A Diagrammatic Approach to Model Completion

Fazle Rabbi
Bergen University College
University of Oslo
Fazle.Rabbi@hib.no

Yngve Lamo
Bergen University College
Yngve.Lamo@hib.no

Ingrid Chieh Yu
University of Oslo
ingridcy@ifi.uio.no

Lars Michael Kristensen
Bergen University College
Lars.Michael.Kristensen@hib.no

Abstract

Metamodelling plays an important role in model-driven engineering as it can be used to define domain-specific modelling languages. During the modelling phase, software designers encode domain knowledge into models which may include both structural and behavioural aspects of a system. In this paper we propose a diagrammatic approach to aid the software designer to complete partial models and thereby reduce modelling effort. We introduce a declarative approach where we define completion rules that are executed by the use of model transformations. We also study the termination of such model transformation systems and provide sufficient conditions for termination by generalizing existing work on termination of model transformation systems.

1 Introduction

Model Driven Engineering (MDE) [Sch06] is considered to be an efficient way of improving the quality of software and enhance software development productivity by automating repetitive, error-prone, and time-consuming tasks. In MDE, models are first–class artefacts of the software development process where models are incrementally refined starting from requirements and eventually used for code generation. Metamodels serve as the underlying foundation defining the modelling languages used to capture domain-specific knowledge and concepts.

In the process of development, software designers are often confronted with a variety of inconsistencies and/or incompleteness in the models under construction [MVDS07]. In particular, the modeller will most of the time be working with a *partial model* not conforming (i.e., not being typed by and satisfying modelling constraints) to the metamodel that defines the modelling language being used [SBP07]. As an example, consider the following example from the healthcare domain. Figure 1(a) shows a metamodel $M2$ (in this case a class diagram) and Figure 1(b) shows a partial model (in this case an object diagram) $M1$ typed by $M2$. The typing of the nodes are represented by dotted arcs. In this example, Bryan works at Ward10; Ward10 is controlled by the Emergency department. The model $M1$ is a partial model as it is not satisfying the following domain constraint: "An employee who is involved in a ward must work in the controlling department". This constraint is defined in the metamodel $M2$ by the OCL constraint shown to the upper right at Figure 1. Since Ward10 is controlled by the Emergency department, Bryan must work at the Emergency department.

In the above example, the partial model $M1$ can be transformed into a model conforming to the metamodel $M2$ by adding an arc from the Emergency department to Bryan which is missing in $M1$. Figure 1(c) shows a model $M1^*$ that conforms to the metamodel $M2$. Clearly, the productivity of the modeller could be improved by providing editing support that could either automatically add such missing model elements or make suggestions

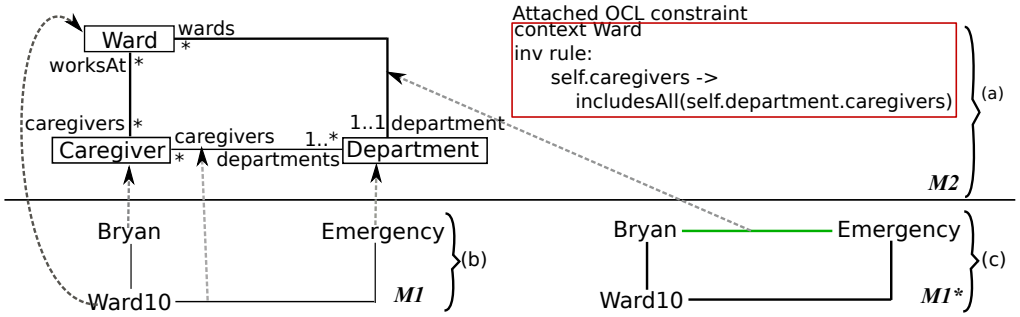


Figure 1: (a) Model $M2$, (b) a partial model $M1$ (not conforming to $M2$), (c) a completed model $M1^*$ (conforming to $M2$)

based on *completion rules* to assist the modeller in completing the model. In many respects, this idea is similar to code completion features as found in IDEs. More generally, modelling effort could be reduced by providing editing support for automated rewriting of models so that they conform to the modelling language used. Such rewriting may also involve the deletion of model elements.

The contribution of this paper is a framework for rewriting partial (incomplete) models so that they conform to the underlying metamodel. Our approach is based on rewriting of models by means of model transformation rules which supports both addition and deletion of model elements. We refer to them as *completion rules*. The proposed framework has been developed as an extension of the Diagram Predicate Framework (DPF) [LWM+12] which supports multilevel metamodelling. This approach is, however, general and could be used for other modelling frameworks as well. DPF is a language independent formalism for defining metamodelling hierarchies which provides an abstract visualization of concrete constraints. In the extended DPF, one can graphically specify completion rules. Our framework exploits the locality of model transformation rules and provides a foundation that enables automated tool-support to increase modelling productivity. In order to guarantee termination of the rewriting, we provide a set of sufficient termination criteria. A link to the prototype implementation of the proposed framework is available at *http://dpf.hib.no* where one can graphically design modelling artefacts in a web browser, apply the completion rules and check for termination.

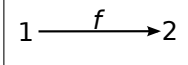
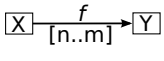
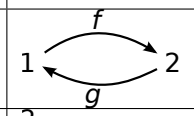
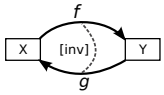
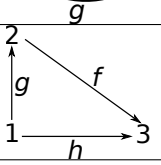
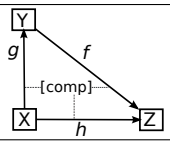
The rest of this paper is organised as follows. Section 2 provides background knowledge on DPF, section 3 provides an outline of the partial model completion with an example, section 4 formalizes the concept of diagrammatic rewriting systems, and section 5 presents sufficient termination criteria. In section 6, we briefly discuss some related work. We assume that the reader is familiar with basic category theory and graph transformation systems[BW95], [EEPT06].

2 Diagrammatic Modelling with DPF

In order to work with partial models and completion rules in a formal framework; we extend the Diagram Predicate Framework (DPF) [Rut10]. DPF provides a formal diagrammatic approach to metamodelling based on category theory and model transformations. In the DPF approach, models at any level are formalised as diagrammatic specifications which consist of graphs and diagrammatic constraints. Figure 2 shows an example of a DPF model (specification) $\mathfrak{S}_1$ with its metamodel (specification) $\mathfrak{S}_0$. The graphs represent the structure of the models; constraints are added into the structure by predicates. The metamodel (specification) $\mathfrak{S}_0$ provides the typing for the model (specification) $\mathfrak{S}_1$. Nodes in model (specification) $\mathfrak{S}_1$ are either of type Class or DataType; edges in $\mathfrak{S}_1$ are either of type Reference or Attribute. Table 1 shows a list of predicates used for constraining the specification $\mathfrak{S}_1$. Each predicate has a name (p), a shape graph ($\alpha(p)$), a visualisation and a semantic interpretation. The semantic of a predicate is provided by a set of instances. The (atomic) constraining constructs which are available for the users of the modelling language are provided in the signature Σ_i. A signature consists of a collection of diagrammatic predicates. Table 2 shows how the predicates are constraining the specification $\mathfrak{S}_1$ by a graph homomorphism $\delta : \alpha(p) \to S$ from the shape graph to the specification.

In DPF, a modelling language is formalised as a modelling formalism $(\Sigma_i, \mathfrak{S}_i, \Sigma_{i-1})$ where i and $i-1$ represent two adjacent modelling levels. The corresponding metamodel of the modelling language is represented by the specification $\mathfrak{S}_i$ which has its constraints formulated by predicates from the signature Σ_{i-1}. A DPF metamodelling hierarchy consists of (a possible stack of) metamodels, models, and instances of models. A metamodel specification determines a modelling language, the specification of a model represents a software system, and

Table 1: Predicates of a sample signature Σ_0

p	$\alpha^{\Sigma_0}(p)$	visualisation	Semantic Interpretation
[mult(n,m)]			f must have at least n and at most m instances. Formally, $\forall x \in X : m \leq \lvert f(x)\rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
[inverse]			For each instance of f there exists an instance of g or vice versa. Formally, $\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[composite]			For each instance of $(g; f)$, there exists an instance of h. Formally, $\forall x \in X : \bigcup \{f(y) \mid y \in g(x)\} \subseteq h(x)$

the instances of models typically represent possible states of a software system. In this paper, we present a new type of DPF construct named *completion rules* based on model transformation rules. The completion rules are connected to (domain) constraints.

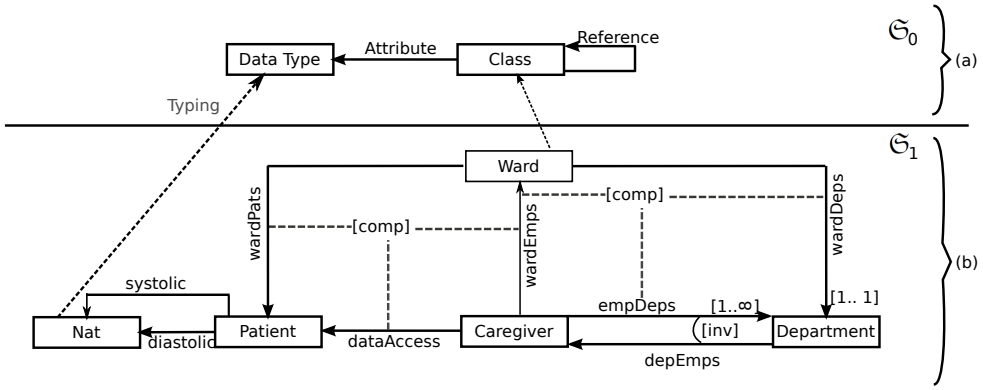


Figure 2: a) Metamodel $\mathfrak{S}_0$, b) Model $\mathfrak{S}_1$

3 Model Completion Example

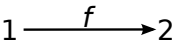
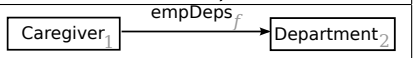
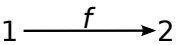
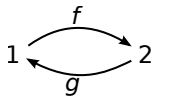
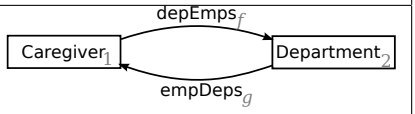
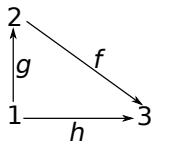
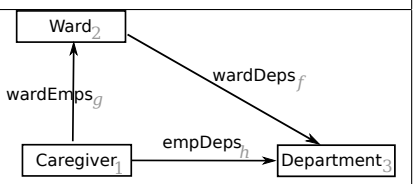
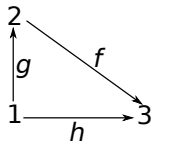
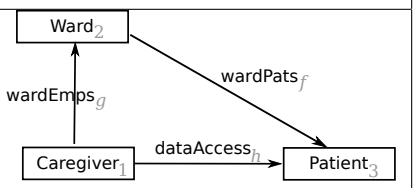
The completion rules defined in a metamodel are used to complete a partial model. We introduce requirement R1-R5 for an envisioned system from the healthcare domain. These requirements specify the constraints of a system. Completion rules are used to automatically complete a partial model not satisfying these constraints.

R1. *A Caregiver (e.g., nurse, doctor) must work for at least one Department.*
R2. *A Caregiver may work for more than one Department.*
R3. *A Ward must be controlled by exactly one Department.*
R4. *A Caregiver who is involved in a Ward, must work for its controlling Department.*
R5. *All Caregivers assigned to a Ward have access to the patients information who are admitted to the same Ward.*

Figure 3(a) shows a DPF model (specification) $\mathfrak{S}_1$ developed from the above requirements. The metamodel (specification) of $\mathfrak{S}_1$ was shown in Figure 2(a). In $\mathfrak{S}_1$ we used the signature Σ_0 from Table 1 to define the set $C^{\mathfrak{S}_1}$ of atomic constraints. R1 and R2 is encoded in $\mathfrak{S}_1$ by the [*mult*$(1, \infty)$] predicate on the edge *empDeps*; R3 by the [*mult*$(1, 1)$] predicate on the edge *wardDeps*; R4 by the [*composite*] predicate on edges *wardEmps*, *wardDeps* and *empDeps* where (*wardEmps*; *wardDeps*) $\subseteq$ *empDeps*; and R5 by the [*composite*] predicate on edges *wardEmps*, *wardPats* and *dataAccess* where (*wardEmps*; *wardPats*) $\subseteq$ *dataAccess*.

Figure 3(b) shows a candidate instance $\iota : I \to S$ of $\mathfrak{S}_1$ (also represented as (I, ι)) where S is the underlying graph of specification $\mathfrak{S}_1$. Even though I is typed by S, (I, ι) is not a valid instance of $\mathfrak{S}_1$ since it does not satisfy the predicates [*mult*$(1, \infty)$] and [*composite*]. This can be checked from the pullback operation shown in Figure 4.

Table 2: The set of atomic constraints $C^{\mathfrak{S}_1}$ of $\mathfrak{S}_1$

(p, δ)	$\alpha^{\Sigma_0}(p)$	$\delta(\alpha^{\Sigma_0}(p))$
$([mult(1,\infty)], \delta_1)$		
$([mult(1,1)], \delta_2)$		
$([inverse], \delta_4)$		
$([composite], \delta_5)$		
$([composite], \delta_6)$		

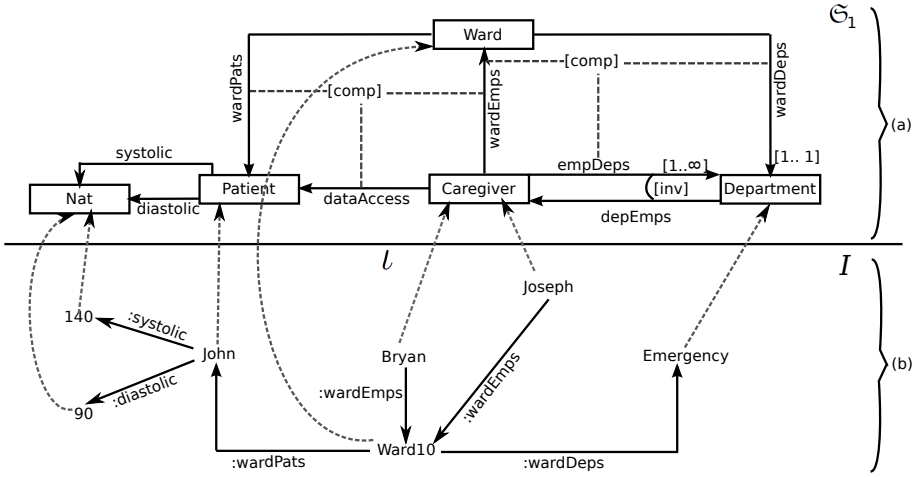


Figure 3: a) A DPF model $\mathfrak{S}_1 = (S, C^{\mathfrak{S}_1} : \Sigma_2)$, and b) an instance (I, ι) of $\mathfrak{S}_1$

By performing a pullback of $\alpha^{\Sigma_0}([composite]) \xrightarrow{\delta_1} S \xleftarrow{\iota} I$, we extract the fragment of the graph that is affected by the [*composite*] constraint. The graph homomorphism ι^* (see Figure 4) is not a valid instance of the [*composite*] as the semantics of the [*composite*] predicate is not satisfied. In this example, Bryan works at Ward10; Ward10 is controlled by the Emergency department. Since Ward10 is controlled by the Emergency department, Bryan must work at the Emergency department. Also, the caregivers are supposed to work for at least one department which is missing in $\iota : I \to S$ of $\mathfrak{S}_1$. Figure 5 shows a screenshot of the editor that highlights which part of the model instance violates the constraints.

In the next section, we extend DPF with model transformation rules which allows us to incorporate model completion rules. We augment the above mentioned example with completion rules at the meta-model level that can be used as a basis for automatically removing the inconsistencies of the partial model (I, ι). Table 3 shows some completion rules linked to predicates. Completion rules for a predicate ensures that the predicate becomes satisfied. The semantics of completion rules are defined by coupled transformation rules [SLK11] [Bec08] with negative application conditions as in Table 3. Negative application conditions are typically used

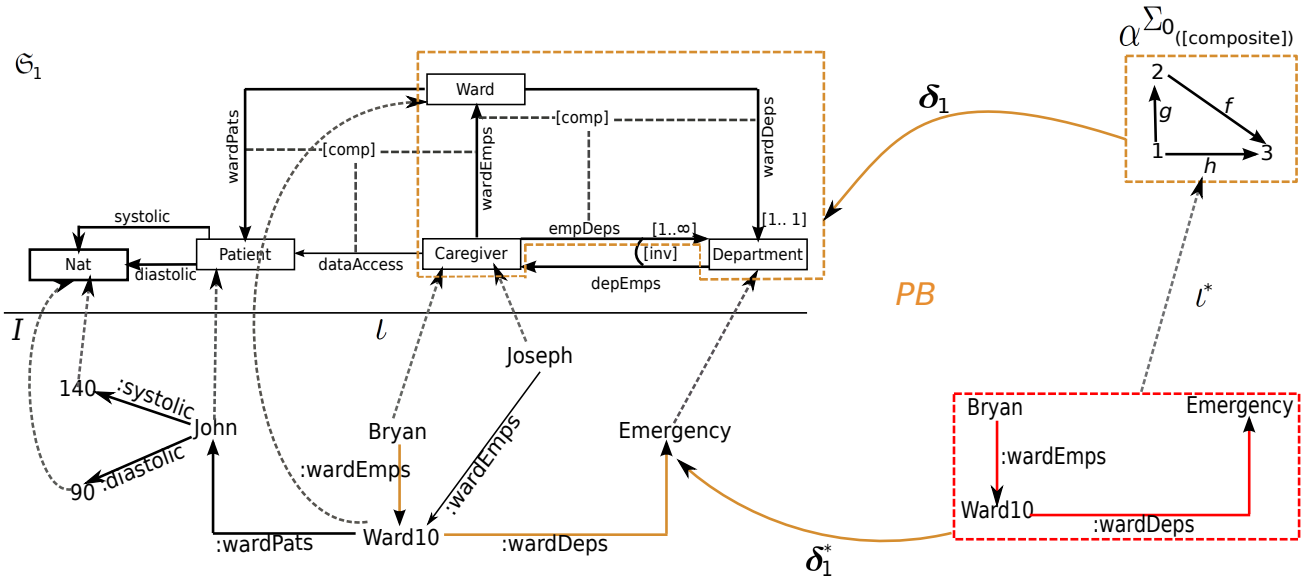


Figure 4: Pullback $\alpha^{\Sigma_0}([composite]) \xleftarrow{\iota^*} O^* \xrightarrow{\delta_1^*} I$ of $\alpha^{\Sigma_0}([composite]) \xrightarrow{\delta_1} S \xleftarrow{\iota} I$

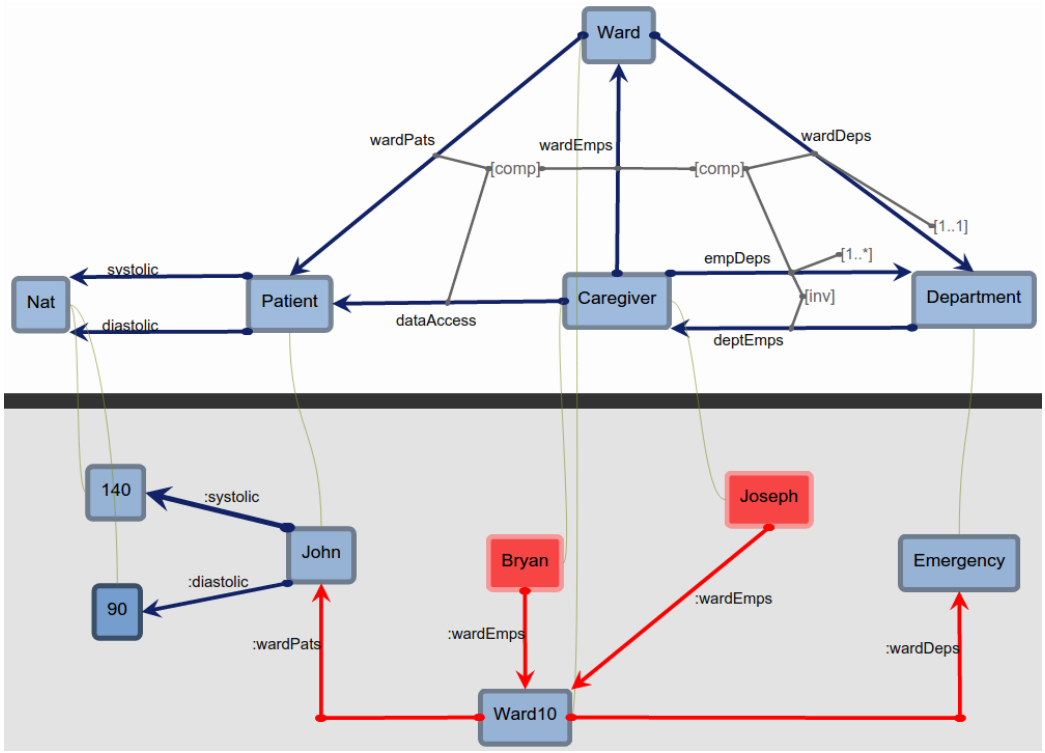
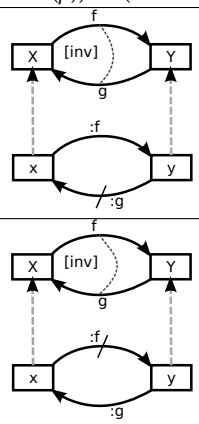
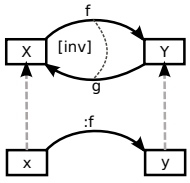
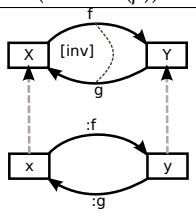
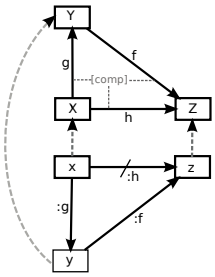
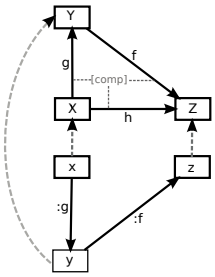
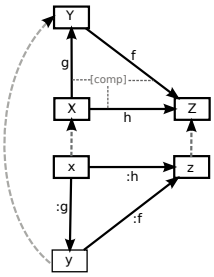


Figure 5: An inconsistent instance (I, ι) (bottom) of $\mathfrak{S}_1$ (top)

in graph transformations to prohibit an infinite number of rule applications. We use a special type of coupled transformations where the metamodels remain unchanged [RMWL12]. Notice that the matching patterns and negative application conditions are represented in the same diagram in Table 3 to make it more readable. Negative application conditions on model elements are represented by a strike through the corresponding model elements in the diagram to represent that they must not exist while matching.

By applying the completion rules from Table 3 on (I, ι), the partial model is updated to become a model, (I^*, ι) which conforms to its metamodel. Once we apply the completion rules for $[inv - com\rangle_{[inv]}$ and $[comp - com\rangle_{[comp]}$

Table 3: Completion scheme for predicates and completion rules of $\mathfrak{S}_1$

C_p	$\zeta(C_p)$			Interpretation
	$(N \to \alpha^{\Sigma}(p)) \leftarrow (L \to \alpha^{\Sigma}(p))$	$(K \to \alpha^{\Sigma}(p))$	$(R \to \alpha^{\Sigma}(p))$	
$[\text{inv-com}\rangle_{[inv]}$				derive an edge $y \xrightarrow{:g} x$ (if it does not exist) from the existence of an edge $x \xrightarrow{:f} y$ or vice versa.
$[\text{comp-com}\rangle_{[comp]}$				derive an edge $x \xrightarrow{:h} z$ (if it does not exist) from the existence of edges $x \xrightarrow{:g} y$ and $y \xrightarrow{:f} z$.

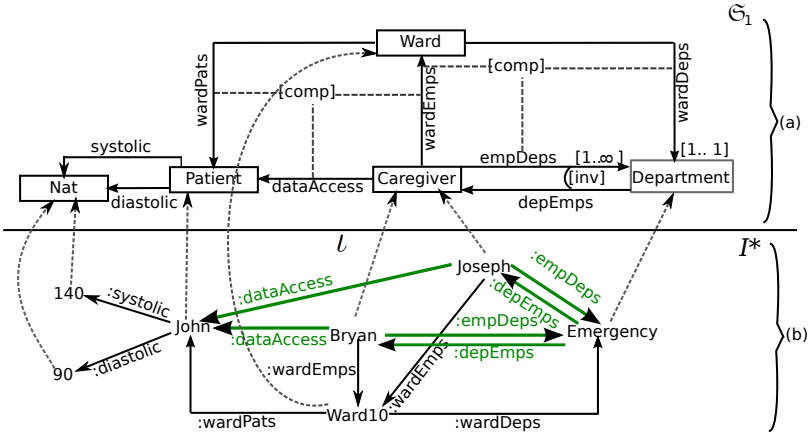


Figure 6: a) A specification $\mathfrak{S}_1 = (S, C^{\mathfrak{S}_1} : \Sigma_0)$ and b) an instance (I^*, ι) of $\mathfrak{S}_1$

on (I, ι), we obtain the instance of specification $\mathfrak{S}_1$ shown in Figure 6. Bold arcs (also shown with a green color) in Figure 6 represent the additional edges derived from the application of completion rules. We do not have deleting rules in this example, but in general it is possible to have deletion rules to repair e.g., multiplicity constraints [TMAL13].

4 Diagrammatic Model Completion

Our diagrammatic rewriting system is based on completion rules. Completion rules are typed coupled transformation rules where type graphs are not changed by the transformation. The rules are linked to predicates and they are applied to a partial model to correct inconsistencies. We use the standard double-pushout (DPO) approach [EEPT06] for defining completion rules. In this section we give a formal definition of completion rules

and formalize its application and matching.

Definition 1 (Completion scheme of a predicate and completion rules) *Let $\Sigma = (P^\Sigma, \alpha^\Sigma)$ be a signature. A completion scheme C_p for a predicate $p \in P^\Sigma$, is given by a symbol cp, and a set of completion rules $\zeta(C_p)$ where the meta-models remain unchanged. A rule $r \in \zeta(C_p)$ of a completion scheme of a predicate C_p has a matching pattern (L), a gluing condition (K), a replacement pattern (R), and an optional negative application condition, NAC $(n : L \to N)$ where L, N, K, R are all typed by the arity $\alpha^\Sigma(p)$ of the predicate p.*

A completion rule as defined in Definition 1 is illustrated in the following figure. The matching pattern and replacement pattern are also known as left-hand side and right-hand side of a rule, respectively. In the figure, k and l are injective morphisms. A completion rule is nondeleting if $K = L$. In this case we have an injective morphism $L \hookrightarrow R$.

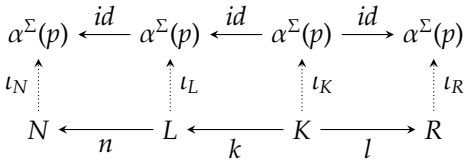


Figure 7 illustrates a rule associated with the [*composite*] predicate where the graphs L, N, K, R are all typed by the arity of the [*composite*] predicate.

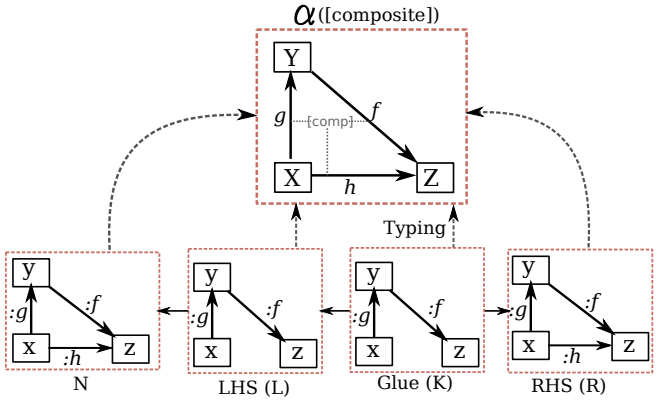
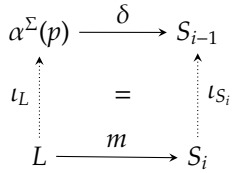


Figure 7: A transformation rule linked to the [*composite*] predicate

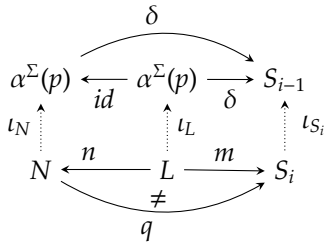
Since the rules are not directly linked to a model, they can be reused by constraining a model with appropriate predicates.

Definition 2 (Match of a completion rule). *Let C_p be a completion scheme of a predicate with a set of completion rules $\zeta(C_p)$. A match (δ, m) of a rule $r \in \zeta(C_p)$ is given by an atomic constraint $\delta : \alpha^\Sigma(p) \to S_{i-1}$ and a match $m : L \to S_i$ such that constraint δ and match m together with typing morphisms $\iota_L : L \to \alpha^\Sigma(p)$ and $\iota_{S_i} : S_i \to S_{i-1}$ constitute a commuting square: $\iota_L; \delta = m; \iota_{S_i}$ as in the diagram below.*

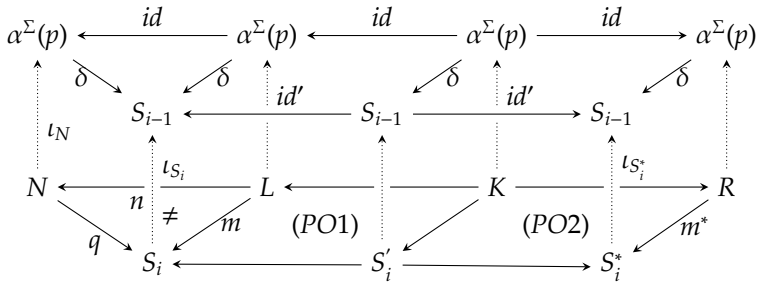


Now we consider the satisfaction of a negative application condition of a completion rule. A rule is applied as long as it satisfies its negative application condition which is typically expressed by a graph structure in a graph transformation system.

Definition 3 (Satisfaction of a NAC of a completion rule) *A match (δ, m) of a rule $r \in \zeta(C_p)$ satisfies a NAC $(n : L \to N)$ if there does not exist an injective graph morphism $q : N \to S_i$ with $n; q = m$ such that the typing morphisms $\iota_N : N \to \alpha^\Sigma(p)$ and $\iota_{S_i} : S_i \to S_{i-1}$ constitute a commuting square: $\iota_N; \delta = q; \iota_{S_i}$ as shown in the diagram below. This is written $(\delta, m) \models NAC$.*



Definition 4 (Application of a completion rule) *Let C_p be a completion scheme of a predicate with a set of completion rules $\zeta(C_p)$. A rule $r = ((N \to \alpha^{\Sigma}(p)) \leftarrow (L \to \alpha^{\Sigma}(p)) \leftarrow (K \to \alpha^{\Sigma}(p)) \to (R \to \alpha^{\Sigma}(p))) \in \zeta(C_p)$ transforms a partial model (S_i, ι_{S_i}) to $(S_i^*, \iota_{S_i^*})$ if there exists a match (δ, m) where $(\delta, m) \models NAC$. The transformation consist of two commuting cubes and two pushout diagrams (PO1 and PO2) as shown below:*



Note that $(S_i^*, \iota_{S_i^*})$ could be a partial model and in that situation, further application of completion rules are required to obtain a a model that satisfies all the constraints. Our framework supports the execution of rules in two different ways: one can interactively apply the rules where the user guides the execution order; or automatically considering all possible orderings of rule execution. In the latter case, it is important to perform an analysis to ensure that the execution of the transformation rules will eventually terminate.

5 Termination Criteria

In this paper we propose a termination analysis based on the principles adapted from layered graph grammars [EEPT06]. In a layered typed graph grammar, transformation rules are distributed across different layers. The transformation rules of a layer are applied as long as possible before going to the next layer. Ehrig et al [EEPT06] distinguished between deletion and nondeletion layers where all transformation rules in deletion layers delete at least one element and all transformation rules in nondeletion layers do not delete any elements. A set of layer conditions was specified in [EEPT06] that need to be satisfied by each layer k to guarantee termination. Transformation rules in nondeletion layers must have negative application conditions. The layer conditions do not permit a rule r to use an element x of a given type t for the match if any element of type t has been created in a layer $k' \leq k$. The layer conditions also imply that the creation layer of an element of type t must precede its deletion layer. This restriction prevents the repetitive application of a particular rule. This layered typed graph grammar approach is suitable for situations where repetitive application of rules are not required. Unfortunately, there are many situations where repetitive application of rules are desirable such as to compute transitive closure operations [LPE07].

We generalize the layer conditions from [EEPT06] allowing deleting and non-deleting rules to reside in the same layer as long as the rules are loop-free. Furthermore, this generalization permits a rule to use newly created edges allowing us to perform transitive closure operations. To achieve this, we rely on a loop detection algorithm that overestimates the existence of a loop from a set of rules. The loop detection algorithm is based on the following sufficient conditions for loop freeness. Let R_k be the set of rules of a layer k.

- If a rule $r_i \in R_k$ creates an element x of type t, then r_i must have an element of type t in its NAC,
- If a rule $r_i \in R_k$ deletes an element of type t, then there is no rule in $r_j \in R_k$ that creates an element of type t

Note that we are assuming, that there is a finite number of rules in each layer and that the rules are applied on a finite input graph. The algorithm guarantees termination if all the rules for each layer satisfies the above

mentioned conditions. A complete specification of the algorithm and a formal proof of correctness is available as technical report on *http://dpf.hib.no/publications*.

6 Related and Future Work

Partial models have been used in MDE for various purposes. It has been used for modelling with uncertainty by Salay et al. in [SFC12]. They proposed four types of partiality that can be defined in a modelling language–independent way. Their definition of metamodel consists of a First Order Logic theory that includes a set of sentences representing the well–formedness constraints. Modelling with uncertainty is essential in situations such as: i) the requirements are not clearly specified, ii) alternative resolution to inconsistency is present, and iii) stakeholders opinions differ. In this article, we focus on precise modelling of systems with metamodelling assuming the requirement is clearly specified.

An approach similar to our work was presented in [SBP07] where the authors presented a methodology to automatically solve partial model completion problems. They transformed a partial model, its metamodel, and additional constraints to a constraint logic program (CLP). The metamodel specifies the domain of possible assignments for each and every property. This information is being used by the CLP to complete a partial model object. They provided an algorithm to generate code that assigns a domain of values to the attributes and relationships. When the CLP is queried, a solver selects values from the domain of each property such that the conjunction of all the constraints is satisfied. In our approach, we use multilevel metamodelling for specifying domain models. We use diagrammatic approach to define completion rules and use model transformations to derive a complete model instance that satisfies the predicate constraints.

An automated approach for detecting and tracking inconsistencies in real time was presented in [Egy11]. The author listed 24 consistency rules that cover the most significant concerns of keeping sequence diagrams consistent with class and statechart diagrams and evaluated the rules on UML design models. While their approach is based on the syntactical constraints of UML diagrams, ours is based on domain specific diagrammatic constraints. In our approach, the completion rules are graphically formulated, customizable, and vary from one domain to another.

The eMoflon tool [ALPS11] is built on the Eclipse Modelling Framework (EMF), using Ecore for metamodelling that supports rule-based unidirectional and bidirectional model transformation. eMoflon is featured with MOF 2.0 compliant code generation and concentrates on bidirectional model transformations with triple graph grammars and their mapping to unidirectional transformations. In contrast, we proposed metamodelling with partial model completion based on coupled model transformation and provided termination analysis for such model transformation.

In [BPP10], Bottoni et al. proposed an approach to the identification of a sufficient criterion for termination of graph transformations with negative application conditions. Their approach is based on labelled transition systems and they use a double pushout approach to graph transformation. Their approach is focused on the termination of a single non-deleting rule. A graph transformation system typically have multiple rules and termination becomes a complex issue with multiple rules. Ehrig et al. presented a layered graph transformation systems where rules are grouped into different layers [EEPT06]. Mixing deleting and non-deleting rules in a layer is not possible. In this paper, we generalize the layer conditions from [EEPT06] allowing that deleting and non-deleting rules remains in the same layer as long as the rules are loop-free. Furthermore, we permit a rule to use newly created edges allowing us to perform transitive closure operation.

Termination criterion for double pushout transformation that covers transitive closure was addressed in [LPE07] by Levendovszky et al. where the authors proposed a transformation based on E-concurrent production which is an amalgamation technique for concurrent execution of rules. This approach is based on the construction of concurrent rules from different combination of productions. A disadvantage of their approach is that it is hard to find all the possible sequences of graph productions, and prove that the corresponding series of cumulative 'left-hand side' exceeds all limits which is required to show termination.

An application of partial model completion is in the evolving software engineering process. In many cases, software models need to migrate because of the evolution of software requirements and/or modelling languages [TMAL13]. The concept of partial models is applicable also in this setting if accompanied by rewriting rules and would potentially be able to automate the model migration process.

References

[ALPS11] Anthony Anjorin, Marius Lauder, Sven Patzina, and Andy Schürr. eMoflon: Leveraging EMF and Professional CASE Tools. In *3. Workshop Methodische Entwicklung von Modellierungswerkzeugen (MEMWe2011)*, 2011.

[Bec08] Steffen Becker. Coupled model transformations. In *Proceedings of the 7th International Workshop on Software and Performance*, WOSP '08, pages 103–114, NY, 2008. ACM.

[BPP10] Paolo Bottoni and Francesco Parisi-Presicce. A termination criterion for graph transformations with negative application conditions. *ECEASST*, 30, 2010.

[BW95] Michael Barr and Charles Wells, editors. *Category Theory for Computing Science, 2nd Ed.* Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1995.

[EEPT06] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.

[Egy11] Alexander Egyed. Automatically detecting and tracking inconsistencies in software design models. *Software Engineering, IEEE Transactions on*, 37(2):188–204, March 2011.

[LPE07] Tihamer Levendovszky, Ulrike Prange, and Hartmut Ehrig. Termination criteria for dpo transformations with injective matches. *Electr. Notes Theor. Comput. Sci.*, 175(4):87–100, 2007.

[LWM+12] Yngve Lamo, Xiaoliang Wang, Florian Mantz, Wendy MacCaull, and Adrian Rutle. Dpf workbench: A diagrammatic multi-layer domain specific (meta-)modelling environment. In Roger Lee, editor, *Computer and Information Science 2012*, volume 429 of *Studies in Computational Intelligence*, pages 37–52. Springer, 2012.

[MVDS07] Tom Mens and Ragnhild Van Der Straeten. Incremental resolution of model inconsistencies. In *Recent Trends in Algebraic Development Techniques*, volume 4409 of *LNCS*, pages 111–126. Springer, 2007.

[RMWL12] Adrian Rutle, Wendy MacCaull, Hao Wang, and Yngve Lamo. A metamodelling approach to behavioural modelling. In *Proceedings of the Fourth Workshop on Behaviour Modelling - Foundations and Applications*, BM-FA '12, pages 5:1–5:10, NY, USA, 2012. ACM.

[Rut10] Adrian Rutle. *Diagram Predicate Framework: A Formal Approach to MDE*. PhD thesis, Department of Informatics, University of Bergen, Norway, 2010.

[SBP07] Sagar Sen, Benoit Baudry, and Doina Precup. Partial model completion in model driven engineering using constraint logic programming. In *INAP'07*, Germany, 2007.

[Sch06] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. 39(2):25–31, February 2006.

[SFC12] Rick Salay, Michalis Famelis, and Marsha Chechik. Language independent refinement using partial modeling. In *Fundamental Approaches to Software Engineering*, volume 7212 of *LNCS*, pages 224–239. Springer, 2012.

[SLK11] Christoph Schulz, Michael Löwe, and Harald König. A categorical framework for the transformation of object-oriented systems: Models and data. *J. Symb. Comput.*, 46(3):316–337, March 2011.

[TMAL13] Gabriele Taentzer, Florian Mantz, Thorsten Arendt, and Yngve Lamo. Customizable model migration schemes for meta-model evolutions with multiplicity changes. In *16th International Conference, MODELS 2013, Miami, Proceedings*, volume 8107 of *LNCS*, pages 254–270. Springer, 2013.