

# Forschungsberichte der FHDW Hannover

---

## Individual Local Checking of Global Consistency in Heterogenous Multimodeling: The Story behind the Scenery

*Harald König, Zinovy Diskin*

Bericht Nr.: 02016/01

---

Fachhochschule für die Wirtschaft Hannover  
Freundallee 15  
30173 Hannover  
[techrep@fhdw.de](mailto: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](mailto:techrep@fhdw.de)

ISSN 1863-7043

# Individual Local Checking of Global Consistency in Heterogenous Multimodeling: The Story behind the Scenery \*

Harald König

University of Applied Sciences FHDW Hannover, Germany  
harald.koenig@fhdw.de

Zinovy Diskin

NECSIS, McMaster University, Canada  
zdiskin@uwaterloo.ca

Software design requires deployment of interdependent models conforming to different metamodels. This set of models is called a *multimodel*, and it must satisfy a set of *global* constraints which simultaneously constrain multimodel components.

Categorically, a multimodel is a diagram and a straightforward approach to global consistency checking constructs the colimit of component metamodels, adding, perhaps, new global constraints, constructs the colimit of models, and checks the latter colimit against the constraints in the former one. Being a natural *definition* for global consistency, these steps can not be used algorithmically because of two major practical drawbacks: they involve costly (meta)model matching to specify overlaps, and they require building big and unfeasible colimit metamodels and models.

The present paper presents a new algorithm to check each global constraint individually, and *as local as possible*, i.e., only using those (meta)model elements that affect the validity of the constraint. We present a theorem which secures equivalence of global definition and local algorithm.

## 1 Introduction

Modeling a complex system normally results in a *multimodel*, i.e., a set of heterogenous models each one conforming to its own metamodel. A fundamental fact about multimodeling is that the merge of legal local models can result in a model violating the global constraints declared in the integrated metamodel. This can be easily observed even for the simple homogeneous case, when all local models, and hence their merge, are instances of the same metamodel. For example, suppose that the metamodel of a domain says that (car insurance) contracts are uniquely identified by a field 'contract-id', i.e., the field has imposed a unique key constraint. Then the merge of two perfectly legal local instances can violate the constraint, if there are (physically) different insurance contracts with the same 'contract-id' but they do not appear in the same instance.

*Heterogeneous* multimodeling expands the issue of global consistency enormously. For example, consider a metamodel  $M_1$  that extends the Contract metamodel above with attribute 'contractType' (with values 'standard', 'extended'), and a metamodel  $M_2$  that extends the metamodel with reference 'traffic telematics enabled' to class 'Contract'. Suppose that the domain is 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 multimodel  $(\tau_1, \tau_2)$  with  $\tau_{1,2}$  being legal instances of

---

\*This work is supported by the Automotive Partnership Canada via the Network on Engineering Complex Software Intensive Systems (NECSIS)

$M_{1,2}$  is important. A more complex example is consistency between a UML sequence diagram specifying collaborative behavior, and a statechart specifying a state machine protocol for that behavior. An obvious consistency requirement that traces specified by the sequence diagram should be allowed by the statechart is again global and cannot be declared in either of the local metamodels. Following [7], we call such requirements *inter-metamodel constraints*.

In general, complex relations of metamodels are diagrams  $\mathcal{D}$  in a suitable category, metamodels are objects, relations are arrows between them and probably auxiliary objects. A straightforward approach to global consistency checking would require constructing the colimit of  $\mathcal{D}$  (yielding contracts with type and telematics information in the above example), adding, perhaps, new global constraints to this merge ('only extended contracts can be controlled via traffic telematics'), merging component models in the same way, and checking the model merge (again colimit) against the constraints over the metamodel merge. 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) costly (meta)model matching needed to specify the overlaps, and (b) necessity to build big and unfeasible merges of metamodels and models.

A more efficient approach is already proposed in [7, 2]. It prescribes to do matching, merging and checking not for entire component models but for certain projections to the respective metamodel portions by grouping adjoint constraints accordingly (partially local). For exactly one constraint under consideration and projection to the smallest possible fragment, the approach coincides with ours.

As a first step, the present paper makes this approach more precise for the binary case of two metamodels. We provide an algorithm for individual constraint checking as local as possible, and prove equivalence of global definition and local approach. The astonishing fact is that the algorithm makes checking *completely independent* of model matching outside the constraint fragments. The equivalence has already been stated in [7] for the partially local approach, but was not yet proven formally. The generalization of the algorithm from the binary case to the general case seems to be straightforward whereas the generalization of the proof from pushout to general colimits is more involved and subject to future research.

Besides reduced matching and merging workload, additional advantages of the new algorithm are (a) better tailored and stepwise model repairing (in the *per constraint* fashion), and (b) possibilities to realize the *living with inconsistency* paradigm [9], when non-urgent consistency repairs (together with the respective matching and merging) can be postponed.

We start the illustration with background considerations concerning underlying categorical structures, a short report on constraint checking diagrammatically and necessary theoretical results for graph-like structures (Sect. 2). After having introduced multi(meta)models, we briefly describe the above mentioned global approach, state the algorithm for the local one and prove their equivalence (Sect.3). We conclude the paper with possible future research directions in Sect.4.

## 2 Background

Metamodels are usually specified by UML class diagrams. The compact syntax of the latter hides many details that need to be explicated and formalized. In this section, we perform this formalization: We use directed graphs as underlying category, diagrammatic predicates and explain the definition of multimodels. The fourth section provides the necessary theoretical background for the proof of the main theorem in Sect.3.3. The formalism of diagrammatic constraints, first developed under the name of *generalized sketches* [4, 6], and then promoted as the *Diagram Predicate Framework, DPF* [17, 16], is less

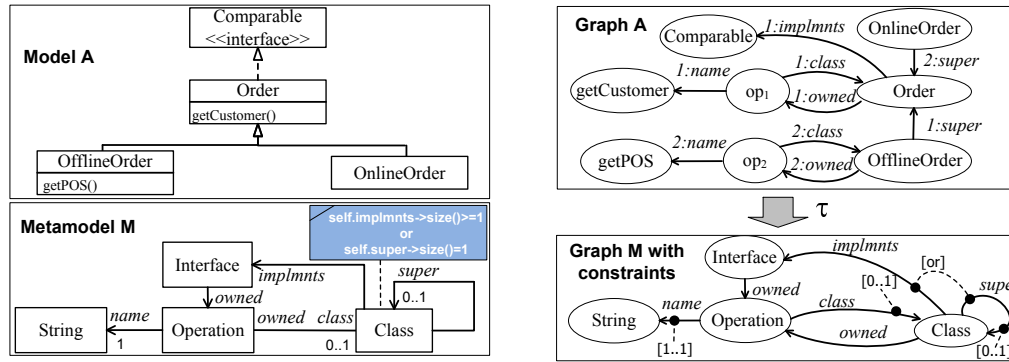


Figure 1: UML model and metamodel represented as typed graph

known, and we present in Sect. 2.1 its basics in the amount needed for our work in the paper to make it self-contained. Sect.2.2 introduces multimodels and Sect.2.3 lists some basic facts needed for the forthcoming proof.

## 2.1 Diagrammatic Constraints

A (directed multi-)graph  $G = (V, E, src, tgt)$  consists of a set  $V$  of *vertices* (or *nodes*), a set  $E$  of *edges*, and two functions  $src: E \rightarrow V$ ,  $tgt: E \rightarrow V$  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 graphs in the usual way: vertices are ellipses (or circles) and edges are arrows from their source to their target vertex, cf. Fig.1, graphs  $A$  and  $M$ . A graph *morphism*  $f: G \rightarrow G'$  between graphs  $G = (V, E, src, tgt)$  and  $G' = (V', E', src', tgt')$  is a pair of functions  $f_V: V \rightarrow V'$  and  $f_E: E \rightarrow E'$  preserving the incidence between vertices and edges, i.e.  $\forall e \in E: f_V(src(e)) = src'(f_E(e))$ ,  $f_V(tgt(e)) = tgt'(f_E(e))$ . Since the definition of  $f$  on an edge  $e$  determines the mapping on  $e$ 's source and target, we will sometimes describe graph morphisms by giving its mapping behavior on edges only.  $\mathbb{G}$  denotes the category of graphs and graph morphisms.

The left lower quadrant of Fig. 1 presents a fragment of a metamodel for UML class diagrams with several constraints declared. Three multiplicity constraints are depicted in the usual UML style. They prescribe (i) each operation to have a unique name and (ii) belong to at most one class, and (iii) prohibit multiple inheritance. The more complex OCL-constraint (the coloured memo) for classes shall guide developers to code their programs in a polymorphic style (if there is no superclass, there should be at least one interface implementation and vice versa). The left upper quadrant shows a class diagram instantiating the metamodel.

The right half of the figure shows the translation into a typed graph. The metamodel is presented as a *type* graph  $M$  with a set of four constraint declarations. Each of them consists of a constraint name given in square brackets, and the constraint *scope* shown by dashed lines. The (typed) model is a graph morphism  $\tau: A \rightarrow M$ , i.e. a *typing* mapping between graphs, which assigns types to every model element, e.g.  $\tau(Order) = Class$ ,  $\tau(op_1) = Operation$ ,  $\tau(getCustomer) = String$ , as well as  $\tau(1:implmnts) = implmnts$ , and so on. We will write  $a:T$ , if  $\tau(a) = T$ .  $\tau$  is called a typed graph (over  $M$ ). The object collection of the category of all typed graph over  $M$  will be denoted  $Mod(M)$ .

A key feature of constraints used in metamodeling 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 any multiplicity constraint is a single arrow, while the shape of the *or*-constraint is

two arrows with a common source, see Table 1.

To declare a constraint over a metamodel graph  $M$ , we recognize the constraint shape in the graph and visualize it as was shown in Fig. 1. This recognition is a graph mapping  $\delta : S^c \rightarrow M$  (called (*shape*) *binding*) from the arity shape  $S^c$  of a constraint with name  $c$  to graph  $M$ . E.g. in Fig.1, we have constraint  $[or]$  declared by binding  $\delta : S^{[or]} \rightarrow M$  ( $S^{[or]}$  is shown in Table 1) with  $\delta(01) = implmnts$ ,  $\delta(02) = super$ , i.e.  $\delta(1) = Interface$ ,  $\delta(0) = Class = \delta(2)$ . The set of elements in  $M$  the shape is mapped to, is called the *image* of the binding.

The pair  $(c, \delta)$  is called *constraint declaration*. Note that for the  $[0..1]$ -declarations in  $M$  we can reuse shape  $S^{[0..1]}$  in two different bindings, one of them mapping edge 12 to edge *super*, the other mapping 12 to edge *class*. In the sequel, we write  $c@\delta$ , meaning *constraint  $c$  is imposed on metamodel  $M$  at image of binding  $\delta$* .

In order to check consistency of typed graph  $\tau$  against a fixed constraint declaration  $c@\delta$ , we need to define  $c$ 's *semantics* irrespective to  $\tau$ . This is done by programming a validator function

$$\text{VALIDATE}_c(\tau^c : \mathbb{G} \downarrow S^c) : \text{BOOLEAN}$$

which has input typed graph  $\tau^c : A^c \rightarrow S^c$ . Semantics is used in the check function:

$$\text{CHECK}(\tau : \mathbb{G} \downarrow M, c@\delta : \text{String} \times \mathbb{G} \downarrow M) : \text{BOOLEAN}$$

which, basically, performs two steps:

1. Construct *pullback*  $\tau^c : A^c \rightarrow S^c$  of  $\tau : A \rightarrow M$  along  $\delta : S^c \rightarrow M$  (short:  $\tau^c = \delta^*(\tau)$ ).
2. Return the result of  $\text{validate}_c(\tau^c)$ .

Thus, semantics of a constraint is defined only once and then reused for each constraint declaration with this constraint name. We say that  $\tau$  *satisfies*  $c@\delta$ , written  $\tau \models c@\delta$ , if  $\text{CHECK}(\tau, c@\delta) = \text{true}$ . Model  $\tau$  is a *legal* model over metamodel  $M$ , if it satisfies all constraints declared in  $M$ .

This mechanism has been decribed in detail in [3, 17]. it can be observed in Fig.2.  $\text{VALIDATE}_{[or]}$  returns true if each element of type 0 in  $A^c$  has an outgoing edge to some element of type 1 or to some element of type 2. The image of  $\delta$  is shown in the lower right part (elements not in the image are veiled).  $A$  is restricted (top right quadrant),  $A^c$  is the domain of  $\tau^c$ . Obviously,  $\text{VALIDATE}_{[or]}(B) = \text{true}$ .

Note the "pullback intelligence": For each *class*-instance in  $A$ , we have to create *two* vertices in  $A^c$ , because we must incorporate their two possible roles as subclass (source of edge *super*) and superclass (target of edge *super*).

## 2.2 Multimodeling

Modeling a complex system normally results in a *multimodel*, i.e., a set of heterogenous models each one conforming to its own metamodel. Besides class diagrams, other types of UML diagrams are produced, for instance sequence diagrams, statecharts, activity diagrams, etc. Even class diagrams may conform to different metamodels: Business analysts may use behavioural specifications only [10] (no attributes or associations), whereas in another modeling team class models are more technically oriented and associations and attributes are used. In all cases the models collectively represent a single system to be build and any formal treatment has to consider *overlaps*, i.e. the definitions of common terminology in different models. E.g. (meta) concepts *class* occur in the above mentioned behavioural specifications

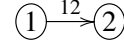
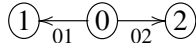
Name	Shape
[0..1]	
[or]	

Table 1: Sample Constraints

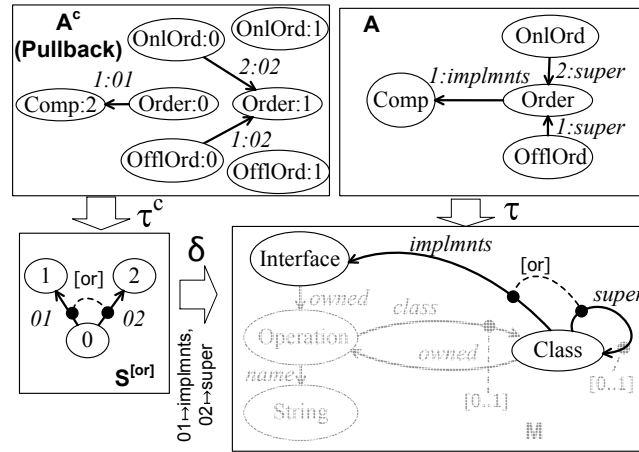


Figure 2: Function CHECK

and in technical metamodels. Names of common concepts, however, may differ: One development team may use the term *String* for character strings, the other may use *Text*, yet speaking of the same thing.

In the binary case (two metamodels  $M_1$  and  $M_2$ ), *overlaps* can be specified by two graph mappings

$$M_1 \xleftarrow{r_1} M_0 \xrightarrow{r_2} M_2$$

in which  $M_0$  contains all common concepts. Any pair  $x_1 \in M_1$  and  $x_2 \in M_2$  is declared to be the same, if there is  $x \in M_0$  such that  $r_1(x) = x_1$  and  $r_2(x) = x_2$ . We call the pair (the span)  $(r_1, r_2)$  a *multimetamodel*.

A multimetamodel is shown in Fig.3:  $M_1$  is the metamodel already used in Fig.1, but without constraints.  $M_2$  is a more technical metamodel which specifies *Classes* with superclasses, attributes and directed associations.  $M_0$  is the overlap specification. It declares *Class* together with its super-relation to be in the overlap of both components and, by defining  $r_1(Str\_Txt) = String$  and  $r_2(Str\_Txt) = Text$  it also states that *String* and *Text* are the same concept. Identity relation are represented as shaded vertices. The *pushout*  $M_1 +_{M_0} M_2$  represents the merge of the multimetamodel components.

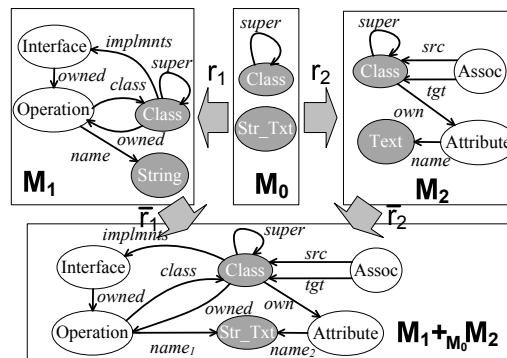


Figure 3: Multimodel and Merge

### 2.3 Facts for Topoi

A topos is a category with finite limits and colimits, which is cartesian closed, and where the subobject functor is representable. We list some facts:

1.  $\mathbb{G}$  is a topos [13].
2. In topoi, pushouts with one monic leg are Van Kampen Squares [14].
3. There is an exact characterization of the Van Kampen property for presheaf topoi. It shows that pushouts may also enjoy this property although both legs are not monic [19].
4. If, in a topos, the squares

$$\begin{array}{ccc} A_i & \xrightarrow{a_i} & A \\ f_i \downarrow & \lrcorner & \downarrow g \\ M_i & \xrightarrow{g_i} & M \end{array}$$

are pullbacks<sup>1</sup> for  $i \in \{1, 2\}$ , then so is

$$\begin{array}{ccc} A_1 + A_2 & \xrightarrow{[a_1, a_2]} & A \\ f_1 + f_2 \downarrow & \lrcorner & \downarrow g \\ M_1 + M_2 & \xrightarrow{[g_1, g_2]} & M \end{array}$$

[11]

5. Pullbacks preserve epis as well as monos [13].
  6. Let in a topos a commutative diagram be given with an epimorphism as indicated. If (1) + (2) and (1) are pullbacks, then (2) is pullback, too, [1].
- $$\begin{array}{ccccc} \cdot & \longrightarrow & \cdot & \longrightarrow & \cdot \\ \downarrow & & \downarrow & & \downarrow \\ \cdot & \xrightarrow{(1)} & \cdot & \xrightarrow{(2)} & \cdot \\ \downarrow & & \downarrow & & \downarrow \\ \cdot & \longrightarrow & \cdot & \longrightarrow & \cdot \end{array}$$
7. Pullback functors admit right-adjoints [13], consequently they preserve colimits.
  8. All morphisms admit unique epi-monic-factorizations up to isomorphism [13]. If  $f : A \rightarrow C$  is decomposed in this way, i.e.  $f = f^m \circ f^e$ , we call  $f^e(A)$  the image of  $f$ .

## 3 Managing Global Constraints

In the present section we analyse *global* constraints, that is, constraints that reside in neither of the component metamodells alone, and thus involve elements from several metamodells. Correspondingly, we use the name *inter-metamodel constraints* that accurately describes the case. In Sect.3.1, we will state a *definition* of *global satisfaction* against an inter-metamodel constraint. The definition treats the binary case only, but the generalization for the N-ary case is straightforward. Models typed over different metamodells are said to be *globally* consistent if they satisfy all imposed inter-metamodel constraints. We will argue that it is impractical to use this definition as an algorithm for global consistency checking. Hence, in Sect.3.2, we introduce another algorithm, in which global satisfiability against an individual inter-metamodel constraint is checked *locally*, and illustrate its advantages with a running example. Sect.3.3 compares the global satisfaction definition of Sect. 3.1 with the local algorithm of Sect.3.2. Finally, the equivalence of all three methods is stated as our main theorem.

<sup>1</sup> An angle in one corner of a commutative diagram indicates universality - e.g. pullback.



### 3.1 Global Consistency

Inter-metamodel constraints reside in different components of a multimetamodel  $(r_1, r_2)$ , i.e. specify *global* constraints. Consider e.g. a binary multimetamodel as in Fig.3. One may, for instance, specify the following constraint declaration on classes<sup>2</sup>:

$[acc]@\delta$ : For each attribute with name  $n$  there has to be an accessor with name  $getN$ .

and ask whether several models typed over  $M_1$  and  $M_2$ , resp., satisfy this constraint.

The declaration  $[acc]@\delta$  spreads over different metamodels, thus a binding is possible only on the pushout of  $r_1$  and  $r_2$ . cf. Fig.3 in which morphisms  $\bar{r}_1 : M_1 \rightarrow M_1 +_{M_0} M_2$  and  $\bar{r}_2 : M_2 \rightarrow M_1 +_{M_0} M_2$  map all elements of the metamodel components to the corresponding element in the pushout. Now we can impose  $[acc]$  to the pushout object with the help of binding map  $\delta$ . This is shown in Fig.4, where  $\delta$  maps according to the indicated shape.

$S^{[acc]}$ 's intended semantics is controlled by function  $VALIDATE_{[acc]}$  (cf. Sect.2.1), which has input a typed graph over  $S^{[acc]}$ . It will return true if and only if for each  $x:0$  for any reached element  $n:3$  along link paths of the form  $e_1:02, e_2:23$  there is a path  $e'_1:01, e'_2:13$  starting at vertex  $x:0$  and ending at vertex  $getN:3$ . Note that the *super* relation is not included in the image of  $\delta$ , because getters shall exist for *own* attributes only (inherited attributes already yield respective *get*-methods).

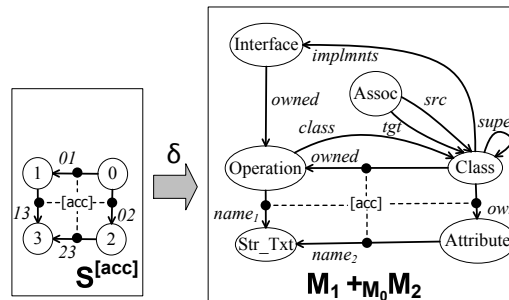


Figure 4: Imposing global constraint on merged multimodel

In Sect.2.2 we described two modeling teams. Assume the first team creates legal model (one or more class diagrams)  $\tau_1 : A_1 \rightarrow M_1 \in Mod(M_1)$  and the other team creates legal model  $\tau_2 : A_2 \rightarrow M_2 \in Mod(M_2)$ . What about validity of  $[acc]@\delta$ ? Again, conjoint treatment of models requires their matching, i.e. to decide on common concepts in the models. But *model overlap* might not be possible to be deduced automatically: for example, an entity *Online Order* in  $A_1$  maybe called *Online Purchase Order* in  $A_2$ . In general, cross-(meta)model terminology may be very heterogeneous, and the structure of models may vary significantly while still reflecting identical concepts. Therefore, modelers (users) must costly determine (type-conformant) overlap  $A_0$  of  $A_1$  and  $A_2$  manually. Since the entire set of models can be of significant size, manual can be considerable.

Formally – and similarly to metamodels – one must determine two graph morphisms

$$A_1 \xleftarrow{a_1} A_0 \xrightarrow{a_2} A_2$$

<sup>2</sup>A standard requirement for Java Beans©

and the overlap typing  $\tau_0 : A_0 \rightarrow M_0$ , such that  $r_1 \circ \tau_0 = \tau_1 \circ a_1$  as well as  $r_2 \circ \tau_0 = \tau_2 \circ a_2$ . Only now is it possible to perform the component merge of the multimodel, which, basically, is carried out in the same way as for metamodels: One constructs the pushout  $A_1 +_{A_0} A_2$ , which – by the universal property of pushouts – yields a unique typing mapping

$$\tau_1 +_{\tau_0} \tau_2 : A_1 +_{A_0} A_2 \rightarrow M_1 +_{M_0} M_2$$

We can thus say, that a (typed) *complete multimodel*<sup>3</sup> (over multimetamodel  $(r_1, r_2)$ ) is a span

$$\tau_1 \xleftarrow{(r_1, a_1)} \tau_0 \xrightarrow{(r_2, a_2)} \tau_2$$

in the arrow category  $\mathbb{G}^{\rightarrow}$  with merge (pointwise pushout)

$$\tau_1 \xrightarrow{(\bar{r}_1, \bar{a}_1)} \tau_1 +_{\tau_0} \tau_2 \xleftarrow{(\bar{r}_2, \bar{a}_2)} \tau_2$$

In the light of the above considerations, this yields the following terminology [18, 7]:

**Definition 1 (Global Consistency)** *Given an inter-metamodel constraint  $c@ \delta$  over multimetamodel  $(r_1, r_2)$ . We say that complete multimodel*

$$\tau_1 \xleftarrow{(r_1, a_1)} \tau_0 \xrightarrow{(r_2, a_2)} \tau_2$$

*satisfies  $c@ \delta$ , if*

$$\tau_1 +_{\tau_0} \tau_2 \models c@ \delta.$$

*If the multimodel satisfies all inter-metamodel constraints imposed on  $(r_1, r_2)$ , we call it globally consistent.*

We remark that the binary case can be generalized to the N-ary case (arbitrary diagrams) by constructing *colimits* instead of pushouts.

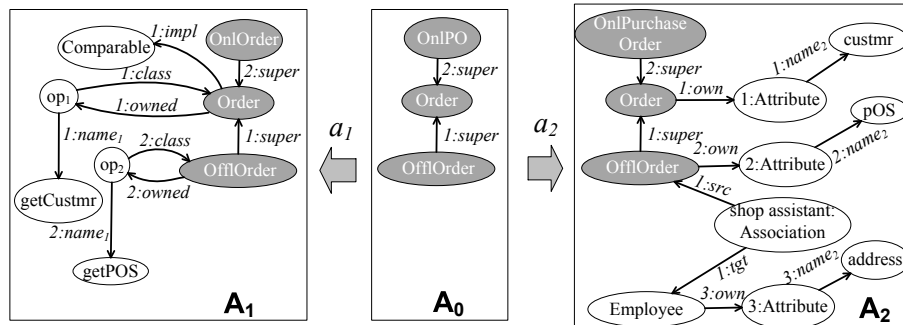


Figure 5: Multimodel: Models with Overlap

Unfortunately, the approach of checking consistency along the lines of this definition, i.e constructing *globally* typed data before checking, has major disadvantages:

<sup>3</sup> In contrast to *incomplete* multimodels to be explained a little bit later.

1. 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.
2. To specify overlaps of typed data structures, this enormous collection of data has to be traversed manually or at least semi-automatically. Overlaps have to be complete, i.e. they are not specific to the given constraint declaration.

Consider e.g. Fig.5:  $A_1$  contains owned operations and implemented interfaces of the order classes.  $A_2$  represents the same order classes. Shaded nodes and their *super*-links are in the overlap  $A_0$ , i.e. *OnlOrder* and *OnlPurchaseOrder* are declared to be the same classes despite their different names. Besides own attributes,  $A_2$  contains the *shop assistant* who processed the *Offline Order* (via a directed association). The huge pushout is not shown due to lack of space.

If we want to check whether the multimodel satisfies constraint  $[acc]@\delta$ , the above mentioned disadvantages manifest as follows:

1. Although  $[acc]@\delta$  only "talks" about classes and names of their attributes and operations, we have to deal with interface implementations and operation's reference to its class (from  $A_1$ ), as well as (usually many) associations (from  $A_2$ ), but also with superclass relations (from the overlap).
2. The user must search the set of all classes and all their superclass relations for identical concepts. In the example he must specify sameness of *OnlOrder* and *OnlPurchaseOrder*, the other two identities *Order* and *OfflOrder* may automatically be proposed based on identical naming, yet have to be confirmed by the user. He or she also has to declare several superclass relations to be the same.

Both aspects become more severe, if there is a big number of class diagrams in both modeling teams, probably stored with different techniques. Moreover, examples in this paper are small compared with real class diagrams. Thus the proportion of matching (i.e. overlap specification) of non-relevant data (being outside the fragment that matters for the constraint) will be significantly bigger than in our examples.

### 3.2 Individually Local Checking

Thus the question arises whether there is a more efficient technique for checking inter-metamodel constraints: In the example, a better approach would be to consider a priori only those pieces of data and models and their overlaps that matter for checking.

We propose to pursue the objective of checking validity of constraints *as local as possible*. In the binary case, a pair  $(\tau_1, \tau_2) \in Mod(M_1) \times Mod(M_2)$  is given. We call  $(\tau_1, \tau_2)$  an *incomplete* multimodel over  $(r_1, r_2)$ , because the two components are not yet interrelated (there is no  $\tau_0$  and no arrows  $a_1, a_2$  between domains of the models so far).

**Algorithm for Local Checking** Let incomplete multimodel  $(\tau_1, \tau_2)$  be given over multimetamodel  $(r_1, r_2)$ . An inter-metamodel constraint  $c@\delta$  shall be verified as follows:

1. Identify those fragments of  $M_1$ ,  $M_2$ , and  $M_0$ , that matter for checking, by constructing pullbacks of  $\delta$  along the diagonal in the metamodel pushout, i.e.  $k_1 := \bar{r}_1^*(\delta)$ ,  $k_2 := \bar{r}_2^*(\delta)$ , and  $k_0 := (\bar{r}_1 \circ r_1)^*(\delta)$ . Let  $S_1^c$ ,  $S_2^c$ , and  $S_0^c$  be the domains of these morphisms.
2. Construct pullbacks  $\tau_i^c := k_i^*(\tau_i)$  for  $i \in \{1, 2\}$ . Call the domains (i.e. locally restricted models)  $A_1^c$ , and  $A_2^c$ . This yields arrows  $k'_i : A_i^c \rightarrow A_i$ ,  $i \in \{1, 2\}$ .  
Provide the user (modeler) with the images  $k'_1(A_1^c)$  and  $k'_2(A_2^c)$ , resp. cf. Fact 8, Sect.2.3.
3. Determine compatibly typed overlap  $\hat{\tau}_0^c : \hat{A}_0^c \rightarrow k_0(S_0^c)$  of these two images such that one of the correspondence maps  $\hat{a}_1 : \hat{A}_0^c \rightarrow k'_1(A_1^c)$  and  $\hat{a}_2 : \hat{A}_0^c \rightarrow k'_2(A_2^c)$  is a monomorphism.<sup>4</sup>

<sup>4</sup> This is no serious restriction, as in almost many examples *both* correspondences are injective.

4. Compute pullback  $\tau_0^c : A_0^c \rightarrow S_0^c$  of this overlap along the epi-part of  $k_0$  and apply  $validate_c(\tau_1^c + \tau_0^c \tau_2^c)$ .

In the sequel, we will show that all these constructions neatly fit together, i.e.

- The precise calculation of  $k'_i(A_i^c)$  in Step 2 has to be clarified.
- A complete overlap  $\tau_0 : A_0 \rightarrow M_0$  can automatically be determined and thus be compared with the global approach.
- There are unique arrows  $u_i : A_0^c \rightarrow A_i^c$ ,  $i \in \{1, 2\}$ , such that  $\tau_1^c + \tau_0^c \tau_2^c$  can automatically be determined.
- $\tau_1^c + \tau_0^c \tau_2^c$  is typed over  $S^c$ , hence  $VALIDATE_c$  can be applied.

We first illustrate application of the algorithm for our running example. Fig.6 shows the situation from

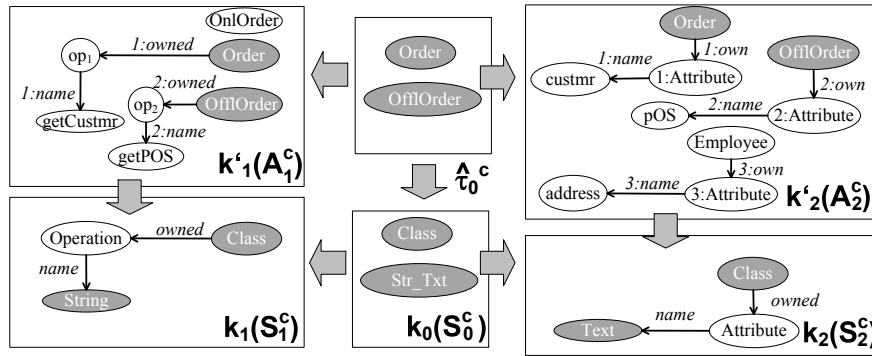


Figure 6: Individually Local Consistency Checking: Steps 1 to 3

the modeler's perspective.

**Step 1** : The images of  $\delta$  (more precise of  $\delta$ 's restrictions  $k_1, k_0, k_2$ ) of  $S_1^c, S_2^c$ , and  $S_0^c$  are depicted in the lower half. Once the complete overlap  $M_0$  is known, they are automatically derived from the scope of the constraint. Shaded vertices again depict overlap. The important improvement is that *Interfaces* together with their operations and interface implementations now vanish. In the same way, class membership of operations can be omitted. Since the constraint declaration does not involve superclass relations, they can be omitted, too. Moreover, we must not care about associations and their source and targets.

**Step 2** : The upper half shows images of appropriately narrowed  $A_1$  and  $A_2$ . Again, this step can be carried out automatically. Note that *OnlPurchaseOrder* is omitted since it does not possess own attributes and hence automatically satisfies constraint declaration  $[acc]@\delta$ .

**Step 3** : The only manual activity is overlap specification. This is shown in the top middle of the figure. It is now reduced to the selection of classes *Order* and *OfflOrder*. We do not have to deal with superclass relations and classes without attributes in the overlap. Model structures simplify accordingly. Moreover, declaration of *OnlOrder-OnlPurchaseOrder-identity* is no longer necessary.

**Step 4** : The pullback of  $\hat{\tau}_0^c$  along the epi-part of  $k_0$  yields the appropriately retyped elements (classes are retyped to type 0 in the arity shape). Pushout calculation is again an automatic procedure. In Fig.7 we depict again the user’s view, i.e. images under  $\delta$  in order to make the localization clear. The resulting model now contains no superfluous elements. It is reduced to four involved classes only: *OnlOrder* still appears (but now only as automatic leftover from  $A_1$ ). The other three classes can easily be traversed. One detects satisfaction for classes *Order* and *OffOrder* (green coloured rectangles). However, the constraint is violated for class *Employee* (red rectangle with “?”).

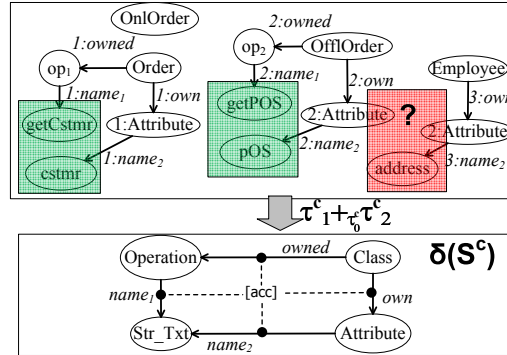


Figure 7: Local consistency checking: Step 4

The reader may compare the unstructured contents of Fig.5 with the reduced data in the upper half of Fig.6. The presented technique obviously reduces workload and manual activities significantly. It remains to ensure that the algorithm always yields the same result than the global definition (cf. Sect.3.1). The challenge is to compare a single invocation of function CHECK on the huge structure  $\tau_1 + \tau_0 + \tau_2$  with two simultaneous partial invocations.

### 3.3 Global-Local-Equivalence

**Theorem 2 (Local Inter-Metamodel Checking is Correct)** *Let  $(\tau_1, \tau_2)$  be an incomplete multimodel over multimetamodel  $(r_1, r_2)$ , in which  $r_1$  or  $r_2$  is injective. Let  $c@ \delta$  be an inter-metamodel constraint over  $(r_1, r_2)$ . Then from the algorithm in Sect.3.2 a complete multimodel*

$$\tau_1 \xleftarrow{(a_1, r_1)} \tau_0 \xrightarrow{(a_2, r_2)} \tau_2$$

*can universally be derived which satisfies  $c@ \delta$  according to Def. 1, if and only if the algorithm returns true.*

Before we give the precise proof, we provide the reader with an idea, how to organize it: both, the above definition and the invented algorithm contain a merging step: one for the entire metamodel and one for only those parts that matter for the constraint. In the proof of our theorem we compare both approaches by - virtually - carrying them out simultaneously: we will use the fact that these *simultaneous merges* can be controlled with the so-called *Van Kampen Property*, whenever one of the two graph mappings  $r_1$  and  $r_2$  is injective<sup>5</sup>. This exactness property for well-behaved interaction of pullbacks and pushout is formally defined as follows:

<sup>5</sup> All examples in the present paper are such that *both*  $r_1$  and  $r_2$  are injective.

**Definition 3 (Van Kampen Square)** A pushout as indicated in the left part of Fig. 8 is called Van Kampen Square, if in every commutative cube with this pushout in the bottom and with back and left faces pullbacks (see the right part of Fig. 8), the following equivalence holds:

The top face is a pushout if and only if the front and right faces are pullbacks.

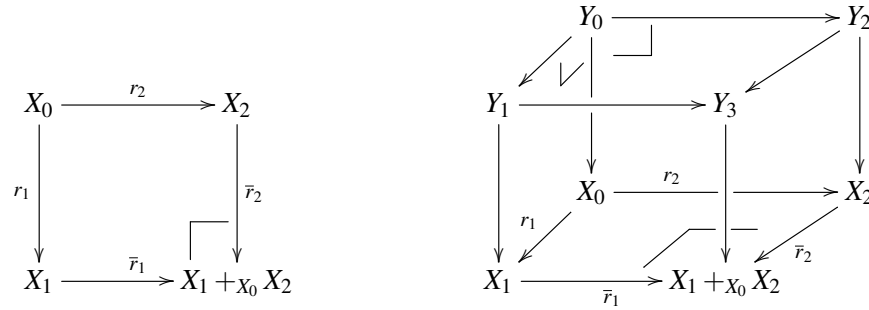


Figure 8: Definition of the Van Kampen Property

We note that the "if"-part in this definition is always fulfilled in  $\mathbb{G}$ , see Sect. 2.3, fact 7.  $\square$

### 3.4 Proof of the Theorem

Additional statements "(a)" to "(d)" will show the points at which the corresponding aspects given after the algorithm in Sect. 3.2 are clarified. The pushout

$$\begin{array}{ccc} M_0 & \xrightarrow{r_2} & M_2 \\ r_1 \downarrow & \lrcorner & \downarrow \bar{r}_2 \\ M_1 & \xrightarrow{\bar{r}_1} & M_1 +_{M_0} M_2 \end{array} \quad (1)$$

is the starting point. The constraint  $c@ \delta$  is recognized in the pushout by a graph morphism  $\delta : S^c \rightarrow M_1 +_{M_0} M_2$ . Step 1 of the algorithm proposes to construct pullbacks of  $\delta$  along the diagonal. This yields  $k_i : S_i^c \rightarrow M_i$ ,  $i \in \{0, 1, 2\}$ , and the commutative cube in Fig.9

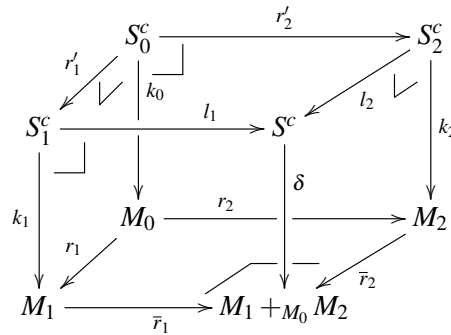


Figure 9: Formula decomposition per metamodel

The first (although obvious) application of Def. 3 is that in the cube in Fig.9 the top face is a pushout, because all 4 side faces are pullbacks. Intuitively this means that  $S^c$  is the union of the projected parts of the sketched area to  $M_1, M_2$ , resp.

Steps 2 and 3 deal with epi-monic-factorisations. In this case

$$S_i^c \begin{array}{c} \xrightarrow{k_i} \\ \xrightarrow{k_i^e} k_i(S_i^c) \xrightarrow{k_i^m} \end{array} M_i$$

$i \in \{0, 1, 2\}$ , see Fact 8. By decomposing pullbacks of  $\tau_i$  along  $k_i = k_i^m \circ k_i^e$  in Step 2 for  $i \in \{1, 2\}$  we obtain epi-monic-factorization of  $k_i^e$  as well, since pullbacks preserve monos as well as epis, cf. Fact 5 (a). Now we define  $A_0 := \hat{A}_0^c$  and  $\tau_0 := k_0^m \circ \hat{\tau}_0^c : A_0 \rightarrow M_0$  (b). Thus

$$\begin{array}{ccc} A_0 & \xlongequal{\quad} & \hat{A}_0^c \\ \tau_0 \downarrow & & \downarrow \hat{\tau}_0^c \\ M_0 & \xleftarrow{k_0^m} & k_0(S_0^c) \end{array}$$

is cartesian. In Step 4  $\tau_0^c = (k_0^e)^*(\hat{\tau}_0^c)$ , thus

$$\tau_0^c \cong k_0^*(\tau_0)$$

We can assume that in Step 4 the calculation was carried out such that equality holds in this relation. Thus we obtain cartesian squares not only for  $i \in \{1, 2\}$  (by Step 2), but also for  $i = 0$ :

$$\begin{array}{ccc} A_i & \xleftarrow{k_i^e} & A_i^c \\ \tau_i \downarrow & & \downarrow \tau_i^c \\ M_i & \xleftarrow{k_i} & M_i^c \end{array} \quad (2)$$

Let

$$a_i := (k_i^e)^m \circ \hat{a}_i, i \in \{1, 2\}. \quad (3)$$

where  $(k_i^e)^m$  are the monic parts of  $k_i^e$ , resp. Via the involved epi-mono-factorisations one computes

$$\tau_i \circ a_i = r_i \circ \tau_0, i \in \{1, 2\}.$$

completing the multimodel. Thus  $\tau_1 \circ a_1 \circ k_0^e = r_1 \circ \tau_0 \circ k_0^e = r_1 \circ k_0 \circ \tau_0^c = k_1 \circ r_1' \circ \tau_0^c$  (cf. (2) and Fig.9) there is a unique arrow  $u_1 : A_0^c \rightarrow A_1^c$  (c) yielding

1.  $\tau_1^c \circ u_1 = r_1' \circ \tau_0^c$
2.  $k_1^e \circ u_1 = a_1 \circ k_0^e$

In the same way a mediating arrow  $u_2 : A_0^c \rightarrow A_2^c$  is obtained, which together with  $u_1$  yields the local correspondence span

$$\tau_1^c \xleftarrow{(u_1, r_1')} \tau_0^c \xrightarrow{(u_2, r_2')} \tau_2^c. \quad (4)$$

By a previous remark  $S^c$  is the "union" of the projected parts, hence the codomain of the pushout (in  $\mathbb{G}^{\rightarrow}$ ) of this span can be taken to be  $S^c$  (d). Let's abbreviate this pushout  $\tau^c := \tau_1^c +_{\tau_0^c} \tau_2^c : A_1^c +_{A_0^c} A_2^c =: A^c \rightarrow S^c$ .

We now construct a cartesian square of the form

$$\begin{array}{ccc} A^c & \xrightarrow{\tau^c} & S^c \\ \downarrow \lrcorner & & \downarrow \delta \\ A_1 +_{A_0} A_2 & \xrightarrow{\tau_1 +_{\tau_0} \tau_2} & M_1 +_{M_0} M_2 \end{array} \quad (5)$$

The proof will then be complete, since this gives  $check(\tau_1 +_{\tau_0} \tau_2, c@ \delta) = validate_c(\tau^c)$ , i.e.  $\tau_1 +_{\tau_0} \tau_2 \models c@ \delta$  if and only if the local check returns true, as required.

For this, we first observe that the equations 1. and 2. make

$$\begin{array}{ccc} \tau_0 & \xleftarrow{(k'_0, k_0)} & \tau_0^c \\ (a_1, r_1) \downarrow & & \downarrow (u_1, r'_1) \\ \tau_1 & \xleftarrow{(k'_1, k_1)} & \tau_1^c \end{array} \quad (6)$$

a commutative diagram in  $\mathbb{G}^{\rightarrow}$ . The arrows  $k_1, r_1, k_0, r'_1$  between codomains of the four  $\mathbb{G}^{\rightarrow}$ -objects in the corners of this square constitute the left face pullback of Fig.9 such that, by pullback composition and decomposition

$$\begin{array}{ccc} A_0 & \xleftarrow{k'_0} & A_0^c \\ a_1 \downarrow & & \downarrow u_1 \\ A_1 & \xleftarrow{k'_1} & A_1^c \end{array}$$

is cartesian. Replacing index 1 by 2 in this argumentation shows that

$$\begin{array}{ccc} A_0 & \xleftarrow{k'_0} & A_0^c \\ a_2 \downarrow & & \downarrow u_2 \\ A_2 & \xleftarrow{k'_2} & A_2^c \end{array}$$

is cartesian, as well. Hence in

$$\begin{array}{ccccc} & & A_0^c & \xrightarrow{u_2} & A_2^c \\ & u_1 \swarrow & \lrcorner & \searrow \bar{u}_2 & \downarrow k'_2 \\ A_1^c & \xrightarrow{\bar{u}_1} & A^c & & \\ \downarrow k'_1 & & \downarrow k'_0 & & \downarrow k^c \\ & a_1 \swarrow & A_0 & \xrightarrow{a_2} & A_2 \\ & & \lrcorner & \searrow \bar{a}_2 & \\ A_1 & \xrightarrow{\bar{a}_1} & A_1 +_{A_0} A_2 & & \end{array} \quad (7)$$

the back faces are pullbacks. Since  $A^c$  was constructed as pushout, we obtain a mediator  $k^c : A^c \rightarrow A_1 +_{A_0} A_2$  (dashed arrow in (7)).



Since one of  $\hat{a}_i$  is monic (Step 3),  $a_1$  or  $a_2$  is monic, as well, cf. (3). Thus the involved pushout square

$$\begin{array}{ccc} A_0 & \xrightarrow{a_2} & A_2 \\ a_1 \downarrow & & \downarrow \bar{a}_2 \\ A_1 & \xrightarrow[\bar{a}_1]{} & A_1 +_{A_0} A_2 \end{array}$$

has the Van Kampen property such that Definition 3 ensures that both front faces in (7) are pullbacks. In the commutative "cube"

$$\begin{array}{ccc} \tau_i^c & \xrightarrow{(\bar{u}_i, l_i)} & \tau^c \\ (k'_i, k_i) \downarrow & & \downarrow (k^c, \delta) \\ \tau_i & \xrightarrow[\bar{a}_i, \bar{r}_i]{} & \tau_1 +_{\tau_0} \tau_2 \end{array} \quad (8)$$

this observation shows that the left part of

$$\begin{array}{ccccc} A_i^c & \xrightarrow{\bar{u}_i} & A^c & \xrightarrow{\tau^c} & S^c \\ k'_i \downarrow \lrcorner & & \downarrow k_c & & \downarrow \delta \\ A_i & \xrightarrow[\bar{a}_i]{} & A_1 +_{A_0} A_2 & \xrightarrow[\tau_1 +_{\tau_0} \tau_2]{} & M_1 +_{M_0} M_2 \end{array}$$

is cartesian and that the complete rectangle is a pullback, too, for  $i \in \{1, 2\}$  (again pullback (de-)composition). By fact 5 and Lemma 4 (stated below) the same situation occurs in

$$\begin{array}{ccccc} A_1^c + A_2^c & \xrightarrow{[\bar{u}_1, \bar{u}_2]} & A^c & \xrightarrow{\tau^c} & S^c \\ k'_1 + k'_2 \downarrow \lrcorner & & \downarrow k_c & & \downarrow \delta \\ A_1 + A_2 & \xrightarrow[\bar{a}_1, \bar{a}_2]{} & A_1 +_{A_0} A_2 & \xrightarrow[\tau_1 +_{\tau_0} \tau_2]{} & M_1 +_{M_0} M_2 \end{array}$$

The most important fact, however, is that  $[\bar{a}_1, \bar{a}_2]$  is epic because it is the cocone of a pushout, s.th. Fact 6 yields the desired result.  $\square$

**Lemma 4** *Let in a category with coproducts*

$$\begin{array}{ccccc} A & & & & \\ & \searrow f & & & \\ & & A + A' & \xrightarrow{g} & B \xrightarrow{g} C \\ & \nearrow i & & & \\ A' & & & & \\ & \searrow i' & & & \\ & & & & \\ & \nearrow f' & & & \end{array}$$

be given where  $i, i'$  are coproduct injections. Then

$$g \circ [f, f'] = [g \circ f, g \circ f'].$$

*Proof:*  $u := [g \circ f, g \circ f']$  is unique with the property  $u \circ i = g \circ f$  and  $u \circ i' = g \circ f'$ . But  $g \circ [f, f']$  fulfills this property, too.  $\square$

## 4 Future Work

We presented a new approach for individually local checking of constraints imposed on heterogeneous multimodels, which significantly reduces workload and error-prone manual interaction. Our second contribution is a formal underpinning of global consistency, which essentially employs the diagrammatic nature of constraint; in this framework, we were able to prove the global-local-equivalence formally.

The most important direction for future research is to generalize the proposed algorithm together with the necessary correctness theorem for the general N-ary case with complex overlapping considered in [7]. This encompasses non-monic legs in relation spans. Moreover, view definitions (on metamodels) and view execution (on models)[5] should be taken into consideration. The challenge will be to find appropriate generalization and extensions of the mathematical machinery, which, at least to our knowledge, have not yet been provided.

Another direction of future research is to extend the scope of underlying graphical structures: Up to now, our approach is formally underpinned by simple directed graphs, but more general structures have also been considered, e.g. attributed graphs [8]. Obviously this also requires a generalization of the underlying diagrammatic framework.

Last but not least, we plan to evaluate the algorithm in the tooling framework developed at Bergen University College [15, 17]. Our idea is to enhance the DPF editors to make them inter-metamodel aware. Alternatively, we can try to integrate our approach into other constraint checking tools, e.g. USE, a tool to specify additional integrity constraints on models which has the ability to check system state snapshots against OCL constraints [12].

## References

- [1] A. Carboni, G. Janelidze, G.M. Kelly & R. Paré (1997): *Localization and stabilization for factorization systems*. *Applied Categorical Structures* 5(1), pp. 1–58.
- [2] Z. Diskin (2011): *Towards generic formal semantics for consistency of heterogeneous multimodels*. Technical Report GSDLAB 2011-02-01, University of Waterloo.
- [3] Z. Diskin & U. Wolter (2008): *A Diagrammatic Logic for Object-Oriented Visual Modeling*. *Electr. Notes Theor. Comput. Sci.* 203(6), pp. 19–41, doi:10.1016/j.entcs.2008.10.041.
- [4] Zinovy Diskin, Boris Kadish, Frank Piessens & Michael Johnson (2000): *Universal Arrow Foundations for Visual Modeling*. In Michael Anderson, Peter Cheng & Volker Haarslev, editors: *Diagrams, Lecture Notes in Computer Science* 1889, Springer, pp. 345–360. Available at <http://link.springer.de/link/service/series/0558/bibs/1889/18890345.htm>.
- [5] Zinovy Diskin, Tom Maibaum & Krzysztof Czarnecki (2012): *Intermodeling, queries, and kleisli categories*. In: *Fundamental Approaches to Software Engineering*, Springer, pp. 163–177.
- [6] Zinovy Diskin & Uwe Wolter (2007): *A Diagrammatic Logic for Object-Oriented Visual Modeling*. In: *Proceedings of the Second Workshop on Applied and Computational Category Theory (ACCAT 2007)*, ENTCS, pp. 19–41, doi:10.1016/j.entcs.2008.10.041.
- [7] Zinovy Diskin, Yingfei Xiong & Krzysztof Czarnecki (2011): *Specifying Overlaps of Heterogeneous Models for Global Consistency Checking*. In Juergen Dingel & Arnor Solberg, editors: *Models in Software Engineering, Lecture Notes in Computer Science* 6627, Springer Berlin Heidelberg, pp. 165–179, doi:10.1007/978-3-642-21210-9\_16. Available at [http://dx.doi.org/10.1007/978-3-642-21210-9\\_16](http://dx.doi.org/10.1007/978-3-642-21210-9_16).
- [8] H. Ehrig, K. Ehrig, U. Prange & G. Taentzer (2006): *Fundamentals of Algebraic Graph Transformations*. Springer.

- [9] S. Fickas, M. Feather & J. Kramer (1997): *Workshop on Living with Inconsistency*. Proceedings of ICSE-97, Boston, USA.
- [10] Martin Fowler (1997): *Analysis Patterns: Reusable Object Models*. Addison-Wesley.
- [11] Peter Freyd (1972): *Aspects of Topoi*. *Bull. Austral. Math. Soc.* 7, pp. 1–76, doi:10.1017/S0004972700044828.
- [12] Martin Gogolla, Fabian Büttner & Mark Richters (2007): *USE: A UML-based specification environment for validating UML and OCL*. *Sci. Comput. Program.* 69(1-3), pp. 27–34, doi:10.1016/j.scico.2007.01.013. Available at <http://dx.doi.org/10.1016/j.scico.2007.01.013>.
- [13] Robert Goldblatt (1984): *Topoi: The Categorical Analysis of Logic*. Dover Publications.
- [14] S. Lack & P. Sobociński (2006): *Toposes are Adhesive*. *Lecture Notes in Comput. Sci.* 4178, pp. 184–198, doi:10.1007/11841883\_14.
- [15] Yngve Lamo, Xiaoliang Wang, Florian Mantz, Øyvind Bech, Anders Sandven & Adrian Rutle (2013): *DPF Workbench: A multi-level language workbench for MDE*. *Proc. of the Estonian Acad. of Sciences* 62, pp. 3–15, doi:10.3176/proc.2013.1.02.
- [16] Adrian Rutle, Alessandro Rossini, Yngve Lamo & Uwe Wolter (2009): *A Diagrammatic Formalisation of MOF-Based Modelling Languages*. In Manuel Oriol & Bertrand Meyer, editors: *TOOLS EUROPE, Lecture Notes in Business Information Processing* 33, Springer, pp. 37–56, doi:10.1007/978-3-642-02571-6\_4. Available at [http://dx.doi.org/10.1007/978-3-642-02571-6\\_4](http://dx.doi.org/10.1007/978-3-642-02571-6_4).
- [17] Adrian Rutle, Uwe Wolter & Yngve Lamo (2008): *A Diagrammatic Approach to Model Transformations*. In: *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems (EATIS 2008)*, ACM, pp. 1–8, doi:10.1145/1621087.1621105.
- [18] Mehrdad Sabetzadeh, Shiva Nejati, Sotirios Liaskos, Steve M. Easterbrook & Marsha Chechik (2007): *Consistency Checking of Conceptual Models via Model Merging*. In: *RE, IEEE*, pp. 221–230. Available at <http://dx.doi.org/10.1109/RE.2007.18>.
- [19] U. Wolter & H. König (2015): *Fibred Amalgamation, Descent Data, and Van Kampen Squares in Topoi*. *Applied Categorical Structures* 23(3), pp. 447 – 486, doi:10.1007/s10485-013-9339-2.







UNIVERSITY OF APPLIED SCIENCES

**FHDW**

FACHHOCHSCHULE FÜR DIE WIRTSCHAFT  
HANNOVER

ISSN 1863-7043

