GSDLAB TECHNICAL REPORT

Traceability Mappings as a Fundamental Aspect of Model Transformations

Zinovy Diskin

GSDLAB–TR 2016-05-01 May 2016

Generative Software
Development Lab

Generative Software Development Laboratory
University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1

WWW page: http://gsd.uwaterloo.ca/

Traceability Mappings as a Fundamental Aspect of Model-to-Model Transformations

Zinovy Diskin[1,2]

[1] McMaster University, Canada
[2] University of Waterloo, Canada
zdiskin@gsd.uwaterloo.ca

Abstract. Technological importance of traceability mappings is well-known. The paper argues that traceability mappings are also fundamental for semantics: we present a simple example showing that specifying (model-to-model) model transformations without traceability makes their semantics essentially incomplete. We also show that the traceability mapping from the transformed to the original model should correlate with a corresponding mapping from the target to the source metamodel. Moreover, the transformation can be seen as the result of execution of the metamodel mapping, and the latter thus appears as (a special encoding of) the transformation definition.

1 Introduction

Translating models from one to another metamodel (also known as model-to-model transformation, MMT) is ubiquitous in software engineering. The technological importance of traceability for MMT is well recognized in the MMT community. Such widely used transformation languages as ATL [10] and ETL [11] automatically create traceability records/links during the transformation execution in order to resolve dependencies between the rules. After the execution is finished, traceability data still have multiple important applications such as debugging, change management and maintenance, and back-annotations [2,12,9].

In all applications mentioned above, traceability appears as an important but auxiliary component of MMT, intended to facilitate the design and maintenance, and correct execution, of the transformation rather than to define it. The goal of the present paper is to argue, step-by-step, that traceability mappings (= sets of traceability links) play a more fundamental—*semantic*—role for MMTs so that the very definition of an MMT can be seen as a (meta-)traceability mapping from the target metamodel to the source one. We will first show that (traceability) *mapping-free* semantics of MMTs is essentially incomplete: we present two simple *different* transformations that nevertheless produce the same target model for any source model, but taking traceability into account does differentiate the transformations. Then we argue that traceability links between models' elements are to be complemented with traceability links between metamodels' elements (extracted from transformation rules) so that the traceability and typing mappings form a commutative diagram. Commutativity ensures that model

traceability links are properly typed by meta-traceability links so that meta-traceability mappings play the role of metamodels for traceability links. Finally, we show that meta-traceability mappings can be executed, and moreover, transformations can be defined by defining respective traceability mappings. In this sense, the approach seems close to QVT [**?**] and TGG [8] MMTs, but there is an important distinction. QVT and TGG transformations are defined by *sets* of transformation rules operating over *individual* model elements, while in the view presented in the paper, an entire transformation is defined by a *single* metatraceability mapping, involving operations (queries) over *sets* of model elements.

Including traceability mappings into semantics essentially changes the specification framework underlying MMTs. It requires a convenient notation and terminology, and a mapping-centric mathematical support with mappings properly directed and operated. The paper proposes several steps in this direction borrowed from category theory. On the other hand, defining an MMT by a mapping provides several prominent technological benefits. Since mappings (being *sets* of links) enjoy intersections and union operations over them, these operations can be transferred to transformation definitions encoded by mappings and employed for reuse. Since mappings (being sets of *directed* links) can be sequentially composed, we obtain a natural composition operation for transformations, which can be employed for their incremental compositional design.

Our plan for the paper is as follows. In Sect. 2, we first prove the semantic necessity of traceability mappings, and then discuss their properties and present categorical patterns for working with them. Section 4 shows how an MMT can be decomposed into, first, execution of a query against the source model, and then retyping the result by the target metamodel according to the meta-traceability mapping. In Sect. **??** we discuss algebraic operations over MMTs, related work is discussed in Sect. 5, and Sect. 6 concludes.

2 Taking traceability mappings seriously

After showing the semantic necessity of traceability mappings in Sect. 2.1, we consider their properties (structure preservation, commuting with typing, and span representation, Sect. 2.2), show how traceability mappings can be executed (Sect. 2.3), and finally discuss the underlying mathematical framework (Sect. 3).

2.1 The semantic necessity of traceability mappings

Semantics of a model-to-model transformation $\mathbf{T}$ is commonly considered (e.g., [**?**] and elesewhere) to be a function $[\![\mathbf{T}]\!]\colon [\![M]\!] \to [\![N]\!]$, where $[\![M]\!]$ and $[\![N]\!]$ are model spaces defined by, resp., the source, M, and the target, N, metamodels However, in this section we present two different transformations generating the same model space mapping, and then call traceability mappings to the rescue.

Figure 1(a) presents a toy transformation example. The source metamodel M specifies two classes, Car and Boat, and the target metamodel N specifies their possible roles as Commuting and Leisure Vehicles, connected by association same

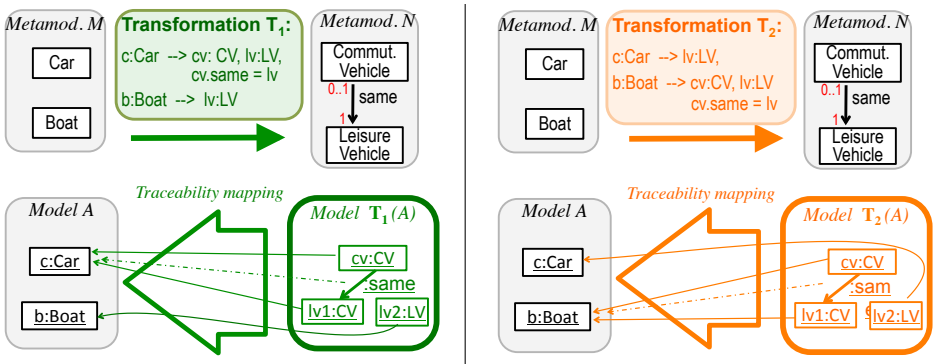

Fig. 1. Two sample transformations. FIX 1.CHANGE GREEN AND ORANGE FONTS TO BLACK. EVEN BLUE FONTS ARE TO BE DARKER/ASLO BLACK? FIX 2. WHY THE FIGS ARE OF DIFF HEIGHT? IT'S UGLY!!!! FIX 3. MOVE DEF. TERMS METAMOD OUT OF THE BOX? SEE THE NEXT FIGS AND DECIDE WHAT'S BETTER, BUT IT'D BE UNIFORM, RIGHT?

if two roles are played by the same physical vehicle; e.g., such a transformation may be needed for an insurance company. The transformation $\mathbf{T}_1$ consists of two rules specified in Fig. 1(a) in some pseudo MT language. The first rule says that a car produces a commuting vehicle and a leisure vehicle connected by a same-link. The second rule says that a boat generates a leisure vehicle. An example of executing this transformation for a model A consisting of a car and a boat is shown in the lower half of Fig. 1(a), where $\mathbf{T}_1(A)$ denotes the target model produced by the transformation, but please ignore the traceability mapping from $\mathbf{T}_1(A)$ to A for the moment. Here we write $\mathbf{T}(A)$ for $[\![\mathbf{T}]\!](A)$, and we will often abuse such a notation and use the same symbol for a syntactic construct and its intended semantics.

Figure 1(b) presents a different transformation $\mathbf{T}_2$. Now a boat gives rise to a commuting and a leisure vehicle, whereas a car only produces a leisure vehicle (think about people living on an island). Clearly, being executed for the same source model A, transformation $\mathbf{T}_2$ produces the same target model consisting of three objects and a same-link. More accurately, models $\mathbf{T}_1(A)$ and $\mathbf{T}_2(A)$ are isomorphic rather than equal, but as MMTs are normally defined up to OIDs, the same transformation $\mathbf{T}$ executed twice for the same model A would also produce isomorphic rather than absolutely equal models. We will always understand equality of models up to OID isomorphism, and thus can write $\mathbf{T}_1(A) = \mathbf{T}_2(A)$. It is easy to see that such an equality will hold for any source model containing equal numbers of cars and boats. However, let us suppose that the metamodel M includes a constraint requiring the numbers of cars and boats to be equal, eg, it may have an association between classes Car and Boat with multiplicity 1..1 at both ends. Then any instance X of M necessarily consists of equal numbers of cars and boats, and hence $\mathbf{T}_1(X) = \mathbf{T}_2(X)$ holds for *any* source instance $X \in [\![M]\!]$.

Thus, the common semantics of model transformations as model space mappings is too poor and should be enriched. Comparison of the two transforma-

tions in Fig. 1(a,b), now with traceability mappings, shows what should be done: we need to include traceability mappings into the semantics of model transformations, and define it as a function $[\![\mathbf{T}]\!]\colon [\![M]\!] \to [\![N]\!] \times Map([\![N]\!],[\![M]\!])$, where $Map([\![N]\!],[\![M]\!])$ denotes the set of all mappings from N-models (i.e., elements of $[\![N]\!]$) to M-models (in $[\![M]\!]$). The space of mappings is to be equipped with the *source* and *target* functions $\partial_{\mathbf{s}}\colon Map([\![N]\!],[\![M]\!]) \to [\![N]\!]$ and $\partial_{\mathbf{t}}\colon Map([\![N]\!],[\![M]\!]) \to [\![M]\!]$, and we write $m\colon A \leftarrow B$ for mapping m with $\partial_{\mathbf{t}}(m) = A$ and $\partial_{\mathbf{s}}(m) = B$. Of course, we require that if $(B, m) = [\![\mathbf{T}]\!](A)$, then $\partial_{\mathbf{s}}(m) = B$ and $\partial_{\mathbf{t}}(m) = A$. It is convenient to split semantics into two functions: $[\![\mathbf{T}]\!]^{\bullet}\colon [\![M]\!] \to [\![N]\!]$ and $[\![\mathbf{T}]\!]^{\blacktriangleleft}\colon [\![M]\!] \to Map([\![N]\!],[\![M]\!])$ such that for any source model A, $\partial_{\mathbf{s}}([\![\mathbf{T}]\!]^{\blacktriangleleft}(A)) = [\![\mathbf{T}]\!]^{\bullet}(A)$ and $\partial_{\mathbf{t}}([\![\mathbf{T}]\!]^{\blacktriangleleft}(A)) = A$. Thus, what is missing in the common MMT-semantics is the mapping-valued function $[\![\mathbf{T}]\!]^{\blacktriangleleft}$. However, including this function into semantics has several important consequences, which we discuss in the next section.

2.2 Traceability under the microscope

We discuss properties of traceability mappings: structure preservation, commuting with typing, and their span representation.

2.2.1 Structure preservation. A mapping is a collection of directed links that is compatible with models' structure. If models are graphs, then their graph structure, i.e., the incidence of nodes and edges, should be respected. The dashed link from edge same to node *c*:Car in the traceability mapping $\mathbf{T}_1{}^{\blacktriangleleft}(A)$ shown in Fig. 2 (whose lower level reproduces Fig. 1(a)) actually denotes a link targeted at the *identity* loop of the node *c*, which relates *c* to itself. Such loops can be added to every graph node, and when we draw a link from an arrow to a node, it us just a syntactic sugar to specify a link targeted at the identity loop of the target node. With this reservation, it's easy to see that both traceability mappings in Fig. 1 are correct graph morphisms, which map nodes to nodes and edges to edges so that the incidence between nodes and edges is preserved.

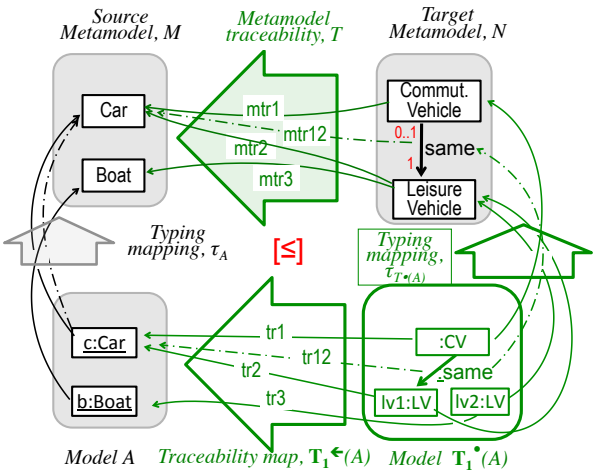

Fig. 2. Meta-traceability [w=2.5]

2.2.2 Meta-traceability and commutativity. Another important condition to be respected is compatibility of links between model elements with relationships between metamodel elements established by the transformation definition. To explicate the latter, we need *meta-traceability* links between metamodels as shown in the upper half of Fig. 2. The mapping T consists of four links mtr_i "tracing" the origin of the target metamodel elements according to definition $\mathbf{T}_1$ in Fig. 1(a): commuting vehicles appear from cars (rule 1) and only from cars (neither of the other rules produce commuting vehicles), and leisure vehicles appear either from cars (rule 1) or boats (rule 2). The dashed link to Car again denotes a formal link from edge same to not-shown identity loop link from Car to Car, and encodes the clause in rule 1 that a same-link appears when a car generates both a commuting and leisure vehicle. Overall, the upper three meta-links in mapping T "trace" rule 1 in transformation $\mathbf{T}_1$, and the lower link traces rule 2.

Now we need to recall that a model A is actually a pair $D_A, \tau_A)$ with D_A the model's datagraph, and τ_A: $D_A \to M$ the typing mapping. By a common abuse of notation, we denote the datagrapg by the same letter A as the entire model. Two typing mappings and two traceability mappings in Fig. 2 form a square, and it is easy to see that this square is *semi-commutative* in the sense that each of the four paths from $anmt_1^\bullet(A)$ to A (via traceability links tr_i) to M (via A's type links) can be matched by the same-source-same-target path from $\mathbf{T}_1^\bullet(A)$ to N (via type links) to M (via meta-traceability links mtr_i), but there is an upper path without match, namely, the path from object $lv1{\in}\mathbf{T}_1^\bullet(A)$ to class LeisureVehicle to class Boat (hence the $\leq$ symbol denoting this property of the square diagram). As commutativity is an important ingredient of the mapping machinery, we need to fix the commutativity violation. The next sections shows how it can be done.

2.2.3 Traceability mappings vis spans. As traceability links are fundamental, we reify them as model elements, and mapping T gives us three node-link nodes $mtr_{1,2,3}$ (see the upper part of Fig. 3), while the arrow-link (dashed) link $tr_{12}{\in}T$ is reified as an arrow mtr_{12} between the respective node-link reifications. Together, the four reifications form a metamodel $|T|$, consisting of three classes (for three inter-class links)

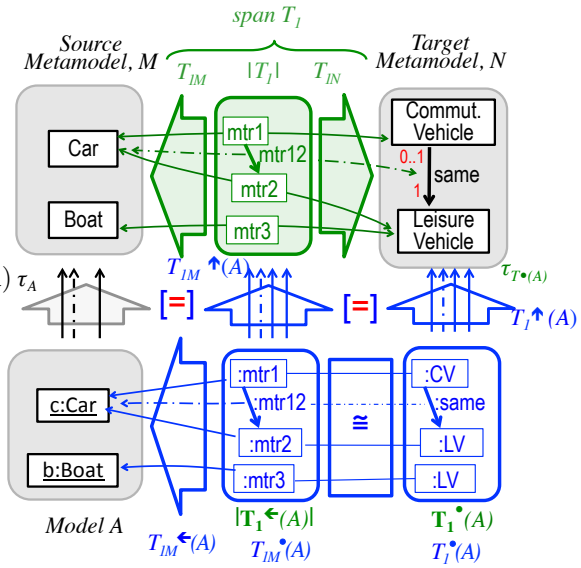

Fig. 3. Meta-traceability via spans. [w=2.]

and one association (for the inter-association dashed link). The special nature of $|T|$'s elements (which are links) is encoded by mapping each element to its ends in metamodels M and N. These secondary links form totally-defined single-valued mappings $T_M\colon M \leftarrow |T|$ and $T_N\colon |T| \rightarrow N$ so that we replaced a many-to-many mapping T by a pair of single-valued (many-to-one) mappings. Working with single-valued mappings is usually much simpler technically, and below we will see that it allows us to fix commutativity.

The triple $T = (|T|, T_N, T_M)$ is called a *span* with the *head* $|T|$, and *legs* T_M and T_N. Note that legs of the span are ordered and the triple $T^\circ = (|T|, T_M, T_N)$ is another span called the *inverse* of T. We will call the first leg in the triple the *source* leg, and the second one the *target* leg. Thus, span T encodes mapping T in Fig. 2, while span T° encodes the inverse mapping. We denote the span by the same letter as the mapping from which it is produced as they are essentially the same, and we will often use the same letter for the head of the span to reduce the number of symbols in our formulas. Note also that the head of the span is a graph (because mapping T is a graph mapping), and its legs are correct graph morphisms. This is an accurate formalization of the structure preservation property discussed in Sect. 2.2.1.

The reification procedure applied to mapping $\mathbf{T}_1{}^{\blacktriangleleft}(A)$ provides the span shown in the lower part of Fig. 3 (ignore the blue color for a moment, it weill be explained in the next subsection). Since in contrast to mapping T, mapping $\mathbf{T}_1{}^{\blacktriangleleft}(A)$ is many-to-one, the right (source) leg of the span is an isomorphism (of graphs), which we show as a block-rectangle rather than a block-arrow (actually we could identify the two models). Now it is easy to check commutativity of the two square diagrams, which is recorded by markers [=] at their centers. Commuting makes it possible to type elements in model $|\mathbf{T}_1{}^{\blacktriangleleft}(A)|$ (i.e., traceability links) by elements in model $|T|$ (i.e., meta-traceabilitylinks), and ensures that typing is a correct graph morphism. In other (UML's) words, meta-traceability links are classifiers for model traceability links, and commutativity provides consistency of traceability links' classification with model elements' classification. We have thus obtained an accurate formal specification of mutually consistent traceability mappings, but this is not the end of the story.

2.3 Meta-traceability links can be executed!

A somewhat surprising observation we can make now is that the meta-traceability mapping can actually replace the transformation definition $\mathbf{T}_1$: by applying two standard categorical operations to the span T and the typing mapping of model A, we can produce model $\mathbf{T}_1{}^{\bullet}(A)$ (together with its typing $\tau_{\mathbf{T}_1{}^{\bullet}(A)}$) and the traceability mapping $\mathbf{T}_1{}^{\blacktriangleleft}(A)$ in a fully mechanized way.

The first operation is called (in categorical jargon) *pull-back (PB)*. Its takes two graph mappings with a common target (a *cospan*), T_M and τ_A as its input, and outputs a span of graph mapping shown in Fig. 3 blank and blue (to recall the mechanic nature of the operation) so that the entire square diagram is commutative. The PB works as follows. For any pair of elements $a{\in}A$ and

$n \in N$ such that there is an element $m \in M$ together with a pair of links (ℓ_1, ℓ_2) targeted at it, $\ell_1 \colon a \to m$ in mapping τ_A and $\ell_2 \colon m \leftarrow n$ in mapping T_{1M}, an object (a, n) is created together with *projection* links to a and n. All such pairs (a, n) together with projection links to N make a model $T^{\bullet}_{1M}(A)$ (whose typing mapping is denoted by $T^{\uparrow}_{1M}(A)$), and projection links to A constitute its traceability mapping $T^{\leftarrow}_{1M}(A)$. The entire operation can be seen as pulling the model A together with its typing mapping back along mapping T_{1M}, hence, the name PB. Note that commutativity of the left square now becomes the very essence of the transformation: we build model A^* and its traceability mapping in such a way that commutativity holds. Moreover, we make this model a maximal model that respect commutativity by collecting in A^* *all* pairs (a, n) that respect commutativity. For example, if model A would have three cars and boats, model A^* would have three commuting and five leisure vehicles with three same-links.

The second operation is fairly easy: we sequentially compose mappings τ^*_A and T_N by composing their links, and obtain a mapping $A^* \to N$ that provides graph A^* with typing over N. We will denote the model $(A^*, \tau^*_A; T_N)$ by A_N, whose datagraph is A^* or its any isomorphic copy A^*_N according to our agreement to consider models up to OIDs isomorphism. This completes the right square in Fig. 3 (which can be seen as a syntactic sugar for the triangle of mappings described above). Now it is easy to see that PB followed by composition produce exactly the same model as rule-based definition $\mathbf{T}_1$ in Fig. 1(a), $A^*_N = \mathbf{T}_1{}^{\bullet}(A)$ (up to OIDs).

A bit more complex example out TR Sect.2.2.3 shows that metalink execution for graphs can be non-trivial for even simple examples.

2.2.4 Constraints for trace models. Paper [12,] provides an extensive in depth discussion of traceability modeling, and argues that stating proper constraints to trace links is an important component of traceability modeling. For example, suppose we state that the back-end multiplciity of mapping T_{1M} is 0..1, i.e., meta-trace mapping is injective (a.k.a. one-to-one mapping). Then, as it is known that PB preserves injectivity (in fact, more general multiplicity constraints), mapping $T^{\leftarrow}_{1M}(A)$ and hence $T^{\leftarrow}_1(A)$ must be also injective. If we obtain the trace mapping by executing PB, this constraint is automatically satisfied, but if the metatrace mapping is just an encoding of a rule-based transformation built for analysis purposes, and trace links will specially be generated by the tool during the run time, then injectivity appears as an important constraint for a proper storage and management of the trace links. Note that if the mutiplicity of the back-end of T_{1M} is 1 (i.e., T_{1M} is bijective), we cannot in general assert that $T^{\leftarrow}_1(A)$ is also bijective as the latter requires, in addition, surjectivity of the typing mapping τ_A.

In this way we can manage some multiplicity constraints but not all. For instance, in our example in Fig. 3, mapping T_{1M} is not injective, but its submapping responsible for Boat transformations is injective. In our example, the corresponding submapping of the trace mapping $T^{\leftarrow}_1(A)$ is injective too, but can we assert this for a general case? We need some machinery to define submappings

and investigate constraint preservation for submappings. We will do it in the next section.

3 Mathematical framework.

3.1 Getting started. An abstract schema of the examples we considered above is shown in Fig. 4. All metamodels are instances of a fixed metametamodel $\mathcal{MM}$ (think of, e.g., graph Node $\leftleftarrows$ Arrow, but more complex graphs are not excluded), which provides meta-metatypes/classes (we will say mm-type or mm-class) and appears as the target of all typing mappings $\tau\tau$[3]. Commutativityof the triangles ensures structure preservation: an element instantiating mm-class Node in T is mapped to an instance of Node in M etc. Models are also (implicit) instances of $\mathcal{MM}$, but typing mapping $\tau\tau$: $A \to M$ is not shown. To call a graph A an *instance of metamodel* A, we need to equip it with a typing mapping τ_A: $A \to M$ such that $\tau_A; \tau\tau_M = \tau\tau_A$ (imagine a commutative triangle diagram), which ensures τ_A preserves the structure defined by $\mathcal{MM}$.

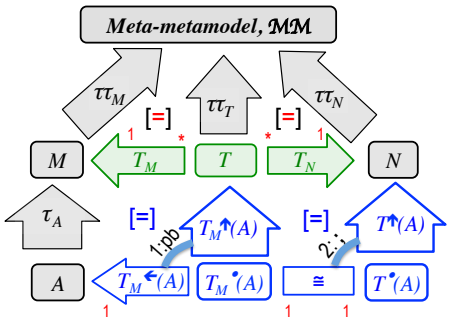

Fig. 4. Meta-traceability and model transformations. Derived nodes and arrows are blank (and blue).

The left square in the diagram is a PB application: two input mappings (together with their source and target objects) are shaded, the output elements are blank (and blue). The arc 1:pb says that at step 1 operation pb was invoked and produced two mappings spans by the arc. PB always results in a commutative diagram, hence the marker = is blue (it's a postcondition rather than a constraint). Recall that the argument (A) in the names of the blue elements refers to the entire model $A = (D_A, \tau_a)$ rather than its data graph (see footnote **??**).

The right square show an application (step 2) of the sequential compositon *;* operation (and the bottom rectangle mapping $\cong$ can be considered identity) followed by an optional isomorphic coping (and then the bottom rectangle is an iso). The MMT operation over model A is the composition (tiling) of PB and *;*—it takes the entire (green) span T and mapping τ_A as its input, and outputs the bottom (traceability) span and mapping $T^{\uparrow}$ (A).

Our next step is to describe how this schema can be formalized. We will begin with an outline of meta- and metamodeling, and then discuss pullbacks. We will not include constraints into the picture—managing and formalizing general constraints is a special story that goes beyond this paper, but we will discuss simple but practically important multiplicity constraints.

3.2 Meta(-meta)modeling. We begin with fixing a category $\mathbb{G}$ whose objects and arrows are called f *graphs* and *graph mappings*. In the abstract setting, we

[3] read $\tau\tau$ as doubled τ rather than π

only require that $\mathbb{G}$ has pullbacks. But we would also like keep some set-and -function intuition motivating the formal constricts, and then *graphs* can be thought as ordinary (directed multi-) graphs, or edge- and node-labeled graphs, or, 2-graphs (with arrows between arrows) so that pullbacks are defined componentwise (for nodes, for arrows, for 2-arrows).[4] The reader may think about $\mathbb{G}$-graph as ordinary graphs, and this is what we want to achieve by calling $\mathbb{G}$-objects graphs.

Let $\mathcal{MM}$ be a graph considered as a meta-metamodel so that metamodels in the sense of sections **??**-2.3 above are *instances* of $\mathcal{MM}$, i.e., pairs $M = (D_M, \pi\pi_M)$ with D_M a *metadata graph* and $\pi\pi_M\colon D_M \to \mathcal{MM}$ a *typing* mapping[5], which is a correct graph morphism. In our notation in sections **??**-2.3 we followed a common (inaccurate) practice and denoted datagraphs by the same letters as (meta)models so that M will denote D_M of the entire pair depending on the context. The primary example to have in mind is again the metamodel for ordinary graphs $\mathsf{Node} \leftleftarrows \mathsf{Arrow}$, or better a (meta)metamodel for class diagrams but more complex graphs are not excluded[6] Interesting diversity appears on the level of metamodels, e.g., M can be a metamodel for class diagrams, and N a metamodel for relational tables, but both M and N are graphs (or, in the refined setting, class diagrams) instantiating $\mathcal{MM}$. We will denote the category of all (double $\pi\pi$ typing) mappings into $\mathcal{MM}$ as objects, and triangle commutative diagrams as arrows, by $[\![\mathcal{MM}]\!]^?$ with the upper index showing that objects of this category *can* be legal metamodels if, in addition, they satisfy some extra constraints, e.g., of being finite graphs.[7] In general, legal metamodels make a subcategory $[\![\mathcal{MM}]\!]$ of $[\![\mathcal{MM}]\!]^?$.

In its turn, an object $M = (D_M, \pi\pi_M) \in [\![\mathcal{MM}]\!]^?$ determines its own instance category $[\![M]\!]$, whose objects are (single τ) typing mappings into M, say, $\tau\colon D_A \to D_M$, and arrows are mappings $f\colon A \to B$ such that $f;\tau_B = \tau_A$ (imagine a commutative triangle again). We denote this category by $[\![M]\!]^?$ since again its objects can be, but not necessary are, legal instances—the latter must satisfy constraints declared in M (so that actually M's datagraph D_M contains a non-instantiatable constraint part; details of this construction can be found in [1]). Thus, the category $[\![M]\!]$ of M's legal instances is a subcategory of $[\![M]\!]^?$.

Now let M, N be two metamodels, and T a mapping (in $[\![\mathcal{MM}]\!]$) between them. Then for any instance $A \in [\![M]\!]$, we apply PB to the pair (T, τ_A) and obtain a graph $T^\bullet(A)$ together with its trace mapping $T^{\leftarrow}(A)$ and typing mapping $T^{\uparrow}\ (A)$, i.e., we define $\pi\pi_{T^\bullet(A)} \stackrel{\text{def}}{=} T^{\uparrow}\ (A); \pi\pi_N$. Thus, we have a mapping $T^\bullet\colon [\![M]\!] \to [\![N]\!]^?$ that constitutes the object part of MMT but as we have seen, the traceability part is not less important. (Note that the result of transforma-

[4] a categorician would say that $\mathbb{G}$ is a presheaf topos, whose base category is freely generated by a graph.

[5] read $\pi\pi$ as double-τ rather than π

[6] to be logically accurate, category $\mathbb{G}$ is the external category providing the universe in which everything else operates.

[7] In category theory, such categories are called *slice* and often denoted by $\mathbb{G}/\mathcal{MM}$.

tion is only a pre-instance of N as we cannot, in general, guarantee that the target constraints are satisfied.)

3.3 Pullback operation under the microscope. We will present several basic results about PBs (one of them seems to be new), and show how they can usefully be applied for several questions considered in the MMT literature. We begin with PB-construction in the category Set os sets and functions (i.e., total single-valued mappings).

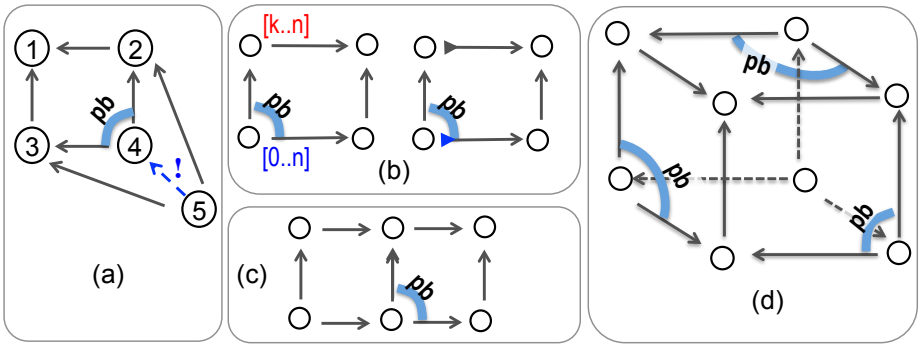

Fig. 5. Lemmas about pullbacks: (a) UP, (b) MP, (c) Pasting, and (d) VK.

Definition 1 (pullbacks in Set). Given two functions with a common target, $f_1\colon X_1 \to Y$ and $f_2\colon Y \leftarrow X_2$, their *pullback, PB* is a pair of functions with a common source, $g_1\colon X_1 \leftarrow Z$ and $g_2\colon Z \to X_2$ where $Z = \{(x_1, x_2) \in X_1 \times X_2 \colon\ f_1(x_1) = f_2(x_2)\}$, and g_i are projections: $g_i(x_1, x_2) = x_i$, $i = 1, 2$. It is esasy to see that $g_1; f_1 = g_2; f_2$, i.e., the square diagram formed by functions is commutative.

The following two results are well known [3].

Lemma 1 (pullbacks in Set).
(a) Universal property, UP. For a given pb-square 1234 *Fig. 5(a) and an outer commuting rectangle* 5312*, there is one and only one mapping* $!\colon 5 \to 4$*. Conversely, if square* 12324 *possesses the universal property, then it is a pb-square according to Definition 1.*
(b) Muiltiplicity preservation, MP. For a given pb-square, if multiplicity of the upper mapping is [k..n], the multiplicity of the lower mapping is [0..n].

The universal property allows us to define PB for complex structures like graphs or Petri nets *componentwise* by separately defining PB for the constituent sets: nodes, arrows, etc. The the existence of necessary mappings like the source and the target nodes for arrows in directed mutligraphs is provided by UP. Details can be found in many textbooks, e.g., [3].

Now we switch to abstract categories whose objects and arrows are black-boxes; the only fact we know about them is that arrows can be associatively composed, and each node has an identity loops, which are units of the arrow composition. Definition 1 does not work for this setting—there are no elements,

but we can take the result of Lemma 1(a) as a definition of PB: a commutative square like in Fig. 5(a) is PB if it has the universal property described in the lemma. It is easy to prove that for any given span of mappings $(31, 21)$, its PB completion up to a square 1234, if it exists, is unique up to canonic isomorphism between nodes 4 (see again [3] for details).

For considering our practical scenarios, we will need the following two results. The first is a well-known Pasting Lemma (see, e.g., [6]). For the diagram in Fig. 5(b) in which the right-hand square is PB, the following holds: the left-hand square is PB iff the outer rectangle is PB (the outer rectangle is logically a square, if we compsoe the two upper arrows into one upper arrows, and similarly for the two, lower arrows).

The second result is although simple, but (surprisingly) seems has not been explicitly stated in the literature. We call it *PB 3D-pasting lemma.*

Lemma 2 (3D-pasting lemma for PBs). *Take any category with PBs, and consider the cubical commutative diagram in Fig. 5d), whose top face is a PB, and the two front faces (those are marked with arcs) are PBs too. Then the following holds: the bottom square is a PB iff one of the back faces is a PB (and hence the other back face is a PB too).*

The proof is in the appendix, and it is a typical categorical proof by chasing diagrams based on iterative applications of the universal property and 2D-pasting Lemma.

Now we consider two scenarios of MMT management, in which the PB-machinery we presented is essentially employed.

Scenario 1. Sub-transformation and constraints. Consider the front face of the prism diagram in Fig. 6: we have a metamodel M, an (unnamed) MMT definition mapping TM, a model A over M and PB execution gives us the transformed model $T^{\bullet}(A)$ with its trace mapping to A.

Now suppose that we are interested in some local property of transformation T, e.g., for the example in Fig. 3, mutliplicity of the transformation part related to boats. Hence, we select a subgraph T_0 of the transformation span head T: a special arrow with triangle tail denotes the embedding of this subgraph into grah T. Now we will consecutively perform several operations over mappings and their properties to analyse the situation.

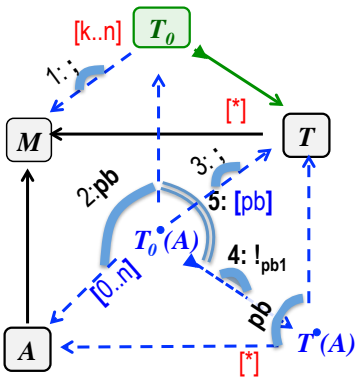

Fig. 6. An MMT scenario via PBs.

Step 1: We compose mappings T_0T and TM, which gives us mapping T_0M: we encode this data by making the arrow dashed (blue) and attaching to it a small arc labeled 1:; to be read "at step 1 operation ; was invoked and produced the mapping captured by the arc".

Step 2: is to apply PB to the pair of mappings AM and T_0M and thus derive two mappings captured by the arc 2:pb.

Step 2+: apply MP Lemma and deduce mutliplicity for mapping $T_0^\bullet(A)$.
Step 3: compose mappings $T_0^\bullet(A)T_0$ and T_0T and get arrow $T_0^\bullet T$.
Step 4: note that pair $T_0^\bullet(A)A, T_0^\bullet(A)T$ completes the span AM, TM up to a commutative square, and hence, by the universal property of PB, there is a unique mapping $T_0^\bullet(A), T^\bullet(A)$.
Step 5: note that the right-hand back face tiled with the front face conform to the situation of the Pasting Lemma (see Fig. 5c)with the left-hand back face being the outer rectangle. Hence, the right-hand square is PB. As PBs provide co-images, it means that model $T_0^\bullet(A)$ derived by executing subtransformation T_0, is simultaneously the submodel of $T^\bullet(A)$ corresponding to the T_0 part. In words: any subtransformation results in the respective submodel of the transformation. This statement has obvious practical consequences as it allows us to obtain the result of subtransformation wihout executing it! Finally,
Step 5+: ensures that mapping $T_0^\bullet(A), T^\bullet(A)$ is injection.

Scenario 2. Sub-metamodel and constraints. The front face of the prism diagram in Fig. 7 is the same: we have a metamodel M, an MMT definition mapping TM, a model A over M and PB execution gives us the transformed model $T^\bullet(A)$ with its trace mapping to A.

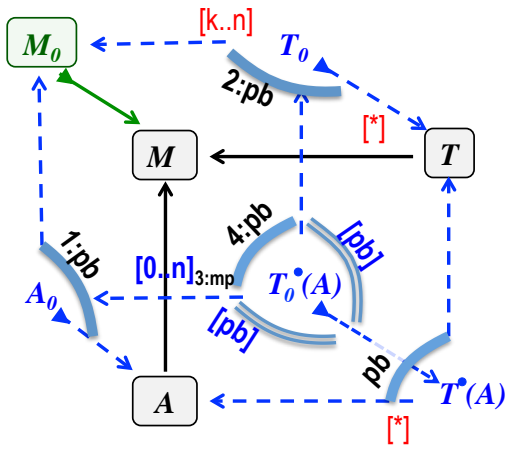

Fig. 7. An MMT scenario with PBs.

4 Towards a general approach: Transformation via queries

Pulling a source model A back along a meta-traceability mapping T_1 produces one or multiple copies of A's elements, which covers a useful but not too wide class of transformations; more complex transformations need a more expressive mechanism. In [4,7], it was proposed to separate an MT into two parts: first, a complex computation over the source model is encoded by a query against the source metamodel, and then the result is relabeled (with, perhaps, multiplication) by the target metamodel according to the meta-traceability mapping.

4.1 Traceability with queries We will illustrate how the machinery works by encoding the same transformation $\mathbf{T}_1$ by a different type of meta-traceability mapping employing queries against the source metamodel as shown in Fig. 8. The

first basic idea of the transformation—creation of commuting vehicles by cars only—is encoded by direct linking class Commut.Vehicle to class Car as we did before. The second idea—creation of leisure vehicles by both cars and boats— is now encoded in two steps. First, we augment the source metamodel M with a derived class Car + Boat computed by applying the operation (query) of taking the disjoint union of two classes; we denote the augmented metamodel by $Q(M)$ with Q referring to the query (or a set of queries) used for augmentation. Second, we link class LeisureVehicle to the derived class Car + Boat, and association same in metamodel N is linked to its counterpart in metamodel $Q(M)$, as shown in Fig. 8.

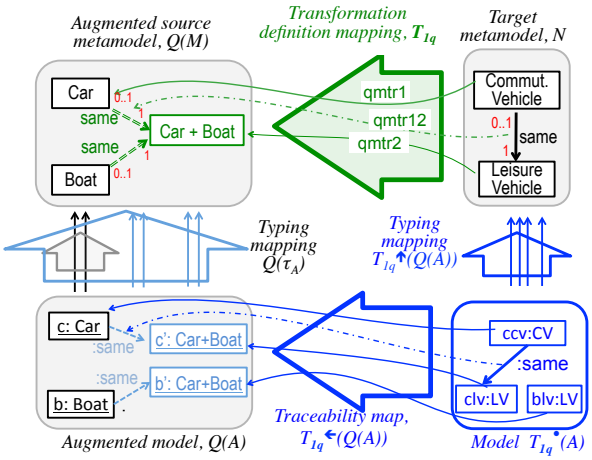

Fig. 8. Meta-traceability via queries [w=2.5]

All links have a clear semantic meaning: given a link *qmt* from an element n of N to an element m on $Q(M)$, we declare that n is to be instantiated exactly as m is instantiated, that is, for any model A, every element instantiating m in A or $Q(A)$ (see below), generates an element instantiating n in $T_1^\bullet(A)$. Note also that the mapping is of one-to-one type: two classes responsible for LeisureVehiclegeneration now contribute to a single query, and two respective links (*mtr2* and *mtr3* in Fig. 2) are replaced by one link *qmtr2* into the query.

Execution of the transformation for a model A also goes in two steps. First, the query used in the mapping definition is executed for the model. In our example, we take the disjoint union of Car and Boat instantiations in A, i.e., the set $\{c', b'\}$. (A reasonable implementation would add a new type Car + Boat to the same object c rather than creating a new object c', but the pair (c, Car) is still different from pair $(c, \mathsf{Car}+\mathsf{Boat})$. Thus, it may happen that $c' = c$ and $b' = b$, but this is just a special case of a general pattern presented in Fig. 8.) Second, objects c, c', b' are retyped according to the respective meta-traceability links $qmtr_1$ (for c) and $qmtr_2$ (for c' and b'). The link cc' is also retyped along the link $qmtr12_{12}$.

Thus, a model transformation definition is divided into two parts: finding a query (or a set of queries) Q against the source metamodel M, which captures the computationally non-trivial part of the transformation, and then mapping the target metamodel into the augmentation $Q(M)$, which shows how the results of the computation are to be retyped into the target metamodel. The second

part can capture some simple computations like multiplication of objects (which appear in MMT surprisingly often), but not more. In contrast, with a broad understanding queries as general operations, the first part is Turing complete with the only reservation that all result of the computation must have new types (which distinguishes queries from updates; a detailed discussion and an accurate formalization can be found in [5]).

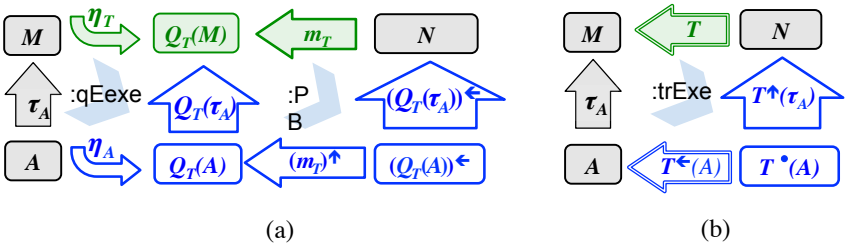

Fig. 9. Meta-traceability and model transformations. Derived nodes ands arrows are blank (and blue).

4.2 Mathematical abstraction. A formal abstraction of the example is described in Fig. 9(a). A model transformation is considered to be a pair $T = (Q_T, m_T)$ with Q_T a query against the source metamodel M and $m_T\colon Q_T(M) \leftarrow N$ a mapping from the target metamodel N to model M augmented with derived elements specified by the query. Formally, we have an inclusion $\eta_T\colon M \hookrightarrow Q_T(M)$. On the level of metamodels, query Q_T is a query definition, which can be executed for any data conforming to schema M, i.e., for any model properly typed over the metamodel M. Execution is modeled by an operation qExe, which for a given query Q_T and model A produces an augmented model $Q_T(A)$[8] properly typed over the augmented metamodel by an augmented typing mapping $Q_T(\tau_A)$. To complete the transformation, the result of the query is retyped according to the mapping m_T (retyping is given by pulling back the augmented typing mapping as discussed above). In Fig. 9(b), an abstract view of Fig. 9(a) is presented, in which the upper double arrow encodes the two upper arrows in diagram (a), and operation trExe of *transformation execution* is a composition of two operations in diagram (a). Paper [5] presents an accurate formalization of this construction by modeling the query language as a monad, view definitions as Kleisli arrows over this monad, and view executions as Cartesian lifting.

5 Related Work, Discussion, and Future work

5.1 Traceability modeling. Traceability understood broadly is an enormous area [2,12] in the present paper we consider its special sub-area connected with MMT. However, in MDE, which is vitalized and driven by model transformations, this special sub-area is a central one. Consider- ing traceability as a special artifact that itself important and deserves (meta)modeling (and even a special

[8] we should write $[\![Q_T]\!](A)$ but we again use the same symbol for both syntactic and semantic constructs.

DSL for building such metamodels) is now well acknowledged by the community: see [12] for an extensive in-depth discussion and motivation.

Table 1 shows correspondences between a common terminology now used in the literature (in the left column) and constructs used in this paper (the right one, where commuting refers to Fig. 4. The counterparts are even technically close as evidences by Fig. 9, in which we present the trace model of our example in Fig. 3 in the style borrowed from [9]. Abstract class traceRe- cord provides several general attributes (such as creationDate and the like), and its subclasses represent elements in the head of our meta-trace span T. In fact, Fig. 9 can be seen as a special representation of spans, in which commutativity conditions are replaced by link inheritance (only two of them (with dotted lines) are shown in the figure). Comparison of the categorical model in Fig. 4 and the metamodel in Fig. 9 shows two advantages of the former. First, commutativity seems to be a more economic notation for the same semantics: imagine all link inheritance arrows are shown in Fig. 9; the more so if we agree to consider all possible commuting diagrams to be commutative by default, and specially mark non- commutativeones (which is a usual category theory practice). Second, UML does not allow for associations between attributes, and the latter need to be reified, which would complicate the metamodel even more. Moreover, Fig. 9 (and its counterparts in the literature) miss the important structure preservation conditions for ref source and ref target associations, which are expressed by commutativity of the triangle diagrams in Fig. 4.

common terms	this paper
trace record	element of the trace span
trace model	trace mapping as a span
case-specific trace metamodel	trace metamapping as a span (with commuting rectangles)
general purpose trace metamodel	meta-metamodel (with commuting triangles)

Table 1. Dictionary [w=1.5, 2.5,4

Trace constraints are important for trace (meta)modeling and deserve a special discussion. Two major types of constraints are identified in [12, Sect.4]: type-safety and case-specific correctness. In our framework, the former is ensured by commutativity constraints,a nd the latter is provided by the localization ma- chinery (submappings) explained in Sect. 3.3. Note that as our trace metamodel (i.e., span) is executable, we can ensure satisfiability of trace constraints if the respective constraints are properly stated on the meta-trace level and PB pre- serves them. This could free the user from the burden of watching whether these trace constraints are satisfied. Of course, not all con-

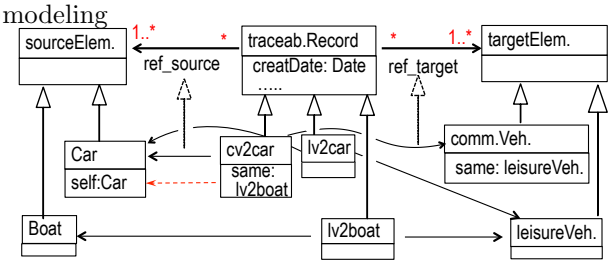

Fig. 10. Traceability metamodel [w=2.]

straints can be treated in this way, but two exemplar constraints considered in [12, Sect.4] can be handled in our framework: one is a local multiplicity constraint, and the other can be captured by introducing a respective arrow into the meta-trace span (and us- ing a simple query (arrow composition) in one of the metamodels (out TR [Sect.4] provides details).

5.2 Execution of meta-traceability links. Executability of meta-trace links is considered in now well-developed framework of triple-graph grammars (TGG), which provide explicit trace-link modeling with correspondence models [8]. However, as mentioned in Sect.1, TGG transformations are defined by sets of transformation rules operating over individual model elements, while in the view presented in the paper, an entire transformation is defined by a single meta-traceability mapping, involving operations (queries) over sets of model elements. This essential distinction seems possessing a non-trivial mathematical re ection: TGG is based on pushouts, while our framework is based on pullbacks dual to pushouts in some precise technical (and deep) sense. Further exploration of the duality between elementwise TGG and set-based approach of the present paper may be an intriguing research problem.

An industrial standard QVT is a technological counter-part of TGG, in which links specifying inter-model relationships are executed. Paper [13] provides an in-depth discussion of traceability in the context of QVT-rules execution, and hence executability of meta-traceability links. The machinery employed is described informally, but seems close to our use of PB. The overall picture is broader than ours and includes mapping refinement, dynamic dispatch, and concurrency. A precise formalization of these constructs in terms of our framework would be a useful application; we leave it for future work. Separation of transformation into the query part, and retyping (the PB part), is not considered.

5.3 Traceability with queries and sequential composition. The separation mentioned above is discussed in [7], but the expressiveness of PB seems underestimated; particularly, the many-to-many traceability mappings are not considered. A precise formalization of traceability mappings with queries in categorical terms as so called Kleisli mappings is provided in [5], sequential composition then follows from Kleisli mapping composition. However, the general context for paper [5] is general inter-model relationships (which corresponds to a broad view of traceability as correspondence emphasized in [2]), while in the present paper we are more focused and consider traceability in the MMT context.

5.4 Algebra for mmts. In neither of the works mentioned above, Boolean operations over transformations are not considered, and we are not aware of their explicit introduction and discussion in the literature. An important future work is to enrich the formal framework developed in [5] with Boolean operations for traceability mappings.

6 Conclusion

A list of idea the reader is expected to take home from reading this paper is as follows.

−Traceability belongs to semantics, not only to technology.
−Traceability mappings have several important properties:
−Meta-traceability mappings can be executed. Moreover, transformation definitions can be represented by traceability mappings. Moreover, algebra of mappings gives rise to an algebra of transformations.
−Working with mappings needs a certain notational and formal discipline provided by category theory.

References

1.
2. N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.
3. M. Barr and C. Wells. *Category theory for computing science*. Prentice Hall, 1995.
4. Z. Diskin. Model synchronization: Mappings, tiles, and categories. In J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, editors, *GTTSE*, volume 6491 of *LNCS*, pages 92–165. Springer, 2009.
5. Z. Diskin, T. Maibaum, and K. Czarnecki. Intermodeling, queries, and kleisli categories. In J. de Lara and A. Zisman, editors, *FASE*, volume 7212 of *LNCS*, pages 163–177. Springer, 2012.
6. P. Freyd and A. Scedrov. *Categories, Allegories*. Elsevier Sciece Publishers, 1990.
7. H. Gholizadeh, Z. Diskin, and T. Maibaum. A query structured approach for model transformation. In J. Dingel, J. de Lara, L. Lucio, and H. Vangheluwe, editors, *Proceedings of the Workshop on Analysis of Model Transformations co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2014), Valencia, Spain, September 29, 2014.*, volume 1277 of *CEUR Workshop Proceedings*, pages 54–63. CEUR-WS.org, 2014.
8. H. Giese and R. Wagner. Incremental model synchronization with triple graph grammars. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*, volume 4199 of *Lecture Notes in Computer Science*, pages 543–557. Springer, 2006.
9. Á. Hegedüs, Z. Ujhelyi, I. Ráth, and Á. Horváth. Visualization of traceability models with domain-specific layouting. *ECEASST*, 32, 2010.
10. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.
11. D. S. Kolovos, R. F. Paige, and F. Polack. The epsilon transformation language. In A. Vallecillo, J. Gray, and A. Pierantonio, editors, *Theory and Practice of Model Transformations, First International Conference, ICMT 2008, Zürich, Switzerland, July 1-2, 2008, Proceedings*, volume 5063 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2008.
12. R. F. Paige, N. Drivalos, D. S. Kolovos, K. J. Fernandes, C. Power, G. K. Olsen, and S. Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Software and System Modeling*, 10(4):469–487, 2011.
13. E. Willink and N. Matragkas. QVT Traceability: What does it really mean? In *Analysis of model transformations, AMT'15*, 4th Workshop Models'15, 2015.