbf{M}_{\text{ER}}$  to, respectively, nodes Table, Column and Type in the metamodel  $\mathbf{M}_{\text{Rel}}$ , and arrows ‘attr’ and ‘type’ in  $\mathbf{M}_{\text{ER}}$  to arrows ‘clmn’ and ‘type’ in  $\mathbf{M}_{\text{Rel}}$  (see the lower part of Fig. 5 and disregard the right-most “blank” extra part of  $\mathbf{M}_{\text{Rel}}$  for a while<sup>2</sup>). Further, R-Nodes are somehow related to foreign keys and we may try to map R-Node in  $\mathbf{M}_{\text{ER}}$  to FKKey node in  $\mathbf{M}_{\text{Rel}}$ . The question is where to map  $\mathbf{M}_{\text{ER}}$ ’s node Role?

Roles are links that connect R-nodes (FKKeys in  $\mathbf{M}_{\text{Rel}}$ ) with corresponding E-nodes (Tables in  $\mathbf{M}_{\text{Rel}}$ ). In metamodel  $\mathbf{M}_{\text{Rel}}$ , 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. The box at the very bottom of the figure describes this procedure (we refer to [5] for details). Now we can complete our metamodel mapping by sending node Role to node  $\text{Graph1} \cup \text{Graph2}$ , and the arrows ‘end’ and ‘owned’ in  $\mathbf{M}_{\text{ER}}$  to arrows

<sup>2</sup> with a color display, this part is blue

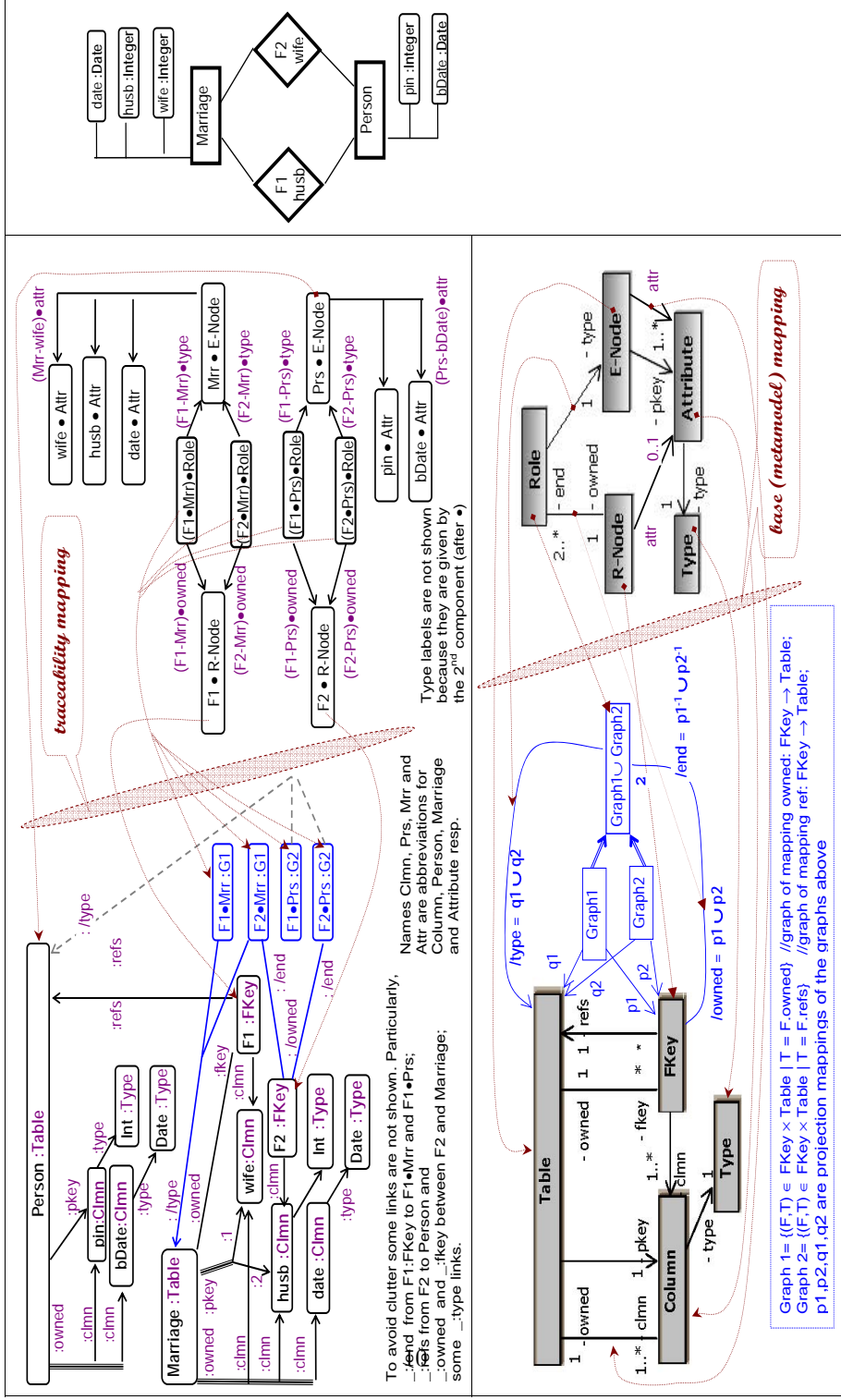


Fig. 5. Model translation via pull-back [(X • Y denotes the pair (X, Y)]

'/owned' and '/type' in  $\mathbf{M}_{\text{Rel}}$ . 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  $\mathbf{M}_{\text{ER}}$  are mapped to the same element in  $\mathbf{M}_{\text{Rel}}$ ), 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 (see [5] for details).

Thus, the PB-operation has produced a syntactically valid ER-diagram but, semantically, the result is disappointing: compare it with 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  $\mathbf{M}_{\text{ER}}$ 's nodes R-Node and Role to, respectively, nodes FKKey and Graph1UGraph2 in  $\mathbf{M}_{\text{Rel}}$ , *each* foreign key is interpreted as a binary diamond. To obtain a proper translation, we need a deeper analysis of relational schemas.

**4.2 The second try.** There are two types of relational 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 in our framework as well. 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 this way we can define new E-, R- and Role-elements in the relational metamodel and build another mapping from  $\mathbf{M}_{\text{ER}}$  to  $\mathbf{M}_{\text{Rel}}$ , which hopefully will determine a better translation algorithm. Details of the required manipulations with the relational metamodel can be found in [5], and the results are 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. In fact, Fig. 6(a) presents a derivable augmentation of the relational metamodel  $\text{der}^{Q2}\mathbf{M}_{\text{Rel}}$ , 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  $\text{er2rel}_2: \mathbf{M}_{\text{ER}} \rightarrow \text{der}^{Q2}\mathbf{M}_{\text{Rel}}$  between the metamodels as specified in the following table (only nodes are shown):

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

Any relational schema, i.e., a model  $S$  over  $\mathbf{M}_{\text{Rel}}$ , can be extended to a model  $\text{der}^Q S$  over the metamodel  $\text{der}^Q\mathbf{M}_{\text{Rel}}$  in a unique way by actually performing operations specified in  $Q$ . That is, given a set of derivations/queries  $Q$ , any mapping  $\sigma: S \rightarrow \mathbf{M}_{\text{Rel}}$  can be uniquely extended to a mapping in  $\bar{\sigma}^Q: \text{der}^Q S \rightarrow \text{der}^Q\mathbf{M}_{\text{Rel}}$ .

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 morphisms  $\bar{\sigma}^{Q2}: \text{der}^{Q2}S \rightarrow \text{der}^{Q2}\mathbf{M}_{\text{Rel}}$  and  $\text{er2rel}_2: \mathbf{M}_{\text{ER}} \rightarrow \text{der}^{Q2}\mathbf{M}_{\text{Rel}}$ , it is just an exercise in computing pull-backs 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.

## 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  $\text{der}^Q\mathbf{M}_S$  allows a semantically justified mapping  $m: \mathbf{M}_T \rightarrow \text{der}^Q\mathbf{M}_S$ .

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. This phase can be especially complicated in the context of RE because of the semantic gap between the models. Indeed, in RE, the intentions and concepts behind the two metamodels can be dramatically different: implementation concerns for  $\mathbf{M}_S$  vs. high-level abstract modeling for  $\mathbf{M}_T$ . Semantic concepts important for the latter can be deeply hidden in the former and their extraction could require complex and bulky query specifications. Note, for example, that in Fig. 6(a) a major part of the model consists of derived elements (is blue with a color display).

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  $m: \mathbf{M}_T \rightarrow \text{der}^Q\mathbf{M}_S$  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  $\sigma: S \rightarrow \mathbf{M}_S$  over the source metamodel, we extend it with derived elements by executing queries specified by  $\text{der}^Q\mathbf{M}_S$ . The result is an extended model  $\bar{\sigma}^Q: \text{der}^Q S \rightarrow \text{der}^Q\mathbf{M}_S$ . The next step is to apply the PB-algorithm to the arrow sink  $(\text{der}^Q\mathbf{M}_S, \bar{\sigma}^Q, m)$ . The algorithm returns an arrow span  $(T, \tau, m^*)$ , which we interpret as the translated model  $\tau: T \rightarrow \mathbf{M}_T$  together with the traceability mapping  $m^*: T \rightarrow \text{der}^Q S$ . 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 of 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.<sup>3</sup>

**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 [6] for some details.

The second “beyond-graphs” case is when not all morphisms  $\sigma: S \rightarrow \mathbf{M}_S$  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* [7]. An important (and easy) result is that if a morphism  $\sigma: S \rightarrow \mathbf{M}_S$  is a legal model, the base mapping  $m: \mathbf{M}_T \rightarrow \mathbf{M}_S$  is a sketch morphism (i.e., is compatible with the constraints embodied into  $\mathbf{M}_T$  and  $\mathbf{M}_S$ ), and  $(T, \tau, m^*) = \text{PB}(S, \sigma, m)$ , then morphism  $\tau$  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.

## References

- [1] P. Bernstein. Applying model management to classical metadata problems. In *Proc. CIDR'2003*, pages 209–220, 2003.
- [2] P. Bernstein, A. Halevy, and R. Pottinger. A vision for management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

<sup>3</sup> It is closely related to the data exchange problem, which lately has been actively studied by the database community [10].

- [3] 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.
- [4] Z. Diskin. Mathematics of generic specifications for model management. In Rivero, Doorn, and Ferragine, editors, *Encyclopedia of Database Technologies and Applications*, pages 351–366. Idea Group, 2005.
- [5] Z. Diskin. Model transformation via pull-backs: algebra vs. heuristics. Technical Report 2006-521, School of Computing, Queen’s University, Kingston, ON, Canada, 2006. <http://www.cs.queensu.ca/TechReports/reports2006.html>.
- [6] 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>.
- [7] 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.
- [8] Z. Diskin and B. Kadish. Generic model management. In Rivero, Doorn, and Ferragine, editors, *Encyclopedia of Database Technologies and Applications*, pages 258–265. Idea Group, 2005.
- [9] 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.
- [10] R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1), 2005.
- [11] J-M. Favre. Towards a basic theory to model model driven engineering. In M. Gogolla, P. Sammut, and J. Whittle, editors, *Workshop on Software Model Engineering, WISME 2004, joint event with UML2004*, 2004.
- [12] R. Holt, A. Schürr, S. Elliott Sim, and A. Winter. GXL: A graph-based standard exchange format for reengineering. *Science of Computer Programming*, 60(2):149–170, 4 2006.
- [13] S. Melnik, E. Rahm, and P. Bernstein. Developing metadata-intensive applications with Rondo. *J. Web Semantics*, 1:47–74, 2003.
- [14] 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.
- [15] Daniel L. Moise and Kenny Wong. Issues in integrating schemas for reverse engineering. *Electr. Notes Theor. Comput. Sci.*, 94:81–91, 2004.
- [16] OMG, Object Management Group, <http://www.omg.org/docs/ptc/05-11-01>. MOF QVT Final Adopted Specification. Formal/05-11-01, 2005.
- [17] Laurence Tratt. Model transformations and tool integration. *Software and System Modeling*, 4(2):112–122, 2005.

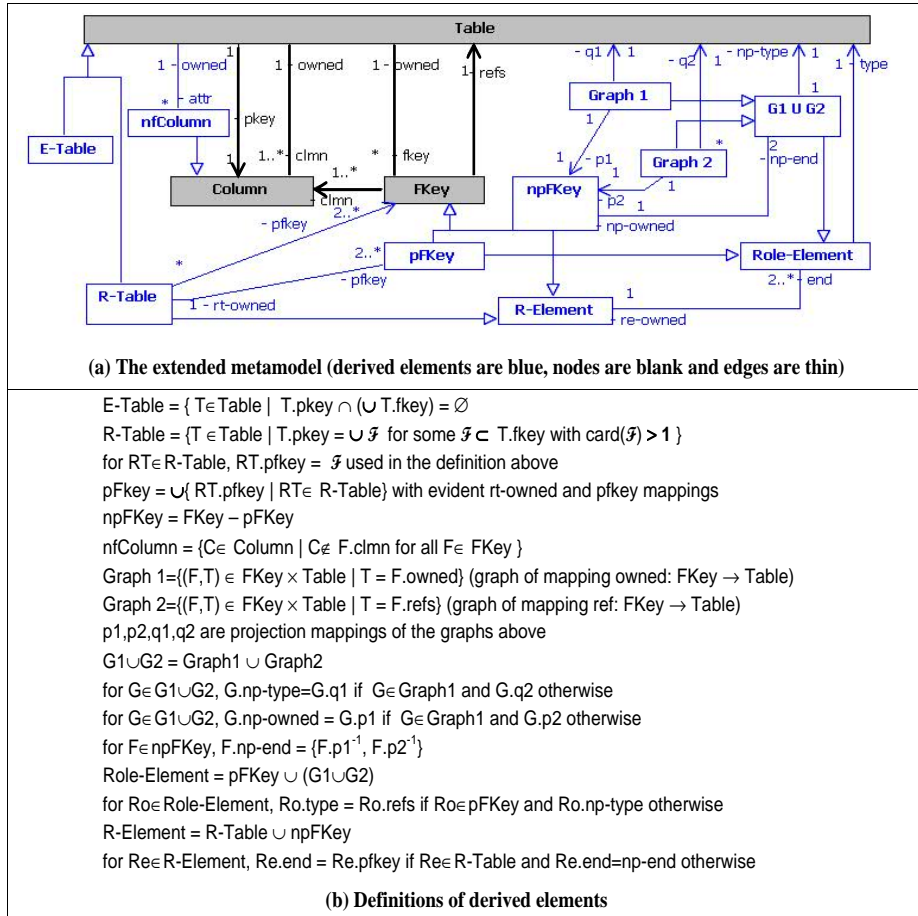


Fig. 6. Metamodel of relational schemas extended with derived elements to map to it the ER-metamodel

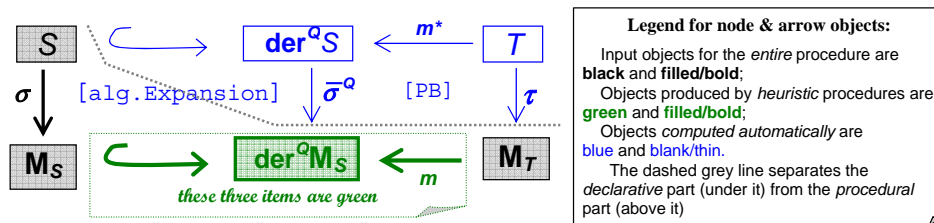


Fig. 7. Algebra and heuristics in model translation