Forschungsberichte der FHDW Hannover

Consistency Checking of Interrelated Models, Long Version

Harald König, Zinovy Diskin

Bericht Nr.: 02017/01

Fachhochschule für die Wirtschaft Hannover
Freundallee 15
30173 Hannover
techrep@fhdw.de

UNIVERSITY OF APPLIED SCIENCES
FHDW
FACHHOCHSCHULE FÜR DIE WIRTSCHAFT
HANNOVER

Impressum

Forschungsberichte der FHDW Hannover – Veröffentlichungen aus dem Bereich Forschung und Entwicklung der FHDW Hannover.

Herausgeber: Die Professoren der FHDW Hannover
Fachhochschule für die Wirtschaft Hannover
Freundallee 15
30173 Hannover

Kontakt: techrep@fhdw.de

ISSN 1863-7043

Consistency Checking of Interrelated Models, Long Version ⋆

Harald König[1] | Zinovy Diskin[2,3]

[1] University of Applied Sciences FHDW Hannover, Germany
[2] Generative Software Development Lab, University of Waterloo, Canada
harald.koenig@fhdw.de
zdiskin@gsd.uwaterloo.ca

Abstract. Software design requires deployment of a collection of interdependent models conforming to different metamodels. These so-called *multi-models* present different views of interest and may be consistent only if they simultaneously satisfy a set of global constraints.
A straightforward approach to global consistency checking is to run constraint validations on the model union (merge). These validations are carried out by reduction to a small view (localization) of the model merge. It turns out that this "merge-prior-to-localization"-approach is not efficient because of considerable matching and merging workload. We propose to perform *early localization* in order to reduce the data space which is subject to the search for commonalities. The algorithm is based on a new method for formally specifying the inter-relation of an arbitrary number of heterogeneously typed models. Behind the scenery we utilize valuable theorems of category theory.

1 Introduction

Modeling a complex system normally results in a *multi-model*, i.e. a set of heterogenous models each one conforming to its own metamodel. A fundamental fact about multi-modeling is that the merge of legal models can result in a model violating the global constraints declared in the integrated metamodel. For example, consider a metamodel M_1 in which car insurance contracts possess an attribute 'contractType' (with values 'standard', 'extended'), and a metamodel M_2, in which class 'Policy' possesses a property 'traffic telematics enabled'. Suppose that – despite their different names – concepts 'Contract' and 'Policy' denote the same business concept. Then the domain may be subject to the constraint that only extended contracts can be controlled via traffic telematics. This *global* constraint cannot be declared in either of the metamodels (the first one knows nothing about telematics, the second one does not know about contract types), yet checking its validity for a multi-model (τ_1, τ_2) with $\tau_{1,2}$ being legal instances of $M_{1,2}$ is important. Following [7], we call such requirements *inter-metamodel constraints*.

The interaction of software models due to the fact that they may (partially) describe identical concepts requires understanding and formalizing *matches* (i.e. sameness declarations) between these artefacts (e.g. sameness of classes 'Contract' in M_1 and 'Policy'

⋆ This work is partly supported by Høgskolen i Bergen and FHDW Hannover

in M_2). Moreover, checking of a single constraint on the entire multi-model requires *localization* of the constraint's affected part. This localization has to be carried out on the formally defined *merge* of the model components with simultaneous consideration of all sameness declarations.

In [11] the interplay of these three *operations* on model collections was investigated:

- *Matching* has a multi-model with unrelated model components as input and outputs this collection together with all detected sameness relations.
- *Merge* has a multi-model with already related components as input and outputs a single model, namely its union modulo the above mentioned sameness declarations.
- *Localization* has a constraint declaration on a metamodel as input and outputs the affected fragment of this metamodel.

A straightforward approach to global consistency checking would require to match and merge the multi-metamodel (yielding contracts/policies with type *and* telematics information in the above example) and adding, perhaps, new global constraints to this merge ('only extended contracts can be controlled via traffic telematics'). [19] describes how to check consistency of component models $\tau_{1/2}$ against these global constraints: One matches and merges component models $\tau_{1/2}$ in the same way and then check the model merge against the constraints over the metamodel merge with the help of localization. In fact, this specification can be regarded as a *definition* of global consistency of a multimodel. However, using this definition algorithmically as a specification of a workflow for global consistency checking would be impractical because of (a) partly manual, hence costly model matching needed to specify the overlaps, and (b) the necessity to build big and unfeasible merges of metamodels and models.

A more efficient approach is *early localization*: In [11] it was proposed to localize first and perform matching and merging afterwards. Not only does this significantly reduce matching and merging workload, it also enables better tailored and stepwise model repairing e.g. in model co-evolution scenarios. However, [11] only covers the case of two model components, and the formalization of model matching was carried out with the help of auxiliary model τ_0, in which *pairs* of matched entities (e.g. contract instance with $id = 42$ in τ_1 equals policy instance with $id = 42$ in τ_2)) were defined.

In the present paper, we keep a promise of [11]: We implement the announced generalization of early localization for the case of an arbitrary number of metamodels. For this it is not enough to maintain pairs of common concepts in τ_0. Instead it would be necessary to enable binary, ternary and higher-order relations which may lead to a rapid increase of the number of auxiliary structures. Thus the first challenge is to find a more efficient method to specify these complex inter-relations. The second challenge is to distribute semi-automatic matching activities efficiently between consecutive automatic localization and merging, such that corresponding workload is minimized.

In this context, the present paper makes two essential contributions: (1) The generalization of early localization to an arbitrary number of metamodels by extending the above mentioned localization operation to heterogeneous input and (2) an improved methodology to relate n models in one common and small auxiliary model.

We introduce a running example in Sect.2 which illustrates all subsequent considerations. In Sect. 3 the basic and fundamental formalization of digrammatic constraints is explained in detail. We follow the original contributions in the literature ([5] and

subsequent). The introduced constraint checking formalization is transferred to Multi-Models (which are explained in Sect.4) in Sect.5, where the original global approach is compared with the new local algorithm. A short conclusion completes the present paper.

The presented theory is the conceptual basis for a running project at FHDW Hannover ('Integrationsprojekt', HFW414) in the year 2017.

2 Running Example

In this section we introduce the main example which serves as illustration for all constructions we perform and facts we declare: Certain authorities are interested in social clustering of inhabitants of a city or a country or even how people around the globe flock together. Social networks provide the chance to observe, measure and monitor this clustering. We consider the UML domain models M_1 and M_2 of two social networks sn_1 and sn_2 as depicted in Fig.1.

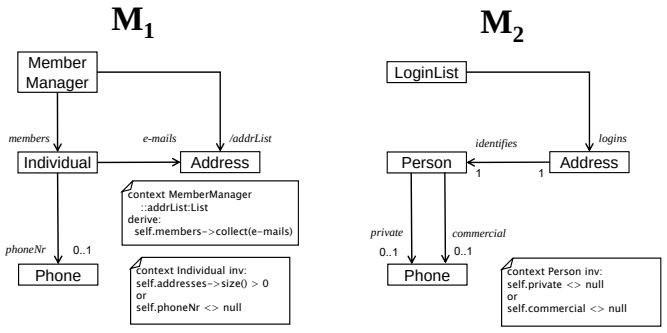


Fig. 1: Domain Models M_1 and M_2 of social networks sn_1 and sn_2

M_1 specifies that a member manager object maintains all members of sn_1, the collection of all their e-mail-addresses being derivable by the manager object. Each individual can provide a phone number. If there is no phone number, at least one e-mail has to be provided. M_2 specifies maintenance of e-mail-addresses that serve as login for network sn_2. Thus, each e-mail-address identifies a person of the network. These persons have to provide a private or a commercial phone number.

Note that we do not provide multiplicities at each association end. If we do not, multiplicity defaults to $0..^*$.[3] Constraints are given in OCL syntax, e.g. the fact that 'addrList' is derived, the derivation logic being specified by the first constraint in M_1.

[3] This is in contrast to UML's default setting, which is 1, but for the forthcoming considerations it is more convenient to deviate from UML standard.

Suppose that for several authorities it has always been important to monitor the amount of e-mail-communication from sn_1 to sn_2: One intelligence apparatus has stored all recorded e-mails according to the domain model M_3, but there is also another secret service, who stored all e-mail-contacts between social networks due to domain model M_4, see Fig.2.

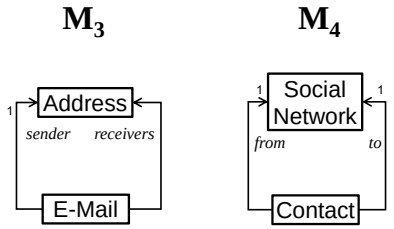


Fig. 2: Domain Models M_3 and M_4 of two intelligence agencies

The two intelligence agencies decide to collaborate in order to consolidate their knowledge. It is a goal to find out whether their collective data is consistent, e.g. they must ask whether the number of tracked contacts from sn_1 to sn_2 (recorded in the second agency) equals the number of e-mails with sender an individual of sn_1 and one of the receivers being a person in sn_2 (recorded in the first agency). Assume our two focused social networks appear as two objects sn_1 and sn_2 of type 'Social Network' in instances of M_4. Then this *inter-model constraint* can formally be written as

$$\begin{aligned} &|\{c \in Contact \mid c.from = sn_1 \wedge c.to = sn_2\}| \\ = &|\{e \in E-Mail \mid e.sender \in sn_1.addrList \wedge e.receivers \cap sn_2.logins \neq \emptyset\}| \end{aligned}$$

assuming appropriate relations between all 4 models: It is for instance reasonable to identify objects of type 'Social Network' (e.g. $sn_{1/2}$) in M_4 with managing objects of all members and all logins, resp. in models $M_{1/2}$.

In the next sections we explain, how to code models as graphs, how to impose constraints on these graphs, and how constraint checking is performed in detail. Moreover, we will explicate model interrelations specifying sameness of overall concepts and how this formally triggers model union (i.e. merge). Finally, we elaborate on efficient constraint checking on multi-models. All aspects will formally be explained and exemplified by means of the above example.

3 Diagrammatic Constraints

(Meta-)Models are usually specified by UML class diagrams. The compact syntax of the latter hides many details that need to be explicated and formalized to allow the forthcoming machinery to work. In this section, we show how it can be done in the formal framework of typed graphs (e.g. [8], Sect. 3.1) and diagrammatic constraints.

To be self-contained, the formalisms for *diagrammatic constraints*, promoted as the *Diagram Predicate Framework, DPF* [17,16], is presented in Sect. 3.2. Finally, Sect. 3.3 prepares the announced improved way of relating n models.

3.1 Typed Graphs

A *(directed multi-)graph* (or just *graph*) $G = (V, E, s, t)$ consists of a set V of *vertices* (or *nodes*), a set E of *edges*, and two functions $s : E \to V$, $t : E \to V$ that assign to each edge its source and target. Writing $x \in G$ means that x is a node or an edge of G. We depict graph vertices by ellipses (or circles) and edges by arrows from their source to their target vertex, cf. right part of Fig.3. A (graph) *morphism* $f : G = (V, E, s, t) \to G' = (V', E', s', t')$ is a pair (f_V, f_E) of functions $f_V : V \to V'$ and $f_E : E \to E'$ which preserve the incidence between vertices and edges, i.e. $\forall e \in E : s'(f_E(e)) = f_V(s(e))$ and $t'(f_E(e)) = f_V(t(e))$.

A graph morphism $f : G \to G'$ is an *isomorphism*, if there is a graph morphism $g : G' \to G$ such that $g \circ f = id_G$ and $f \circ g = id_{G'}$, the identical maps on G, G', resp.[4].

A graph H' is a subgraph of G', written $H' \subseteq G'$, if $H'_V \subseteq G'_V, H'_E \subseteq G'_E$ and source and target function of H' are restrictions of source and target functions of G'. In this case, we write $H' \overset{em}{\hookrightarrow} G'$ to depict the embedding morphism which takes $x \in H'$ to $x \in G'$. If $f : G \to G'$ is a graph morphism, $f^{-1}(H')$ is the graph which consists of all vertices / edges that are mapped to H' by f. In the same way as for sets we write $f|_H$ for the restriction of f to a subgraph $H \subseteq G$. This will be applied later for the subgraph $H = f^{-1}(H') \subseteq G$.

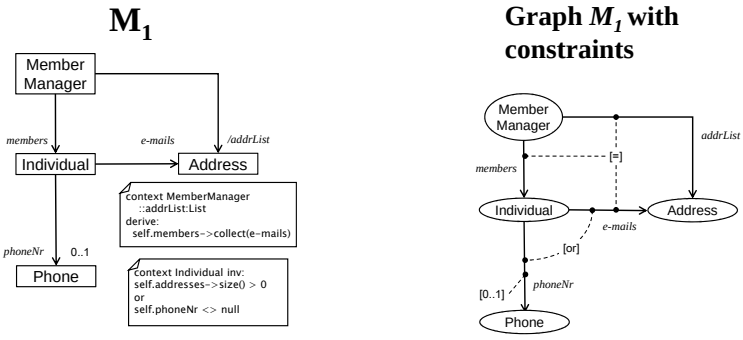


Fig. 3: UML model represented as directed graph

[4] For any two morphisms $h_1 : A \to B, h_2 : B \to C$, we write $h_2 \circ h_1$ for the *composed* morphism $((h_2)_V \circ (h_1)_V, (h_2)_E \circ (h_1)_E)$, i.e. $(h_2 \circ h_1)(a) = h_2(h_1(a))$ for all $a \in A$.

Many types of UML diagrams can be translated to graphs. For instance, in Fig.3, class diagram M_1 from Fig.1 is translated to graph M_1: Classes and associations are translated to nodes and edges. Constraints are also represented grafically, their name given in square brackets, and the constraint *scope* shown by dashed lines, i.e a set of elements over which the constraint is declared. We will elaborate on this in Sect.3.2. We remark that other types of UML diagrams can be translated to graphs, as well [7,18].

There is an obvious way of representing typed data (i.e. objects and links between them) as a graph A together with a graph morphism $\tau : A \to M$, which assigns to every data element of $x \in A$ a type $T = \tau(x) \in M$. In this case, M is called the *type graph* and we say that data A is typed over M.

In Fig.4 there is data A_2 typed over M_2 (taken from Fig.1 now with some obvious abbreviations). Graph morphism τ_2 maps all data elements to their type in M_2. We depict data elements x in the form $a{:}\,T$ (read "element a is of type T") thus declaring that $\tau(x) = T$. In Fig.4 there is one *LoginList*-object which maintains 3 login addresses which in turn identify 3 persons. Person b declares a private phone number which also is the commercial number of person c. Person d declares neither private nor commercial phone number.

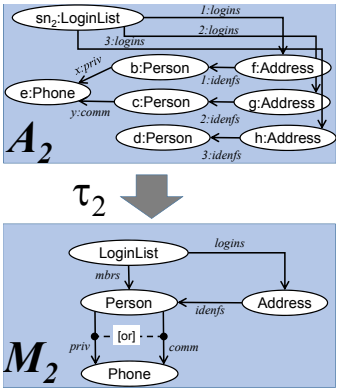


Fig. 4: Typed Graph τ_2

We will say that every graph morphism $\tau : A \to M$ is a *typed graph* w.r.t. to type graph M, since it completely represents the typed data. We distinguish between *legal* and *illegal* typed graphs: A typed graph $\tau : A \to M$ is legal, if it satisfies all constraints declared on M, otherwise illegal. E.g. τ_2 is illegal, because constraint [or] is violated for person d. The other constraints, however, are satisfied.

3.2 Constraint Declarations

A key feature of constraints used in modeling is their *diagrammatic* nature: the set of elements over which a constraint is declared is actually a diagram of some shape specific for the constraint. For example, the shape of multiplicity constraints (e.g. [0..1]) is a single arrow, while the shape of constraint [or] from Fig.3 is two arrows with common source, see Table 1. To declare a constraint in type graph M, we recognize the constraint shape in the graph and visualize it as was shown e.g. in graph M_2 in Fig.4. Formally, this recognition is a graph morphism $\delta : S^c \to M$ (called *(shape) binding*) from the shape S^c of a constraint with name c to graph M. This is shown in Fig.5, where constraint [or] is declared by binding $\delta_1 : S^{[or]} \to M_1$ (shape $S^{[or]}$ is shown in Table 1). In the same way δ_2 recognizes the

Table 1. Sample Constraints

Name c	Shape S^c
[0..1]	1 —12→ 2
[or]	1 ←01— 0 —02→ 2
[=]	0 —01→ 1 —12→ 2, 0 —02→ 2

[or]-constraint in M_2. The latter recognition also shows that shape bindings need not be injective, because we have $\delta_2(1) = Phone = \delta_2(2)$.

In Fig.5 the depicted edge assignments infer $\delta_1(1) = Phone$, $\delta_1(0) = Individual$, $\delta_1(2) = Address$. Thus, to describe the mapping behavior of graph morphisms it is enough to give the assignments on edges and isolated vertices (i.e. vertices that are not source or target of any edge) only.

The set of elements in M the shape is mapped to, is called the *range* of the binding. Any shape binding $\delta : S^c \to M$ is denoted $c@\delta$ and will be called *constraint declaration* of c at (the range of) δ (in graph M).

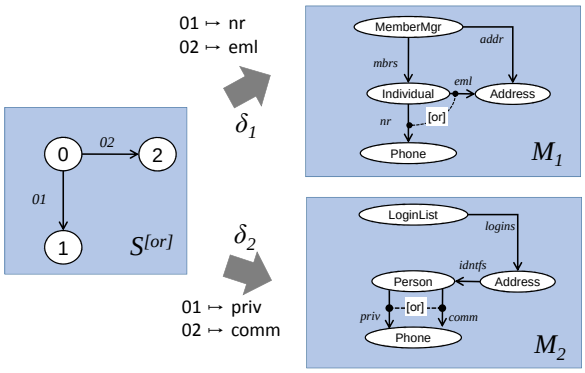


Fig. 5: (Reuse of) Shape bindings

In order to check consistency of typed graph $\tau : A \to M$ against a fixed constraint declaration $c@\delta$, we need to define c's *semantics*. This is done by programming a function VALIDATE$_c(\tau^*$: *Typed Graph*): BOOLEAN which has input $\tau^* : C \to S^c$, i.e. C is a graph typed over c's shape only. Validation shall be independent of element naming, i.e. $validate_c(\tau_1^*) = validate_c(\tau_2^*)$ whenever τ_1^* and τ_2^* are isomorphic[5].

For example, function VALIDATE$_{[or]}$ acts on graphs typed over $S^{[or]}$ (cf. Table 1): it returns *true* for a graph $\tau^* : C \to S^{[or]}$, iff each element of type 0 in C has an outgoing edge to some element of type 1 or to some element of type 2. Correspondingly, function VALIDATE$_{[=]}$ acts on graphs typed over $S^{[=]}$: it evaluates to true, if for each object x: 0 the following commutatitivity constraint holds: If X is the collection of objects reachable via 12-typed links from those objects reachable from x: 0 via 01-links, and Y is the set of objects reachable along 02-links from x: 0, then $X = Y$.

[5] Two typed graphs $\tau_1^* : B_1 \to S^c$ and $\tau_2^* : B_2 \to S^c$ are said to be isomorphic, if there is a type compatible isomorphism $f : B_1 \to B_2$, i.e. for which $\tau_2^* \circ f = \tau_1^*$.

In this way, the validation pattern of constraint c is separated from verification logic of particular declaration $c@\delta$. Therefore an important feature of the DPF framework is *formal reusability* of validation logic based on typed graphs. In this spirit, a function

$$\text{CHECK}(\tau\colon \mathit{Typed\ Graph}, c@\delta\colon \mathit{ConstraintDecl})\colon \text{BOOLEAN},$$

which returns true if and only if $\tau : A \to M$ satisfies constraint declaration $\delta : S^c \to M$, must in a first step *trace back* τ to some typed graph $\tau^* : C \to S^c$ and then validate τ^*. It turns out that this can – in the context of graphs – be carried out with the categorical *pullback operation* [3], which we will now explain. We assume each involved graph X to have vertex set and edge set V_X and E_X as well as source / target functions s_X / t_X.

Fig. 6: Pullback operation

Definition 1 (Pullback Operation). *This operation has input two graph morphisms $\delta : S \to M$ and $\tau : A \to M$ with common codomain (see left part of Fig.6). It has output*

- *Graph C with*

$$V_C := \{(v\colon T) \mid v \in V_A, T \in V_S, \delta(T) = \tau(v)\}$$

  and

$$E_C := \{(e\colon T') \mid e \in E_A, T' \in E_S, \delta(T') = \tau(e)\}$$

  with incidences determined by $s_C(e\colon T') = (s_A(e)\colon s_S(T'))$ and $t_C(e\colon T') = (t_A(e)\colon t_S(T'))$.
- *Graph morphism $\delta^*(\tau) : C \to S$ defined by $\delta^*(\tau)(x\colon T) = T$ for all $(x\colon T) \in C$,*
- *Graph morphism $\delta' : C \to A$ defined by $\delta'(x\colon T) = x$ for all $(x\colon T) \in C$,*

see right part of Fig.6. We say that $\delta^*(\tau)$ is the pullback of τ along δ *and call δ' a* traceability map.[6]

Therefore the assignment $\tau \mapsto \delta^*(\tau)$ realizes the above mentioned trace back to a graph typed over c's shape graph S and traceability map δ' enables type reconstruction (forward trace). $\delta^*(\tau)$ inherits the mapping behavior of τ, because an easy calculation shows that $\tau \circ \delta' = \delta \circ \delta^*(\tau)$, i.e. the recognized type $(\delta^*(\tau))$ in M of any element $x \in C$ is the type (τ) of its forward trace.

We demonstrate the effects of the pullback operation by formally checking that typed graph $\tau_2 : A_2 \to M_2$ (from Fig.4) does not satisfy constraint declaration $[or]@\delta_2$ ($\delta_2 : S^{[or]} \to M_2$) from Fig.5. Fig.7 shows the complete application of the pullback operation. According to the above description of function $\text{VALIDATE}_{[or]}$ one obtains $\mathit{validate}(\delta^*(\tau_2)) = \mathit{false}$, because $d\colon \mathit{Person}\colon 0$ has neither an outgoing edge to a 0-typed nor to a 1-typed vertex.

The pullback operation provides the right amount of restriction and retyping:

[6] See the explanations below why this terminology is used.

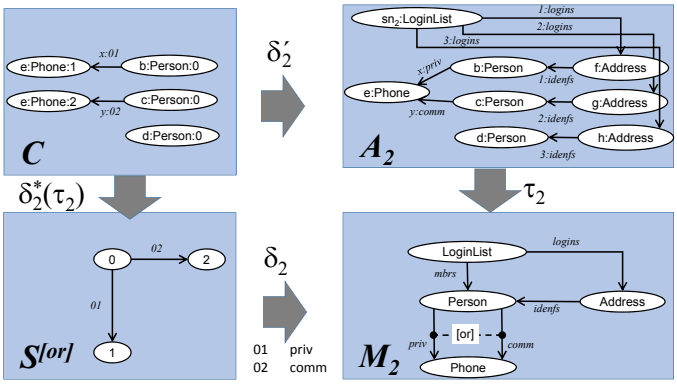


Fig. 7: How function CHECK works

- *Restriction*: In a first step it restricts precisely to those elements of A, that are typed in elements that are in the image of δ.
- *Retyping*: Then this remaining portion is retyped. Each element's new type in C is a preimage of its old type in A under δ, thus tracing back to validation pattern $[or]$.

If there is more than one preimage of type T of an element $x \in A$ (in Fig. 7 this is the case for $x = e\colon Phone$, because $\delta_2(1) = Phone = \delta_2(2)$), then – according to the definition of V_C in Def.1 – there are as many copies of x in C as there are preimages of T in S. This copying is necessary because otherwise the type of element e in C remains ambiguous and constraint satisfaction for persons b and c may fail during validation.

We also stress the fact that constraint violation of $d\colon Person\colon 0$ can be "traced forward": $\delta'_2(d\colon Person\colon 0) = d\colon Person$ enables immediate marking[7] of the very element, which violates the constraint in A_2.

We summarise these considerations by programmatically giving the design of the check function:

BOOLEAN CHECK($\tau : TypedGraph, c@\delta : ConstraintDecl$){

$$\text{RETURN } validate_c(\delta^*(\tau)); \tag{1}$$

}

We say that τ *satisfies* $c@\delta$ and write

$$\tau \models c@\delta,$$

if $check(\tau, c@\delta)$=*true*. τ is called a *legal* typed graph over M, if it satisfies all constraints declared in M, otherwise *illegal*.

[7] e.g. by a grafical tool

Since validation is stable under renaming, each typed graph isomorphic to $\delta^*(\tau)$ can also be taken as pullback. We could e.g. have named the C-vertices $e{:}\,1, e{:}\,2, b{:}\,0, c{:}\,0, d{:}\,0$ in Fig.7. Forward tracing as described above is still possible with the help of δ'. Renaming also enables C to be a proper subgraph of A, if $S \subseteq M$ and $\delta : S \hookrightarrow M$ is the respective embedding, because it is easy to see that δ' is injective in that case. Thus

$$S \subseteq M \quad \Rightarrow \quad (C = \tau^{-1}(S) \text{ and } \delta^*(\tau) = \tau|_C). \tag{2}$$

3.3 Partial Graph Morphisms

In Sect.4, *partial graph morphisms*

$$G \xrightarrow{f} G'$$

will be used (the "partial" arrow tip symbolizes partiality of the morphism): Partial morphisms may only be defined on a subgraph $\mathrm{dom}(f) \subseteq G$. Clearly, any graph morphism $h : G \to G'$ as defined in Sect.3.1 is a partial morphism with $\mathrm{dom}(h) = G$.

Partial morphisms $G \xrightarrow{f} G'$ and $G' \xrightarrow{g} G''$ can be composed by letting

$$f' = f|_{f^{-1}(\mathrm{dom}(g))} : f^{-1}(\mathrm{dom}(g)) \to \mathrm{dom}(g)$$

be the restriction of total $f : \mathrm{dom}(f) \to G'$ to the preimage of $\mathrm{dom}(g) \subseteq G'$ under f. Then the composition $g \circ f$ of partial maps f and g is defined to be the map $g \circ f'$, which is defined on $f^{-1}(\mathrm{dom}(g)) \subseteq \mathrm{dom}(f) \subseteq G$ and undefined otherwise.

Note that a partial morphism $G \xrightarrow{f} G'$ gives rise to an embedding graph morphism $\mathrm{dom}(f) \xhookrightarrow{-f} G$ where f is total on $\mathrm{dom}(f)$.

4 Multi-Models and Multi-Instances

Modeling a complex system usually results in a *multi-model*, i.e., a set $M_1, \ldots, M_n$ of model components for some $n \geq 1$. In Sect.2, for instance, there are the 4 models $M_1, \ldots, M_4$, cf. Fig.1 and Fig.2 which have to be considered as a collective artefact in order to investigate the mentioned inter-model constraint therein. In this case, graphs typed over $M_1, \ldots, M_4$ have to be investigated, i.e. heterogeneously typed data $\tau_j : A_j \to M_j$ must be analysed altogether.

In the example, typed data $A_1, \ldots, A_4$ are artefacts on MOF[8] level 0 whereas $M_1, \ldots, M_4$ are on level 1. Since – in the graph-based view – it makes no difference whether $\tau_j : A_j \to M_j$ is data typed over models, or whether $\tau_j : M_j \to MM_j$ are models that conform to (are heterogeneously typed in) metamodels $MM_1, \ldots, MM_n$, the forthcoming theory can be applied to typed data as well as for typed models. In the latter case, the M_j may be class diagrams, sequence diagrams, state-charts, or activity diagrams on level 1, MM_j being the corresponding metamodels on level 2.

[8] Meta-Object-Facility, cf. http://www.omg.org/mof/

Because scenarios with typed data ($n = 0$) better demonstrate the advantages of the forthcoming inter-model constraint checking algorithm, we adhere to the scenario of Sect.2, keeping in mind that the algorithm can likewise be applied to any MOF level as long as type conformance can be coded by typed graphs.

4.1 Model Matching

In the example of Sect.2, $M_1, \ldots, M_4$ collectively represent a single system and any formal treatment has to consider *overlaps*, i.e. the definitions of common terminology in different models. E.g., the concept "Address" occurs in 3 of the above-mentioned models. Names of common concepts, however, may differ: *Individual* in M_1 and *Person* are differently named, yet both domains may speak of the same concept.

It is our goal to formally *match* common concepts by means of graph morphisms: It is well-known, that in the case of two models M_1 and M_2 this can be achieved with a span of two graph morphisms $M_1 \xleftarrow{m_1} M_0 \xrightarrow{m_2} M_2$, where auxiliary graph M_0 contains all common concepts. Then $x_1 \in M_1$ and $x_2 \in M_2$ are declared to be the same, if there is $x_0 \in M_0$ such that $m_1(x_0) = x_1$ and $m_2(x_0) = x_2$.[11]

It turns out that in the case of an arbitrary number of model graphs $M_1, \ldots, M_n$ these spans between the models are no longer helpful. This can already be seen in the case $n = 3$. It can happen that we want to specify sameness of two elements $x_1 \in M_1$ and $x_2 \in M_2$ on the one hand, and of three elements y_1, y_2, y_3 in M_1, M_2, M_3 on the other hand. Then for the first specification we would need a binary span, and for the second specification we would need a ternary span, each of them with its own auxiliary structure. In general, each new occuring number k of same elements $x_1, \ldots, x_k$ requires a new span of arity k. This may lead to hardly manageable structures.

Instead of using total graph morphims for matching, we can, however, use *partial graph morphisms* as introduced in Sect.3.3. Suppose we want to declare sameness of $x_{i_1} \in M_{i_1}, x_{i_2} \in M_{i_2}, \ldots, x_{i_k} \in M_{i_k}$ for some $k \in \{2, \ldots, n\}$, then in a graph M_0 there must be an element x_0 which represents this common term or concept. Furthermore partial assignments

$$M_0 \xrightarrow{m_j} M_j$$

have to be defined for all $j \in \{1, \ldots, n\}$ such that $m_{i_j}(x) = x_{i_j}$ for $2 \leq j \leq k$ and m_j is undefined on x for all $j \notin \{i_1, \ldots, i_k\}$. Since sameness declaration of several edges in models $M_1, \ldots, M_n$ infers sameness of their respective source and target vertices, this guarantees that M_0 becomes a graph and m_j become partial graph morphisms. We call M_0 the *glueing graph* or *overlap* of models $M_1, \ldots, M_n$. Elements of M_0 are called *match* (or sameness) *witnesses*.

By means of the running example of Sect.2 we illustrate this specification of commonalities: Suppose, one claims that the following concepts are declared to coincide:

1. "Phone" in M_1 and M_2
2. "Individual" and "Person" although they are named differently.
3. Since sn_1 is a network of private persons, it is decided to match "phoneNr" and "private" in M_1 and M_2, too.
4. Obviously "Address" in M_1, M_2, and M_3.

5. "MemberMgr", "LoginList", and "Social Network" in M_1, M_2, and M_4 (see the remarks after the constraint declaration in Sect.2).

Note that declaration 3 infers declarations 1 and 2 because a matched edge yields matching of its source and target, too. Fig.8 shows these four matching declarations[9] by means of glueing graph M_0 (consisting of 4 vertices and one edge) and partial graph morphisms $m_1, \ldots, m_4$. Common colors indicate declared matches. $m_1, \ldots, m_4$ map according to the coloring. Whereas m_1 and m_2 are total morphisms, m_3 and m_4 are proper partial, because they are e.g. undefined on edge n/p. The 5 elements of M_0 witness sameness declarations as follows: Vertex Ph represents declaration 1, I/P reflects declaration 2, edge n/p specifies 3, vertex A witnesses 4, and $M/L/S$ represents 5.

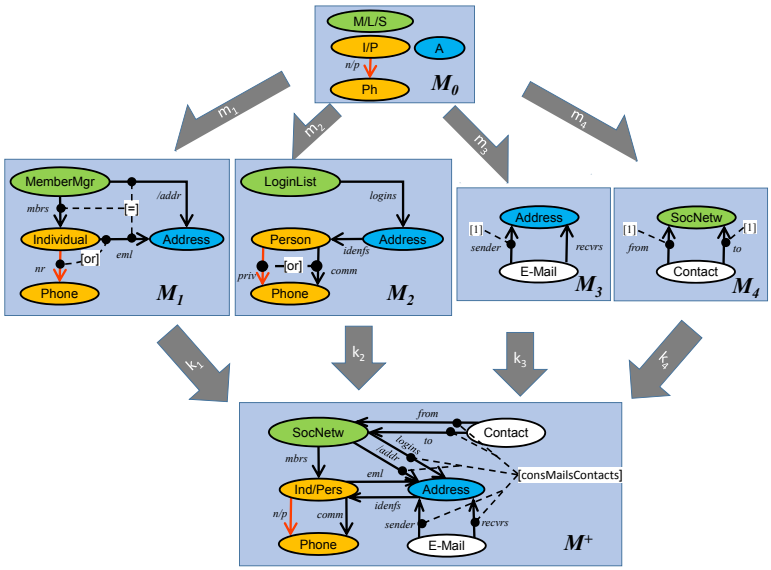


Fig. 8: Matching of concepts in models $M_1, \ldots, M_4$ and resulting colimit graph M^+

Definition 2 (Multi-Model). *We call this configuration of models $M_1, \ldots, M_n$, glueing graph M_0 and partial morphisms $M_0 \xrightarrow{m_j} M_j$ $(1 \leq j \leq n)$ a* multi-model $\mathcal{M}$. *Because it is completely determined by the involved partial morphisms (their domain and codomain specifying all participating graphs) we write*

$$\mathcal{M} = (M_0 \xrightarrow{m_j} M_j)_{1 \leq j \leq n}$$

(we use also the short notation $\mathcal{M} = (m_j)_{1 \leq j \leq n}$, if domains and codomains are known).

This structure is also used in [9], sect. 9.2., where M_0 is called *glue*, the partial morphisms are called *connections*, together they are called a (well-formed) *configuration*.

[9] Due to readability, we depicted only some of the constraints of $M_1, \ldots, M_4$.

4.2 Merging

As announced in the introduction, an important operation on arrangements of graphs and graph morphisms as e.g. shown in the upper half of Fig.8 (graphs $M_0, M_1, \ldots, M_n$ and partial morphisms $m_1, \ldots, m_n$ for $n = 4$) is the construction of the merge of all M_j modulo their overlaps, such that inter-model constraints as in Sect.2 can formally be declared and checked.

Definition 3 (Merge for Multimodels). *Let multi-model* $\mathcal{M} = (M_0 \xrightarrow{m_j} M_j)_{1 \le j \le n}$ *be given as above. We say that* $x_0 \in M_0$ *and* $x \in M_j$ *are related, written*

$$x_0 \sim x$$

if m_j *is defined on* x_0 *and* $m_j(x_0) = x$. *Let* $\equiv$ *be the reflexive and transitive closure of binary relation* $\sim$, *then we say that*

$$M^+ := \left(\biguplus_{0 \le i \le n} M_i \right) \Big/ \equiv$$

is the merge *of* $\mathcal{M}$ [10] *and write* $M^+ = merge(\mathcal{M})$.

For each $i \in \{0, 1, \ldots, n\}$ *there is the graph morphism* $k_i : M_i \to M^+$, *which maps each* x *to its equivalence class* $[x]_\equiv$. *Each* k_i *is called the* reccognition *of* M_i *in the merge of multimodel* $\mathcal{M}$. *By the definition of* $\sim$, *we obtain*

$$k_0 = k_j \circ m_j \tag{3}$$

for all $j \in \{1, \ldots, n\}$ *on the domain of definition of* m_j.

This definition is again in the spirit of [9], where multi-model semantics is explicitely defined by the merge of the resulting diagram. An example of the merge operation for multi-models is shown in Fig.8: We obtain for instance

$$Individual \equiv Person$$

because $m_1(I/P) = Individual$ and $m_2(I/P) = Person$. Therefore $k_1(Individual) = k_2(Person)$ and this element is called $Ind/Pers$ in M^+. In the same way $MemberMgr \equiv LoginList \equiv SocNetw$, $nr \equiv priv$ and sameness of concepts $Address$. Vertex $Contact \in M_4$ is not reached via any m_j and thus not identified with any other concept. The same is true for *E-mail* and also for all edges except *nr* and *priv*.

We note that it is necessary to define M^+ by constructing a corresponding quotient w.r.t. equivalence relation $\equiv$, because it is not possible to define M^+ with the help of the set-theoretical union: The latter would yield two different vertices *Individual* and *Person* in M^+ while the former enables merging this concept to a single one $Ind/Pers$.

[10] $\biguplus_{0 \le i \le n} M_i$ denotes the disjoint union of the model components (including glueing graph M_0), i.e. their union still disregarding sameness.

4.3 Data Matching

In addition to the matching of type graphs, it is necessary to do the same for data graphs and their typings.

Definition 4 (Multi-Instance). *Let* $\mathcal{M} = (\ M_0 \xrightarrow{m_j} M_j\ )_{1\le j\le n}$ *be a multimodel.*

- *An* incomplete *(or* unrelated*)* Multi-Instance *over* $\mathcal{M}$ *is a collection*

$$\mathcal{T} = (\tau_j : A_j \to M_j)_{1\le j\le n}$$

  of typed graphs.
- *A* complete(ly related) *multi-instance over* $\mathcal{M}$ *is a collection*

$$(\tau_i : A_i \to M_i)_{0\le i\le n}$$

  together with partial graph morphisms

$$A_0 \xrightarrow{a_j} A_j$$

  ($1 \le j \le n$*) such that the type compatibility conditions*

$$m_j \circ \tau_0 = \tau_j \circ a_j \tag{4}$$

  hold for all $1 \le j \le n$*. A complete multi-instance over* $\mathcal{M}$ *is denoted*

$$\overline{\mathcal{T}} = ((\tau_i : A_i \to M_i)_{0\le i\le n}, (\ A_0 \xrightarrow{a_j} A_j\ )_{1\le j\le n})$$

  with the corresponding short notation $\overline{\mathcal{T}} = ((\tau_i)_{0\le i\le n}, (a_j)_{1\le j\le n})$ *if all participating graphs are known.*
- *Matching of an incomplete multi-instance* $\mathcal{T} = (\tau_j : A_j \to M_j)_{1\le j\le n}$ *is the process of finding a complete multi-instance* $\overline{\mathcal{T}} = ((\tau_i : A_i \to M_i)_{0\le i\le n}, (a_j : A_0 \rightharpoonup A_j))$*.*

In this context, graph morphisms a_j play the same role as morphisms m_j: While the m_j's control the type matching, a_j declare sameness of elements in the data graphs (data matching). Thus matching is an enhancement of $\tau_1, \ldots, \tau_n$ with ...

1. ... morphism $\tau_0 : A_0 \to M_0$ which specifies typing of sameness witnesses in glueing graph A_0 and
2. ... overlap specification of data $A_1, \ldots, A_n$ with the help of glueing graph A_0 and partial morphisms $A_0 \xrightarrow{a_j} A_j$ $(1 \le j \le n)$ in the same way as for multi-models (cf. Def.2) such that this matching is compatible with type matching.

In the same way as in Def.3 we can now establish the following operation:

Definition 5 (Merge Operation for Data). *Let complete multi-instance* $\overline{\mathcal{T}} = ((\tau_i : A_i \to M_i)_{0\le i\le n}, (A_0 \xrightarrow{a_j} A_j)_{1\le j\le n})$ *be given. Then in the same way as in Def.3 there are relations* $\sim$ *and* $\equiv$ *now based on morphisms* a_j *which make up the* data merge A^+ *of the data graphs* $A_0, A_1, \ldots, A_n$ *modulo their matched overlap* $(a_j)_{1\le j\le n}$ *together with recognition morphisms* $l_i : A_i \to A^+$ $(0 \le i \le n)$*. It can be shown that this yields a unique typing* $\tau^+ : A^+ \to M^+$*, which is compatible with recognitions, i.e.* $\tau^+ \circ l_i = k_i \circ \tau_i$ *for all* $i \in \{0, \ldots, n\}$ *[1]. We write*

$$\tau^+ = Merge(\overline{\mathcal{T}})$$

Note the difference in notation: *merge* depicts the respective operation on multi-models, *Merge* is the operation on multi-instances.

4.4 Pulling Back Multi-Instances

We fix a graph morphism $\delta : S \to M$, then the pullback operation along δ of Def.1 can be applied to arbitrary graph morphisms $k : H \to M$: In

$$\begin{array}{ccc} G & \xrightarrow{\delta'} & H \\ {\scriptstyle \delta^*(k)}\downarrow & (PB) & \downarrow{\scriptstyle k} \\ S & \xrightarrow{\delta} & M \end{array}$$

G and $\delta^*(k)$ are computed in the same way as C and $\delta^*(\tau)$ in Def.1. We now describe an important extension of this operation: Let $k_1 : H_1 \to M$ and $k_2 : H_2 \to M$ be two graph morphisms with common codomain M and $H_1 \xrightarrow{f} H_2$ be a compatible partial graph morphism, i.e. $k_2 \circ f = k_1$, see the left part of Fig.9. Having computed the respective pullbacks $\delta^*(k_1) : G_1 \to S$ and $\delta^*(k_2) : G_2 \to S$ of k_1 and k_2 along δ, we can define the mapping $G_1 \xrightarrow{\delta^{**}(f)} G_2$ by

$$\delta^{**}(f)(x{:}\,T) := (f(x){:}\,T),$$

for all $x \in \mathrm{dom}(f)$, and undefined otherwise, cf. right part of Fig.9. It is easy to see that δ^{**} is a partial graph morphism which is compatible with typing and traceability maps.

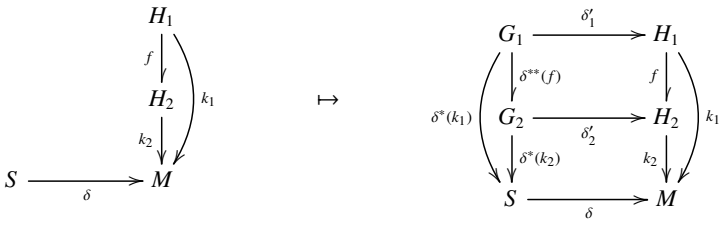


Fig. 9: Pullback operation applied to type compatible graph morphisms

We apply these results to inter-model constraint's shape binding $\delta : S^c \to M^+$: There are the morphisms $k_0 : M_0 \to M^+$ and $k_j : M_j \to M^+$ for $j \in \{1, \ldots, n\}$ for which operator δ^* can be applied. For the partial morphisms $M_0 \xrightarrow{m_j} M_j$ equation (**??**) holds, such that we can compute $\delta^{**}(m_j)$. It can also be shown that we can compute $\delta^{**}(\tau_j)$.

A generalization of the restriction and retyping procedures of Sect.3.2 for graph τ typed over a single model M to multi-instance $\mathcal{T}$ typed over a multi-model $\mathcal{M}$ is

Proposition 1. *Let $\mathcal{M} = (m_i)_{0\le i\le n}$ be a multi-model and $(k_i : M_i \to M^+)_{0\le i\le n}$ be the corresponding recognitions of M_i in the merge of $\mathcal{M}$.*

1. *A graph morphism $\delta : X \to M^+$ gives rise to an operator $\delta^\star$ which transforms any incomplete multi-instance $\mathcal{T} = ((\tau_j : A_j \to M_j)_{1\le j\le n}$ over $\mathcal{M}$ to an incomplete multi-instance $\mathcal{T}' = (\delta^{**}(\tau_j) : B_j \to X_j)_{1\le j\le n}$ over $\mathcal{X} = (\delta^{**}(m_i))_{0\le i\le n}$. We write*

$$\mathcal{T}' = \delta^\star(\mathcal{T})$$

   In the same way any complete multi-instance $\overline{\mathcal{T}} = ((\tau_i : A_i \to M_i)_{0\le i\le n}, (a_j : A_0 \rightharpoonup A_j))$ over $\mathcal{M}$ is transformed to complete multi-instance $\overline{\mathcal{T}'} = (\delta^{**}(\tau_i) : B_i \to X_i)_{0\le i\le n}, (\delta^{**}(a_j) : B_0 \rightharpoonup B_j)$ over $\mathcal{X}$:

$$\overline{\mathcal{T}'} = \delta^\star(\overline{\mathcal{T}})$$

2. *In both cases the collection $(k_i)_{0\le i\le n}$ of $\mathcal{M}$ is transformed to $(\delta^*(k_i) : X_i \to X)_{0\le i\le n}$.*
3. *If $\delta : X \hookrightarrow M^+$ is an embedding, then, in both cases, the traceability maps $X_i \to M_i$ and $B_i \to A_i$ can be taken to be embeddings, i.e. for all i: $X_i \subseteq M_i$ and $B_i \subseteq A_i$.*

A complete formal reasoning for this result can be found in the Appendix. The transformation is shown in Fig.10 for the case of complete multi-instance (traceability maps are omitted).

5 Inter-Model Constraint Checking

In this chapter, an algorithm is introduced which efficiently verifies consistency of a collection of typed graphs, i.e. an incomplete multi-instance

$$\mathcal{T} := (\tau_j : A_j \to M_j)_{1\le j\le n}$$

over multi-model $\mathcal{M} = (\ M_0 \xrightarrow{m_j} M_j\ )_{1\le j\le n}$.

Definition 6 (Inter-Model Constraint Declaration). *Let $M^+ := merge(M)$ and c be a constraint with shape graph S^c (cf. Sect.3.2). A shape binding*

$$\delta : S^c \to M^+$$

is called an inter-model constraint declaration *on multi-model $\mathcal{M}$ and is written $c@\delta$.*

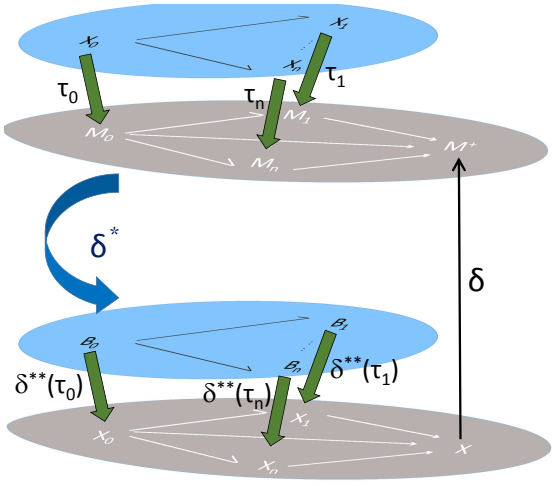


Fig. 10: The Operator $\delta^\star$

Thus it is the goal to decide whether some multi-instance $\mathcal{T}$ satisfies inter-model constraint declaration $c@\delta$, which we write

$$\mathcal{T} \models c@\delta.$$

We already gave an example, namely the requirement of Sect.2, which claims consistency of recorded e-mails according to model M_3 and stored contacts due to M_4 for social networks sn_1 and sn_2, whose domains are based on models M_1 and M_2. In the sequel *consMailsContacts* denotes this constraint declaration, which we repeat here:

$$\begin{aligned} &|\{c \in Contact \mid c.from = sn_1 \wedge c.to = sn_2\}| \\ = &|\{e \in E-Mail \mid e.sender \in sn_1.addrList \wedge e.receivers \cap sn_2.logins \neq \emptyset\}| \end{aligned}$$

We will compare two implementations (Sect.5.1 and 5.2, resp.) both of which have the following input and output:

- **Input**:Inter-model constraint declaration $c@\delta$ on $\mathcal{M}$, incomplete multi-instance $\mathcal{T}$.
- **Output**: The boolean value which indicates whether $\mathcal{T} \models c@\delta$.

In both cases, we assume $M^+ = merge(\mathcal{M})$ to be already computed. Note that this computation must be carried out only once prior to the invocations of the checking functions for different inputs $(c@\delta, \mathcal{T})$. [11]

[11] Moreover, complexity for this computation is low, because $merge(\mathcal{M})$ can be computed algorithmically by partitioning the disjoint union of the models due to binary relation $\equiv$, see Def.3. This requires little effort, since we deal with a manageable number n of small models (in our example $n = 4$). Matching effort (e.g. to decide whether to declare sameness of concepts "Individual" and "Person") is acceptable.

5.1 Match→Merge→Localize (MML)

The following algorithm is the original proposal of [19] to check $\mathcal{T} \models c@\delta$:

1. *Match* incomplete multi-instance $\mathcal{T}$ (cf. Def.4), which gives
$$\overline{\mathcal{T}} = ((\tau_i : A_i \to M_i)_{0\le i\le n}, (\ A_0 \xrightarrow{a_j} A_j\ )_{1\le j\le n}).$$
2. Compute $\tau^+ = Merge(\overline{\mathcal{T}})$ as described in Def.5.
3. The return value of $check(\tau^+, c@\delta)$ provides the result of the algorithm.

Because of the order, in which the steps are carried out, and since in step 3 checking is performed on restricted (hence *localized*) data (cf. Def.1), we call this method the "Match→Merge→Localize"-approach MML. It is thus reduced to a check of a single but huge instance τ^+:
$$\mathcal{T} \models c@\delta : \Longleftrightarrow\ \tau^+ \models c@\delta$$
In [11] it is further analysed and assessed. It turns out that there are two major drawbacks:

1. The *total* collection of data structure has to be traversed in order to perform matching in step 1. Matching decisions can not always be made automatically, yet matching is necessary also for data that is not specific to the given constraint declaration.
2. In step 2, one has to deal with the entire union of data (usually a huge structure) independent of whether there is only a small portion being affected by the constraint.

These disadvantages can easily be demonstrated in our running example: Although constraint declaration $c@\delta = consMailsContacts$ does not affect individuals / persons in M^+ (see Fig.8), this comprehensive approach demands to deal with *all* personal data during computation of τ^+.

More seriously, one has to match individuals (typed in M_1) with persons (typed in M_2). Experience, however, shows that data of the same person stored in two different databases often differs due to typing errors, undocumented properties, or inconsistent updates. Eliminating these contradictions in large databases (with probably thousands of doubly captured data records) is far beyond hopeless.

These considerations show that an implementation of inter-model constraint checking along its definition is not feasible.

5.2 Localize→Match→Merge (LMM)

This section is the main contribution of the paper, in that it proposes a more efficient algorithm to check $\mathcal{T} \models c@\delta$. We fix these two inputs and - for the sake of simplicity - write S instead of S^c for the shape graph of c.

Recall the discussion about pullback characteristics *restriction* and *retyping* after Fig.7. These two characteristics can be formalized by decomposing δ into two mappings:
$$\delta = \delta_r \circ \delta_t.$$

$\delta_r : R \hookrightarrow M^+$ specifies how the range R of δ is embedded in M^+, i.e. it specifies how to *restrict* from M^+ to this range. The *retyping* part $\delta_t : S \to R$ specifies how to trace back the scoped types of the range R in M^+ to the types of c's arity shape.[12] In Fig.8, R would be the subgraph of M^+ consisting only of elements affected by the constraint, namely vertices *SocNetw*, *Contact*, *Address*, *E-Mail* together with the edges between them.

We now describe the improved checking algorithm. When we use the term "local", we always mean that types or data are considered on constraint affected parts only:

1. (Model and Data *Localization*) Compute incomplete multi-instance

$$\mathcal{T}' := \delta_r^\star(\mathcal{T}) = (\tau'_j := \delta_r^{**}(\tau_j) : B_j \to R_j)_{1\le j\le n}$$

   over multi-model $\mathcal{R} = (\ R_0 \xrightarrow{\delta^{**}(m_j)} R_j\ )_{1\le j\le n}$ by applying operator $\delta_r^\star$ from Prop.1. One obtains $R_i \subseteq M_i$ $(0 \le i \le n)$ and $B_j \subseteq A_j$ $(1 \le j \le n)$, because δ_r is an embedding (see Prop.1, 3).
2. (Local Data *Matching*) Match incomplete multi-instance $\mathcal{T}'$, i.e. declare sameness of objects in all B_j by extending to a complete multi-instance

$$\overline{\mathcal{T}}' := (\tau'_i : B_i \to R_i)_{0\le i\le n}, (\ B_0 \xrightarrow{b_j} B_j\ )_{1\le j\le n})$$

   over multimodel $\mathcal{R}$.
   This local matching cannot contradict a hypothetical global match, i.e. it must be carried out such that all sameness witnesses for the local match are equally-typed witnesses in a hypothetical global match $(\tau_i : A_i \to M_i)_{0\le i\le n}$ with partial morphisms $(\ A_0 \xrightarrow{a_j} A_j\ )_{1\le j\le n}$. Formally:

$$\tau'_0 = \tau_0|_{\tau_0^{-1}(R_0)} \text{ and } \mathrm{dom}(b_j) = a_j^{-1}(B_j) \text{ with } b_j = a_j \text{ on } \mathrm{dom}(b_j). \tag{5}$$

3. (Local Retyping) Let

$$\overline{\mathcal{T}}'' := \delta_t^\star(\overline{\mathcal{T}}') = (\tau''_i : C_i \to S_i)_{0\le i\le n}, (\ C_0 \xrightarrow{c_j} C_j\ )_{1\le j\le n})$$

   be complete multi-instance over multi-model $\mathcal{S} = (\ S_0 \xrightarrow{s_j} S_j\ )_{1\le j\le n}$ due to Prop.1 for the transformation of complete multi-instances.
4. Compute $\tau'' = Merge(\overline{\mathcal{T}}'') : C^+ \to S$ according to Def.5.
5. Apply $validate_c(\tau'')$.

Because of *early localization*, this algorithm is called the "Localize→Match→Merge"-approach (LMM), in which step 2 is the only possibly manual or at least semi-automatic step. Before we illustrate the algorithm along the example, we state the main theorem of the paper:

[12] $\delta_r \circ \delta_t$ is obtained by the well-known epi-mono-factorization applied to vertex- and edge-mappings, resp.

Theorem 1 (Correctness). *Let $\mathcal{T}$ be an incomplete multi-instance over multi-model $\mathcal{M}$ and $c@\delta$ be an inter-model constraint declaration on $\mathcal{M}$, then*

$$MML(\mathcal{T}, c@\delta) = LMM(\mathcal{T}, c@\delta).$$

A formal proof of this theorem is given in the Appendix. □

Let's illustrate the algorithm along the example of Sect.2:

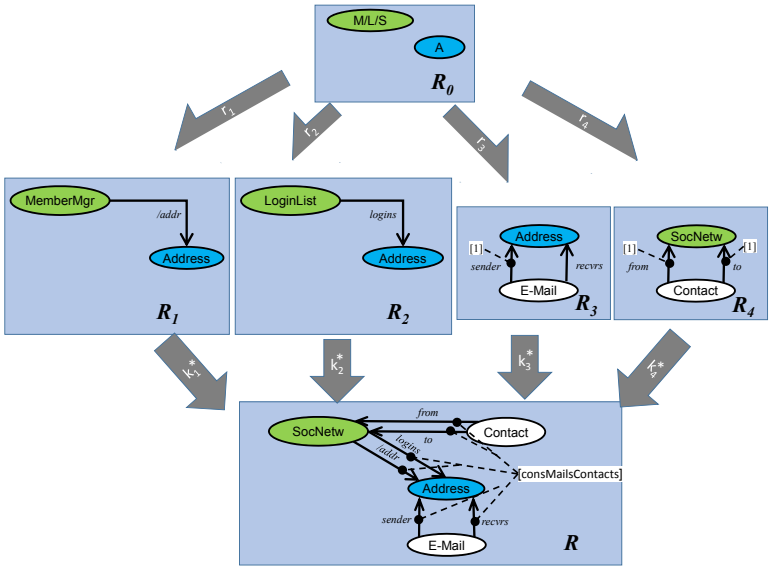


Fig. 11: Step 1: Early Model Component Restriction

Step 1: Fig.11 shows model localization: R_i are the intersections of M_i with the range of δ. Fig.12 shows data localization exemplified for data typed over M_2: Persons and phone numbers are eliminated since they do not participate in constraint checking.

Step 2: Suppose $B_1, \ldots, B_4$ contain the data (vertices) of Tab.2. In order to maintain adresses, there is one *MemberMgr*-vertex sn_1 in B_1 and one *LoginList*-object sn_2 in B_2. Assume B_3 contains only one e-mail with sender $b_{3,1}$ and receiver $b_{3,2}$. Moreover, assume B_4 contains one contact from sn_1 and sn_2. For B_2 see also Fig.12.

Obviously there are some common e-mail-addresses, which have to be matched. In order to do this, we define the set (discrete graph)

$$B_0 = \{ang\colon A, bar\colon A, sn1\colon M/L/S, sn2\colon M/L/S\}$$

of sameness witnesses, where elements are typed over R_0 (cf. Fig.11). Moreover, there are the partial morphisms $B_0 \xrightarrow{b_j} B_j$ which map according to the colors, i.e. they specify $b_{1,2} = b_{2,1} = b_{3,1}$ as well as $b_{1,1} = b_{2,2}$ and also declare sameness of entities with identifier sn_1 as well as objects with identifier sn_2.

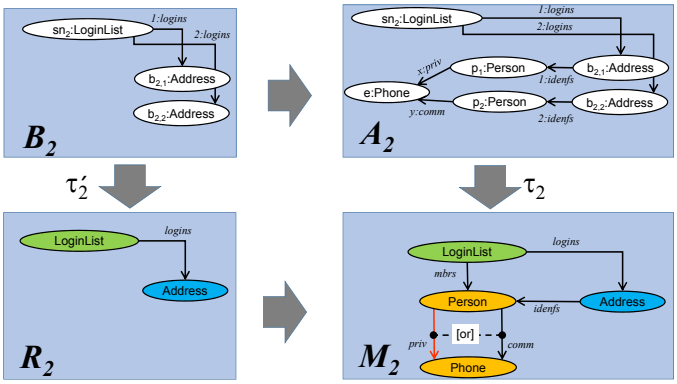


Fig. 12: Step 1: Early Data Restriction for τ_2

Table 2. Data Matching

B_1	B_2	B_3	B_4
sn_1	sn_2	:E-mail	:Contact
$b_{1,1}$ =barack@us.gov	$b_{2,1}$ =:angela@bt.de	$b_{3,1}$ =:angela@bt.de(sender)	sn_1(from)
$b_{1,2}$ =:angela@bt.de	$b_{2,2}$ =:barack@us.gov	$b_{3,2}$ =:justin@ottawa.ca(receiver)	sn_2(to)

This matching can be performed semi-automatically: E-mail addresses can be matched automatically by string equality. Objects sn_1 and sn_2 might be matched manually or an automatic proposal has to be confirmed by a user. Yet there is only small manual effort. Note that step 2 is the most expensive one, since data matching is possibly concerned with bigger data records than in this toy example. But early localization enables significant reduction of the data space, in which matching takes place: *In our case, it is not necessary to match all involved individuals with all persons, which was already considered nearly impossible (see the concluding comments in Sect.5.1).*

Step 3 performs retyping to types of S, the shape graph of inter-model constraint c, which underlies the declaration *consMailsContacts*. This shape graph is shown in the bottom of Fig.13. The color displays of the vertices correspond to the colors of its binding into the graph R in Fig.11. Retyping has the following effects: E-mail-addresses $b_{1,1},\ldots,b_{3,2}$ are retyped from type *Address* to type 1 and social network identifiers are retyped to type 0. Moreover, E-mail-objects are now of type E, Contacts are of type C. Likewise edge retypings $from \mapsto f$, $to \mapsto t$, $addr \mapsto a$, $logins \mapsto l$, $sender \mapsto s$, $receiver \mapsto r$.

Step 4 computes the merge of this data which is shown in Fig.13. The upper graph depicts the facts that there is one contact $(:C)$ from $(:f)$ social network 1 to $(:t)$ social network 2. sn_1 maintains addresses *ang* and *bar* and sn_2 has the same addresses as logins (cf. Tab.2). Finally, there is one e-mail with sender $(:s)$ being the address *ang* and receiver $(:r)$ being the address *just*.

Step 5 validates this instance to $false$, because the contact from sn_1 to sn_2 is not reflected by an e-mail that is received by a member of sn_2 ($just$ is not part of any social network). Immediate marking in input $\mathcal{T}$ is easily possible as follows: The recognition maps of $\overline{\mathcal{T}}''$ enable trace back to instances typed over $S_1,\ldots,S_4$. Subsequent trace forward along traceability maps of pullbacks detect the error in one or several of $A_1,\ldots,A_4$ and can mark it for the user.

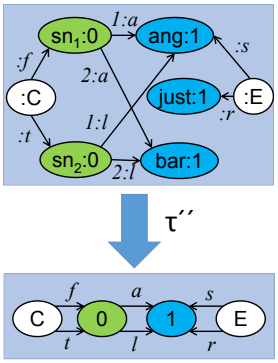


Fig. 13: Final merge of affected data typed over shape graph for $consMailsContacts$

Recall that the key to a successful methodology was *early localization*, i.e. basically we changed the sequence of the application of the operations *match, merge* and *localize* (Defs. 4, 5, 1): Early localization significantly reduces the data space which is subject for match searches. Finally, Theorem 1 guarantees that the result of MML is not falsified by LMM.

6 Conclusion

The most important direction for future research are:

- We plan to evaluate the algorithm in the tooling framework developed at Bergen University College [12,17]. Our idea is to enhance the DPF editors to make them inter-metamodel aware.
- Take into consideration view definitions (on metamodels) and view execution (on models)[4]. The challenge will be to find appropriate generalization and extensions of the mathematical machinery.
- Extend the scope of underlying grafical structures: From simple directed graphs to more general structures have, e.g. attributed graphs [8] or general presheaf topoi in order to inline constraint declarations [14].
- The next natural step is to extend multi-model consistency checking to multi-model-repairing/completion, e.g.[15]. Model relations in the context of update procedures should consist of a generalization of the (binary) delta-lens framework [6].
- After the construction of the colimit of sketches, all intra-model constraint declarations are dealt with. It is now possible to add additional (inter-model) constraint declarations. Categorically this means that we enlarge and thus *leave* the original sketch-colimit: If merged sketches with *additional* inter constraints are used, they are no longer a colimit. Must the merge of models also take this deviation into account? Here is a point where to think about semantics in the context of metamodels that come as functors $B * \alpha \rightarrow Set$.

– The discrete diagram $M : \{1, \ldots, n\} \to \mathbb{G}$ should be replaced by an arbitrary diagram, because there might already be natural relations and dependencies between the metamodels, e.g. an OR-dependency between ER- and object models (table = class, foreign key = association) or different versions of metamodels, M_2 being an evolvement of M_1 hence a span $M_1 \leftarrow\!\!\rightarrow M_2$.

References

1. Arbib, M., Manes, E.: The Categorical Imperative. Academic Press New York San Francisco London (1975)
2. Barr, M., Wells, C.: Category Theory for Computing Sciences. Prentice Hall (1990)
3. Diskin, Z., Wolter, U.: A diagrammatic logic for object-oriented visual modeling. Electr. Notes Theor. Comput. Sci. 203(6), 19–41 (2008)
4. Diskin, Z., Maibaum, T., Czarnecki, K.: Intermodeling, queries, and kleisli categories. In: Fundamental Approaches to Software Engineering, pp. 163–177. Springer (2012)
5. Diskin, Z., Wolter, U.: A diagrammatic logic for object-oriented visual modeling. In: Proceedings of the Second Workshop on Applied and Computational Category Theory (ACCAT 2007). pp. 19–41. ENTCS (2007)
6. Diskin, Z., Xiong, Y., Czarnecki, K.: From state- to delta-based bidirectional model transformations: the asymmetric case. Journal of Object Technology 10, 6: 1–25 (2011), http://dx.doi.org/10.5381/jot.2011.10.1.a6
7. Diskin, Z., Xiong, Y., Czarnecki, K.: Specifying overlaps of heterogeneous models for global consistency checking. In: Models, LNCS, vol. 6627, pp. 165–179. Springer (2011)
8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformations. Springer (2006)
9. Fiadeiro, J.L.: Categories for Software Engineering. Springer (2005)
10. Goldblatt, R.: Topoi: The Categorial Analysis of Logic. Dover Publications (1984)
11. König, H., Diskin, Z.: Advanced local checking of global consistency in heterogeneous multimodeling. In: Modelling Foundations and Applications - 12th European Conference, ECMFA 2016, Held as Part of STAF 2016, Vienna, Austria, July 6-7, 2016, Proceedings. pp. 19–35 (2016), http://dx.doi.org/10.1007/978-3-319-42061-5_2
12. Lamo, Y., Wang, X., Mantz, F., Bech, Ø., Sandven, A., Rutle, A.: DPF workbench: A multi-level language workbench for MDE. Proc. of the Estonian Acad. of Sciences 62, 3–15 (2013)
13. Mac Lane, S.: Categories for the Working Mathematician, Second edition. Springer (1998)
14. Makkai, M.: Generalized sketches as a framework for completeness theorems. J. Pure Appl. Algebra 115, 49–79, 179–212, 214–274 (1997)
15. Rabbi, F., Lamo, Y., Yu, I., Kristensen, L.: A diagrammatic approach to model completion. In: 4th Workshop on Analysis of Model Transformations co-located with MODELS 2015. pp. 56–65 (2015)
16. Rutle, A., Rossini, A., Lamo, Y., Wolter, U.: A diagrammatic formalisation of mof-based modelling languages. In: Oriol, M., Meyer, B. (eds.) TOOLS EUROPE. Lecture Notes in Business Information Processing, vol. 33, pp. 37–56. Springer (2009), http://dx.doi.org/10.1007/978-3-642-02571-6_4
17. Rutle, A., Wolter, U., Lamo, Y.: A Diagrammatic Approach to Model Transformations. In: Proceedings of the 2008 Euro American Conference on Telematics and Information Systems (EATIS 2008). pp. 1–8. ACM (2008)
18. Sabetzadeh, M., Nejati, S., Liaskos, S., Easterbrook, S., Chechik, M.: Consistency checking of conceptual models via model merging. In: RE. pp. 221–230 (2007)
19. Sabetzadeh, M., Nejati, S., Liaskos, S., Easterbrook, S.M., Chechik, M.: Consistency Checking of Conceptual Models via Model Merging. In: RE. pp. 221–230. IEEE (2007)

7 Mathematical Background

In this section, we justify Prop.1 and Theorem 1 with the help of appropriate underlying mathematical theorems.

7.1 A closer look at partial morphisms

If $G \xrightarrow{f_1} G'$, we will denote the embedding $\mathrm{dom}(f) \subseteq G$ by $\mathrm{dom}(f) \xhookrightarrow{f_{-1}} G$. Hence partial morphisms can equivalently be coded as spans

$$G \xhookleftarrow{f_{-1}} \mathrm{dom}(f) \xrightarrow{f_1} G'$$

of total morphisms. The described composition of two partial morphism $G \xrightarrow{f_1} G'$ and $G' \xrightarrow{f_2} G''$ in Sect.3.3 can then be carried out as shown in figure 14. Here *em* is the pullback of f_{-2} along f_1. It can be shown that *em* can be chosen to be an embedding, as well. The composition $G \xrightarrow{f_2 \circ f_1} G''$ is then defined to be the pair of total mappings $G \xhookleftarrow{f_{-1} \circ em} H \xrightarrow{f_2 \circ f_1'} G''$.

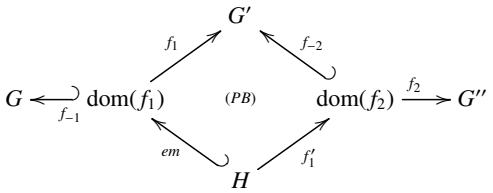


Fig. 14: Composition of Partial Morphisms

In the sequel, we will treat the use of any partial morphism in complete multi-instances as span of total morphisms and composition will be carried out as just described. E.g. partial m_j is actually a pair $(m_{-j} : \mathrm{dom}(m_j) \to M_0, m_j : \mathrm{dom}(m_j) \to M_j)$ of total morphisms. Composition of total τ_0 with partial m_j thus translates the type compatibility conditions (4) for complete multi-instances to the fact that in the commutative diagram

$$\begin{array}{ccccc} A_0 & \xhookleftarrow{a_{-j}} & \mathrm{dom}(a_j) & \xrightarrow{a_j} & A_j \\ \downarrow{\scriptstyle \tau_0} & (PB) & \downarrow{\scriptstyle \tau_{-j}} & & \downarrow{\scriptstyle \tau_j} \\ M_0 & \xhookleftarrow[m_{-j}]{} & \mathrm{dom}(m_j) & \xrightarrow[m_j]{} & M_j \end{array} \tag{6}$$

the left square is a pullback for all $j \in \{1, \ldots, n\}$.

7.2 A closer look at pulling back multi-instances

In Sect.4.4 we described how the pullback operation along any morphism $\delta : X \to M^+$ can be extended to partial morphisms $H_1 \xrightarrow{f} H_2$, if $k_1 : H_1 \to M^+$ and $k_2 : H_2 \to M^+$ are related to f by the equation $h_2 \circ f = h_1$, see Fig.9. If $f : H_1 \to H_2$ is total, we write

$$f : k_1 \to k_2$$

to underline the compatibility with k_1 and k_2. It is an important observation that in a complete (and hence also in an incomplete) multi-instance $\mathcal{T} = ((\tau_i : A_i \to M_i)_{0\leq i\leq n}, (a_j : A_0 \rightharpoonup A_j))$ over multi-model $\mathcal{M} = (m_i)_{0\leq i\leq n}$, operator δ^{**} can be applied to each involved total graph morphism, because it can easily be verified that

- $\tau_i : k_i \circ \tau_i \to k_i$ $(-n \leq i \leq n)$
- $m_j : k_{-j} \to k_j$
- $m_{-j} : k_{-j} \to k_0$
- $a_j : k_{-j} \circ \tau_{-j} \to k_j \circ \tau_j$
- $a_{-j} : k_{-j} \circ \tau_{-j} \to k_0 \circ \tau_0$

$(1 \leq j \leq n)$. Here $k_{-j} := k_j \circ m_j$, and τ_{-j} arises from the new type compatibility diagram (6). This shows that operator $\delta^\star$ in Prop.1 is well-defined and its image is nothing else than the image of all involved morphisms under the pullback functor [10] from $\mathbb{G}\downarrow M^+ \to \mathbb{G}\downarrow X$. More precisely

$$\delta^\star((\tau_i)_{-n\leq i\leq n}, (a_{-j}, a_j)_{1\leq j\leq n}) = ((\delta^{**}(\tau_i))_{-n\leq i\leq n}, (\delta^{**}(a_{-j}), \delta^{**}(a_j))_{1\leq j\leq n}) \tag{7}$$

and likewise for incomplete multi-instances. We have proved Prop.1.

7.3 More Categorical Facts

In order to prove Theorem 1, we provide some important categorical theorems. We use the terms "(comma) category", "colimit(ing cocone)", "diagram" in the usual categorical sense (see e.g. [2]). Furthermore, we will use the fact, that operation *merge* and *Merge* (see Defs 3 and 5) implement the colimit construction, for which, in the sequel, the well-known universal properties will be used[10]. Moreover, for $\delta : S \to M$, we will not distinguish anymore bewteen the application of the pullback functor to objects of $\mathbb{G}\downarrow M$ (δ^*) and to morphisms of $\mathbb{G}\downarrow M$ (δ^{**}). In both cases we will write δ^*.

Lemma 1. *Let $\mathbb{G}$ be the category of graphs, $\mathbb{I}$ a schema graph, $\mathcal{D} : \mathbb{I} \to \mathbb{G}$ a diagram and $k = (k_i)_{i\in\mathbb{I}_0} : \mathcal{D} \Rightarrow M$ a cocone. Consider the diagram*

$$\mathcal{D}^M : \begin{cases} \mathbb{I} \to \mathbb{C}\downarrow M \\ i \xrightarrow{d} j \;\mapsto\; k_i \xrightarrow{\mathcal{D}_d} k_j \end{cases}$$

1. *$k : \mathcal{D} \Rightarrow M$ is a colimiting cocone if and only if $k : \mathcal{D}^M \Rightarrow id_M$ is a colimiting cocone in the comma category $\mathbb{G}\downarrow M$.* [13]

[13] k is then interpreted as the family $(k_i : k_i \to id_M)_{i\in\mathbb{I}_0}$ of $\mathbb{C}\downarrow M$-morphisms.

2. *Let $\delta : S \to M$. If $k : \mathcal{D} \Rightarrow M$ is a colimiting cocone in $\mathbb{G}$ then so is its image under the pullback functor δ^*.*

The statement is applicable to the present setting: $\mathbb{I}$ is taken to be the schema graph for matching w.r.t. to total morphisms, i.e. it has vertices $\{-n, \ldots, n\}$ and for each $j \in \{1, \ldots, n\}$ two edges $d_{-j} : -j \to 0$ and $d_j : -j \to j$. $\mathcal{D}_{d_{-j}}$ must be an embedding in all diagrams such that any diagram represents n partial morphisms $\mathcal{D}_0 \xleftarrow{\mathcal{D}_{d_j}} \mathrm{dom}(\mathcal{D}_{d_j}) \xrightarrow{\mathcal{D}_{d_j}} \mathcal{D}_j$.

Proof of Lemma 1: Second statement: If k is a colimiting cocone in $\mathbb{G}$, then so is k seen as a cocone of $\mathbb{G} \downarrow M$ (using the "only-if"-part of the first statement). It is well-known that the pullback functor $\delta^* : \mathbb{G} \downarrow M \to \mathbb{G} \downarrow S$ preserves colimits [10], hence the "if-part" of the first statement now applied to $M := S$ yields the result.

The "if-part" of the first statement follows from the fact that the functor

$$dom : \left\{ \begin{array}{ccc} \mathbb{G}\downarrow a \to \mathbb{G} \\ c \xrightarrow{h} d \;\mapsto\; c \xrightarrow{h} d \\ \quad f \searrow \quad \swarrow g \\ \quad\quad a \end{array} \right.$$

is left-adjoint to the functor $\_ \times a$, which assigns to any $c \in \mathbb{G}$ the projection $\pi_2 : c \times a \to a$, and hence preserves colimits [13].

"Only-if"-Part: Let $\sigma : \mathcal{D}^M \Rightarrow t$ be any commutative cocone in $\mathbb{C} \downarrow M$ with $t : T \to M$. Since $\sigma_i = \sigma_j \circ \mathcal{D}_d$ whenever $d : i \to j$ in $\mathbb{I}_1$, the assumption yields unique $u : M \to T$ with

$$\forall i \in \mathbb{I}_0 : u \circ k_i = \sigma_i \tag{8}$$

We also obtain the cocone $(t \circ \sigma_i : \mathcal{D}_i \to M)_{i \in \mathbb{I}_0}$ yielding unique $v : M \to M$ such that for all i: $v \circ k_i = t \circ \sigma_i$. Since $\sigma_i : k_i \to t$ is a $\mathbb{G} \downarrow M$-morphism, i.e. $t \circ \sigma_i = k_i$, all i, by uniqueness of v: $v = id$. Furthermore, we also have $t \circ u \circ k_i = t \circ \sigma_i$ for all i by (8), i.e. $t \circ u = v$, hence $t \circ u = id$, s.th. $u : id_M \to t$ is the unique mediator of $\mathbb{G} \downarrow M$. □

The next statement can be found in any textbook on category theory, e.g. [2].

Lemma 2 (Composition of Pullbacks). *Let $\delta_1 : S \to R$ and $\delta_2 : R \to M$ be two graph morphisms and $\delta = \delta_2 \circ \delta_1 : S \to M$. Then for any morphism $f : X \to M$*

$$\delta_1^*(\delta_2^*(f)) \cong \delta^*(f).$$

7.4 Proof of Theorem 1

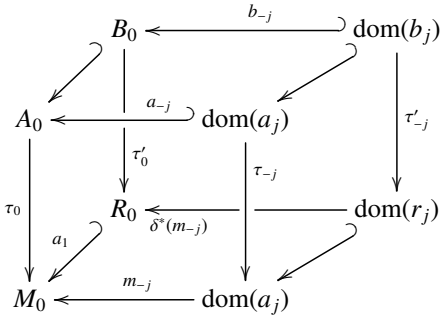
Obviously, we must prove

$$\mathcal{T} \models c@\delta \iff validate_c(\tau'') . \tag{9}$$

First of all we show that the applictation of the pullback functor δ_r^* to a hypothetical global match, i.e. to complete multi-instance $\overline{\mathcal{T}}$ over $\mathcal{M}$ equals the result of step 2 of LMM: This is trivially true for morphisms $\delta_r^*(k_i)$, for $\tau'_j = \delta_r^*(\tau_j)$ and for the morphisms $\delta^*(m_j/m_{-j})$ $(1 \le j \le n)$ of multi-model $\mathcal{R}$, see the incomplete case in Prop.1. It is

also true for τ'_0 and the b_j's by (5), since these properties exactly express the fact that $\tau'_0 = \delta^*_r(\tau_0)$ and $b_j = \delta^*_r(a_j)$.

It remains to show that τ'_{-j}, which arises from (6) for $\overline{\mathcal{T}}'$, equals $\delta^*_r(\tau_{-j})$. This follows from the fact that in



(10)

the 4 arrows from the back to the front can all be taken to be embeddings (because pullbacks preserve the monomorphism δ_r [10]). Moreover, the floor (by construction) and the back (by (6) and because in step 2 completion yields $r_j \circ \tau'_0 = \tau'_j \circ b_j$) of the commutative (!) cube are pullbacks, such that the composition of front and roof is pullback[2]. Since the front is pullback (again (6) and $m_j \circ \tau_0 = \tau_j \circ a_j$ in the hypothetical global match over $\mathcal{M}$), this is also true for the roof by pullback decomposition[2]. Because we already established $\tau'_0 = \delta^*(\tau_0)$ and because it can also be shown that the top square in the right of Fig.9 becomes a pullback square (use pullback decomposition), the left face of the above cube is pullback. Hence, the same argumentation as above shows that the right face is pullback, hence $\tau'_{-j} = \delta^*(\tau_{-j})$ as desired.

This yields two aspects

1. Since M^+ is the colimit of $\mathcal{M}$ with colimiting cocone $(k_i)_{-n \le i \le n}$, Lemma 1 justifies that $\delta^*(k_i)_{-n \le i \le n}$ is the colimiting cocone of $\mathcal{R}$, hence operator $\delta^\star$ is applicable to $\overline{\mathcal{T}}'$ in step 3 (LMM).
2. By Lemma 2 the result $\overline{\mathcal{T}}''$ of LMM-step 3 is the image of $\overline{\mathcal{T}}$ under $\delta^\star$ up to renaiming.

Statement 2 of Lemma 1 and the fact that colimits in the arrow category (operation *Merge*) can be constructed separately on the data and the model level, yield

$$\delta^*(Merge(\overline{\mathcal{T}})) \cong Merge(\delta^\star(\overline{\mathcal{T}}))$$

Since $\tau^+ = Merge(\overline{\mathcal{T}})$ (see Sect.5.1) and $\tau'' = Merge(\overline{\mathcal{T}}'')$ (step 4), this yields

$$\delta^*(\tau^+) \cong Merge(\overline{\mathcal{T}}'') = \tau'' \tag{11}$$

and hence the codomain of τ'' can indeed be taken to be S in step LMM-4. It remains to prove (9). This follows from the following chain of equivalences:

$$
\begin{aligned}
\mathcal{T} \models c@\delta &\iff \tau^+ \models c@\delta && \text{(According to the definition in Sect.5.1)}\\
&\iff \mathit{check}(\tau^+, c@\delta) = \mathit{true} && \text{(Def. of } \models \text{ in Sect.3.2)}\\
&\iff \mathit{validate}_c(\delta^*(\tau^+)) = \mathit{true} && \text{(Def. of function CHECK in Sect.3.2)}\\
&\iff \mathit{validate}_c(\tau'') = \mathit{true}, &&
\end{aligned}
$$

where the last equivalence is due to (11) and insensitivity of VALIDATE w.r.t. isomorphic distortion.

UNIVERSITY OF APPLIED SCIENCES

FHDW

FACHHOCHSCHULE FÜR DIE WIRTSCHAFT
HANNOVER

ISSN 1863-7043