Electronic Communications of the EASST
Volume X (2014)

Proceedings of the
8th International Workshop on Graph-Based Tools
(GraBaTs 2014)

Algebraic Specification to Realize the Diagram Predicate Framework

No \author defined!

14 pages

Guest Editors: Matthias Tichy, Bernhard Westfechtel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Algebraic Specification to Realize the Diagram Predicate Framework

No \author defined!

No \institute defined!

Abstract: No \abstract defined!

Keywords: model driven engineering, model transformation, metamodelling, rewriting theory, workflow model

1 Introduction

Diagram Predicate Framework (DPF) provides a formal diagrammatic approach to metamodelling based on category theory and model transformation [Rut10]. In this approach models at any level are formalised as diagrammatic specifications which consist of underlying graphs and diagrammatic constraints. The graphs represent the structures of the models; the diagrammatic constaints (also known as predicate constraints) are added to the structure; the graphs and constraints together provide the modelling formalisms of DPF. A DPF metamodelling hierarchy consists of metamodels, models and instances of models. A metamodel specification gives us a modelling language, the specification of a model represents a software system and the instances of models represent possible states of a software system. DPF has been extended in [RMWL12, RWM12] for behavioural modelling. A workflow modelling language called DERF was developed and afterwords it was extended in [WRM12] to define (static) semantics for timed and compensable workflow models and defined the dynamic semantics of models by a transition system where the states are instances and transitions are applications of transformation rules. An important observation of software systems is that very often they have different aspects such as user access, process flow, notification systems. In [RLM14] we introduced the use of multiple metamodelling for designing different aspects of systems that facilitates us with abstraction and less coupling among models. In this paper we provide the detailed semantics of multiple metamodelling with a running example from the healthcare domain. We used algebraic specification to realize the static and dynamic semantics of DPF specifications and the integration of multiple metamodels in this article. Although other executable systems may be used, to simulate the software model designed in DPF we propose the use of Maude system [BM06, CDE^{+}07] which is a high–level programming/specification/modeling language based on rewriting logic theory.

In addition, we present a new type of DPF construct named *rewriting predicate* that allows us to partially specify a model and derives inferred knowledge based on model transformation rules. This reduces the efforts required to explicitly specify the complete model of a system as rewriting predicates expand a partial specification to a complete specification.

2 Rewriting Predicate

A diagrammatic specification is formally specified in [Rut10] as a tuple $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma)$ consisting of an underlying graph S together with a set of *atomic constraints* $C^{\mathfrak{S}}$. The predicate constraints are from a predifined *diagrammatic predicate signature* Σ and they are used to add constraints to S. We modify the definition of **Signature** from [Rut10] with Definition. 1 in order to incorporate *rewriting predicate* into Σ.

Definition 1 (Signature) A (diagrammatic predicate) signature $\Sigma = (P^{\Sigma}, \alpha^{\Sigma} \cup \pi^{\Sigma})$ consists of a collection of predicate symbols P^{Σ} with jointly injective multi-maps $\alpha^{\Sigma} \cup \pi^{\Sigma}$. The map α^{Σ} assigns graphs to predicate symbols $p \in P^{\Sigma}$ and $\alpha^{\Sigma}(p)$ is called the arity of the predicate symbol p. The map π^{Σ} assigns transformation rules to predicate symbols $p \in P^{\Sigma}$ and $\pi^{\Sigma}(p)$ is called the rewriting predicate of p.

Remark 1 *A predicate $p \in P^{\Sigma}$ is either a predicate constraint, denoted as $dom(\alpha^{\Sigma})$, iff $\alpha^{\Sigma}(p) \neq \phi$ or a rewriting predicate, denoted as $dom(\pi^{\Sigma})$, iff $\pi^{\Sigma}(p) \neq \phi$.*

Table 1: Predicate constraints of a sample signature Σ_2

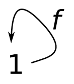
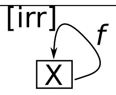
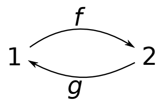
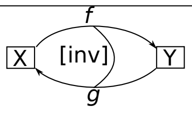
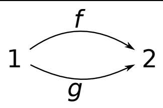
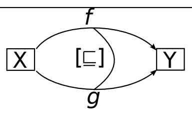
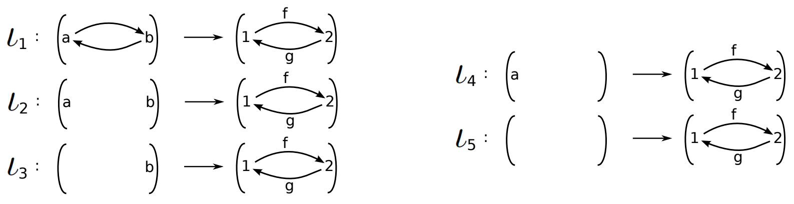
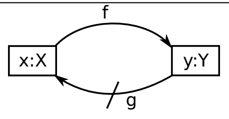
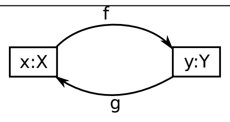
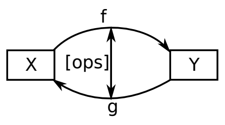
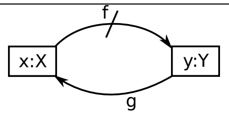
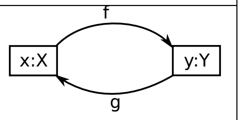
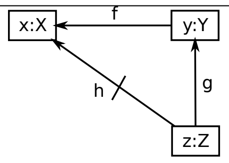
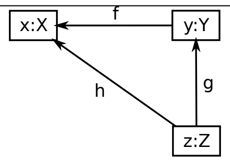
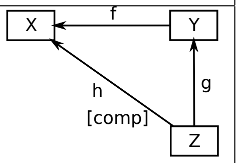
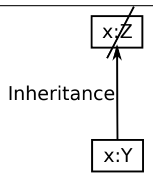
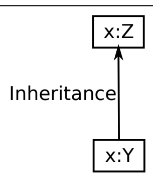
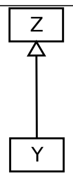
p	$\alpha^{\Sigma_2}(p)$	Proposed Vis.	Semantic Interpretation
[mult(n,m)]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[\text{[n..m]}]{f} \boxed{Y}$	$\forall x \in X : m \leq \lvert f(x) \rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
[irreflexive]			$\forall x \in X : x \notin f(x)$
[injective]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[\text{[inj]}]{f} \boxed{Y}$	$\forall x, x' \in X : f(x) = f(x')$ implies $x = x'$
[surjective]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[\text{[surj]}]{f} \boxed{Y}$	$\forall y \in Y, \exists x \in X, f(x) = y$
[inverse]			$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[image-inclusion]			$\forall x \in X : f(x) \subseteq g(x)$

Table. 1, 2 shows a sample signature $\Sigma_2 = (P^{\Sigma_2}, \alpha^{\Sigma_2} \cup \pi^{\Sigma_2})$ where the first column of both tables show the names of the predicates; the second, third and fourth columns of Table. 1 show the arities of predicates, a possible visualization and the semantic interpretation respectively; the second and third columns of Table. 2 show the transformation rules of predicates and a possible visualization respectively. For readability reasons, the semantic interpretations of *predicate constraints* are presented in a set–theoretical indexed manner in Table. 1, although the actual semantics of each predicate constraint $p \in P^{\Sigma}$ of a sample signature Σ is defined in [Rut10] by a set $[\![p]\!]^{\Sigma}$ of graph homomorphisms $\iota : O \to \alpha^{\Sigma}(p)$ called valid instances of p, where O may vary over all graphs. For example, the valid instances of $[inverse]$ *predicate constraint* of signature Σ_2 (see Table. 1) are defined as follows. In this situation, $[\![inverse]\!]^{\Sigma_2} = \{\iota_1, \iota_2, \iota_3, \iota_4, \iota_5\}$ is a set of graph homomorphisms that provides the semantics of $[inverse]$.



On the other hand, the semantics of *rewriting predicates* are defined by transformation rules with negative application conditions as in Table. 2.

Table 2: Rewriting predicates of a sample signature Σ_2

p	$\pi^{\Sigma_2}(p)$		Proposed Vis.
	$\mathfrak{N} \xleftarrow{n} \mathfrak{L}$	$\mathfrak{R}$	
[opposite]			
			
[composition]			
[inheritance]			

Definition 2 (Semantics of Rewriting Predicates) The semantics of a rewriting predicate $p \in P^{\Sigma}$ is given by a set of transformation rules where each transformation rule $r : \mathfrak{L} \leftarrow \mathfrak{K} \to \mathfrak{R}$ has a negative application condition and there exists a specification morphism $n : \mathfrak{L} \to \mathfrak{N}$.

Notice that the matching patterns $\mathfrak{L}$ and negative application conditions $\mathfrak{N}$ are represented in the same diagram in Table. 2 to make it more readable. Negative application conditions are striked through in the diagram to represent that they must not exist while matching. Now it is worthwhile to make a comparison between these two types of predicates. *Predicate constraints* do not derive any change to an instance of a specification whereas *rewriting predicates* may change the strucutre of an instance. By defining *predicate constraints* we can specify some structural constraints and can decide if an instance is a valid instance of a specification. In order to be consistant we need to modify the definition of **Atomic Constraint** and **Specification** from [Rut10] and include a new definition for **Rewriting Rule** as follows:

Definition 3 (Atomic Constraint) Given a signature $\Sigma = (P^\Sigma, \alpha^\Sigma \cup \pi^\Sigma)$, an atomic constraint (p, δ) added to a graph S is given by a predicate symbol p and a graph homomorphism $\delta : \alpha^\Sigma(p) \to S$ where $p \in P^\Sigma$ is a *predicate constraint*.

Definition 4 (Rewriting Rule) Given a signature $\Sigma = (P^\Sigma, \alpha^\Sigma \cup \pi^\Sigma)$, a rewriting rule (r, δ) attached to a graph S is given by a predicate symbol r and a graph homomorphism $\delta : \pi^\Sigma(r) \to S$ where $r \in P^\Sigma$ is a *rewriting predicate*.

Definition 5 (Specification) Given a signature $\Sigma = (P^\Sigma, \alpha^\Sigma \cup \pi^\Sigma)$, a (diagrammatic) specification $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma, R^{\mathfrak{S}} : \Sigma)$ is given by a graph S, a set $C^{\mathfrak{S}}$ of atomic constraints (p, δ) on S with $p \in dom(\alpha^\Sigma)$, and a set $R^{\mathfrak{S}}$ of rewriting rules (r, δ) on S with $r \in dom(\pi^\Sigma)$.

Lets take the following requirements of a healthcare domain to develop an entity diagram using DPF:

1. *An employee (e.g., nurse, doctor) must work for a department.*
2. *A department may have zero or more employees.*
3. *An ward must be under a department.*
4. *An employee who is involved in a ward, must work in the controlling department.*
5. *Every person has a name.*
6. *A person could be a Patient or an Employee.*
7. *Patients systolic and diastolic blood pressure records are natural numbers.*

Fig. 1(a) shows an entity specification $\mathfrak{S}_1$ that we developed from the above mentioned requirements. In this specification we used the signature Σ_2 from Table. 1 and 2 to define the set $C^{\mathfrak{S}_1}$ of atomic constraints (see Table. 3) and a set $R^{\mathfrak{S}_1}$ of *rewriting rules* (see Table. 4). Requirement 1 is encoded in $\mathfrak{S}_1$ by the $[surjective]$ *predicate constraint* over the morphism $depEmps$; the fourth requirement "an employee who is involved in a ward, must work in the controlling department" is encoded in $\mathfrak{S}_1$ by a *rewriting predicate* $[composition]$ and a *predicate constraint* $[image-inclusion]$ over morphisms $depEmps$, $wardDeps$, $wardEmps$ and $wardEmps'$ where $wardEmps := wardDeps; depEmps$ (the composition of morphisms $wardDeps$ and $depEmps$).

Fig. 1(a) shows a possible instance I of $\mathfrak{S}_1$. Even though I is typed by S, it is not a valid instance of $\mathfrak{S}_1$ since it does not satisfy the *predicate constraints* $[surjective]$, $[inverse]$ and $[image-inclusion]$. We explain this by a pullback operation in Fig. 2. Since $\iota^* \notin [\![inverse]\!]^{\Sigma_2}$, I does not satisfy $[inverse]$. Similarly we can explain why I does not satisfy $[surjective]$ and $[image-inclusion]$.

Until at this point we have not considered the execution of *rewriting predicates* over I. By applying *rewriting predicates* over I we may derive new information. An instance of a specification where *rewriting predicates* are applicable is called a partial instance. Once we apply the *rewriting predicates* $[opposite]$, $[composition]$ and $[inheritance]$ over I we get a complete instance of

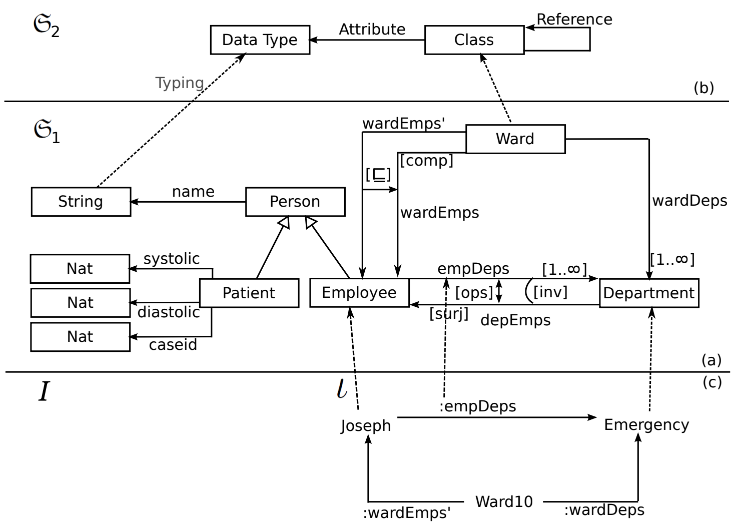


Figure 1: a) Entity specification $\mathfrak{S}_1 = (S, C^{\mathfrak{S}_1} : \Sigma_2, R^{\mathfrak{S}_1} : \Sigma_2)$, b) its meta model $\mathfrak{S}_2$, and c) an instance I of $\mathfrak{S}_1$

specification $\mathfrak{S}_1$ as shown in Fig. 3. We call it a complete instance of $\mathfrak{S}_1$ as no more *rewriting predicates* are further applicable on I. This complete instance is a valid instance of $\mathfrak{S}_1$ as it satisfies all atomic constraints. Now it is easy to understand the motivation of using *rewriting predicate* in DPF as it permits us to specify partial instance of a specification. We may utilize the reasoning capability of *rewriting predicates* to derive a complete instance from a partially specified instance.

Definition 6 (Application of Rewriting Rules) Let $(\Sigma_1 \triangleright S_1, \mathfrak{S}_1, \Sigma_2)$ be a modelling formalism where specification $\mathfrak{S}_1 = (S_1, C^{\mathfrak{S}_1} : \Sigma_2, R^{\mathfrak{S}_1} : \Sigma_2)$ has a set $R^{\mathfrak{S}_1}$ of rewriting rules (r, δ) on S_1. Each rewriting rule (r, δ) is associated with a set of transformation rules $\pi^{\Sigma_2}(r)$. Each transformation rule $t \in \pi^{\Sigma_2}(r)$ has a negative application condition. A rule $t : \mathfrak{L} \leftarrow \mathfrak{K} \rightarrow \mathfrak{R}$ is applicable over $\mathfrak{S}_0 \triangleright S_1$ if there is a match $m : \delta(\mathfrak{L}) \rightarrow \mathfrak{S}_0 \triangleright S_1$ and there does not exist any match to the negative application condition, $q : \delta(\mathfrak{N}) \rightarrow \mathfrak{S}_0 \triangleright S_1$ such that $m = n; q$.

$$\begin{tikzcd}
\delta(\mathfrak{N}) \arrow[drr, "q"'] \arrow[drr, phantom, shift left=3, "\neq"] & & \delta(\mathfrak{L}) \arrow[ll, "n"'] \arrow[d, "m"] & \delta(\mathfrak{K}) \arrow[l] \arrow[r] \arrow[d, "m'"] & \delta(\mathfrak{R}) \arrow[d, "m*"] \\
& & \mathfrak{S}_0 \triangleright S_1 & \mathfrak{S}'_0 \triangleright S_1 \arrow[l] \arrow[r] & \mathfrak{S}^*_0 \triangleright S_1
\end{tikzcd}$$

Definition 7 (Partial and Complete Instance) Let $\mathfrak{S} = (S, C^{\mathfrak{S}} : \Sigma, R^{\mathfrak{S}} : \Sigma)$ be a specification with a graph S, a set $C^{\mathfrak{S}}$ of atomic constraints (p, δ) on S with $p \in dom(\alpha^{\Sigma})$, and a set $R^{\mathfrak{S}}$ of rewriting rules (r, δ) on S with $r \in dom(\pi^{\Sigma})$. An instance I is a *partial instance* of $\mathfrak{S}$ if there exist some rewriting rules $(r, \delta) \in R^{\mathfrak{S}}$ those are applicable over I. An instance I is a *complete instance* of $\mathfrak{S}$ if there does not exist any rewriting rule $(r, \delta) \in R^{\mathfrak{S}}$ that is applicable over I.

Remark 2 *A complete instance I is a valid instance of specification $\mathfrak{S}$ if it does not violate any*

Table 3: The set of atomic constraints $C^{\mathfrak{S}_1}$ of $\mathfrak{S}_1$

(p,δ)	$\alpha^{\Sigma_2}(p)$	$\delta(\alpha^{\Sigma_2}(p))$
$([mult(1,\infty)],\delta_1)$	$1 \xrightarrow{f} 2$	$\boxed{\text{Employee}} \xrightarrow{\text{empDeps}} \boxed{\text{Department}}$
$([mult(1,\infty)],\delta_2)$	$1 \xrightarrow{f} 2$	$\boxed{\text{Ward}} \xrightarrow{\text{wardDeps}} \boxed{\text{Department}}$
$([surjective],\delta_3)$	$1 \xrightarrow{f} 2$	$\boxed{\text{Department}} \xrightarrow{\text{depEmps}} \boxed{\text{Employee}}$
$([inverse],\delta_4)$	$\begin{tikzcd} 1 \arrow[r, bend left, "f"] & 2 \arrow[l, bend left, "g"] \end{tikzcd}$	$\begin{tikzcd} \boxed{\text{Employee}} \arrow[r, bend left, dash, "\text{depEmps}"] \arrow[r, bend right, dash, "\text{empDeps}"'] & \boxed{\text{Department}} \end{tikzcd}$
$([image-inclusion],\delta_5)$	$\begin{tikzcd} 1 \arrow[r, bend left, "f"] \arrow[r, bend right, "g"'] & 2 \end{tikzcd}$	$\begin{tikzcd} \boxed{\text{Ward}} \arrow[r, bend left, dash, "\text{wardEmps}"] \arrow[r, bend right, dash, "\text{wardEmps'}"'] & \boxed{\text{Employee}} \end{tikzcd}$

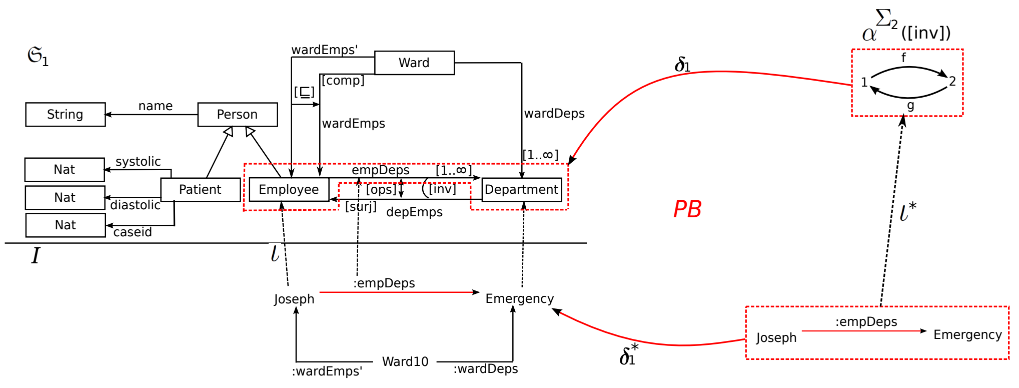


Figure 2: Pullback $\alpha^{\Sigma_2}([inv]) \xleftarrow{\iota^*} O^* \xrightarrow{\delta_1^*} I$ of $\alpha^{\Sigma_2}([inv]) \xrightarrow{\delta_1} S \xleftarrow{\iota} I$

constraint from $C^{\mathfrak{S}}$. We denote it as $I \blacktriangleright \mathfrak{S}$

3 Integration of Multiple Metamodels

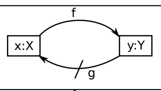
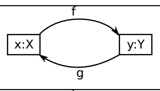
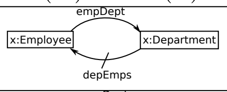
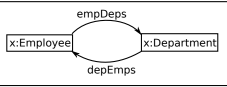
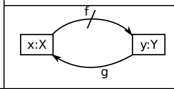
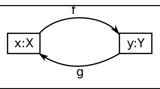
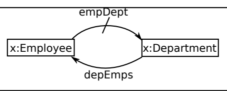
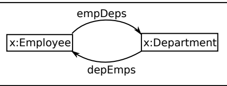
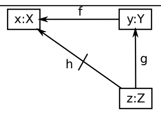
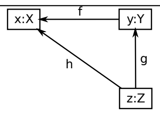
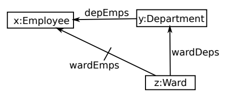
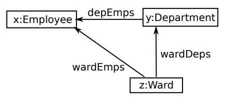
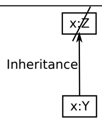
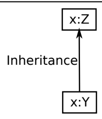
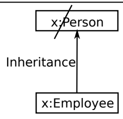
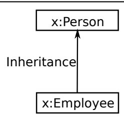
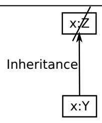
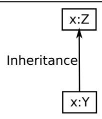
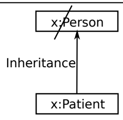
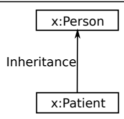
In this section we formally define the integration of specifications and provide its syntax and semantics with a running example.

Definition 8 (Integrated Specification) Given two complete and valid specifications $\mathfrak{S}$ and $\mathfrak{T}$, their integration $\mathfrak{J} = (J, C^{\mathfrak{J}} : \Sigma^J, R^{\mathfrak{J}} : \Sigma^J)$ is given by a graph $J = (S \uplus T)$: a disjoint union of the underlying graphs of $\mathfrak{S}$ and $\mathfrak{T}$, a set $C^{\mathfrak{J}}$ of atomic constraints (p,δ) on J with $p \in dom(\alpha^{\Sigma^J})$ and a set of $R^{\mathfrak{J}}$ of rewriting rules (r,δ) on J with $r \in dom(\pi^{\Sigma^J})$.

Proposition 1 *The integration of two complete and valid instances $\mathfrak{S}_0, \mathfrak{T}_0$ of specifications $\mathfrak{S}$ and $\mathfrak{T}$ respectively is a partial instance $\mathfrak{J}_0$ of $\mathfrak{J}$.*

The integrated partial instance $\mathfrak{J}_0$ is a typed specification $\mathfrak{J}_0 \triangleright J$ that becomes a complete

Table 4: The set of rewriting rules $R^{\mathfrak{S}_1}$ of $\mathfrak{S}_1$

(r,δ)	$\pi^{\Sigma_2}(r)$		$\delta(\pi^{\Sigma_2}(r))$	
	$\mathfrak{N} \xleftarrow{n} \mathfrak{L}$	$\mathfrak{R}$	$\delta(\mathfrak{N}) \xleftarrow{n} \delta(\mathfrak{L})$	$\delta(\mathfrak{R})$
$([opposite],\delta_6)$				
				
$([composition],\delta_7)$			 x:Employee depEmps y:Department wardDeps wardEmps z:Ward	 x:Employee depEmps y:Department wardDeps wardEmps z:Ward
$([inheritance],\delta_8)$				
$([inheritance],\delta_9)$				

instance $\mathfrak{J}_0^*$ of the integrated specification $\mathfrak{J}$ by iterative applications of the set $R^{\mathfrak{J}}$ of rewriting rules. Applying pullback over a complete integrated specification $\mathfrak{J}_0^*$ we get two instances $\mathfrak{S}_0'$ and $\mathfrak{T}_0'$ of specifications $\mathfrak{S}$ and $\mathfrak{T}$ respectively.

$$\begin{tikzcd}
S \arrow[r, hook] & J & T \arrow[l, hook'] \\
\mathfrak{S}_0' \arrow[u] \arrow[r, hook] \arrow[ur, phantom, "P.B"] & \mathfrak{J}_0^* \arrow[u] \arrow[ur, phantom, "P.B"] & \mathfrak{T}_0' \arrow[u] \arrow[l, hook']
\end{tikzcd}$$

$\mathfrak{S}_0'$ and $\mathfrak{T}_0'$ may be partial instances with respect to their specifications and further rewriting rules may be applicable from the set $R^{\mathfrak{S}}$ and $R^{\mathfrak{T}}$. The complete and valid instances $\mathfrak{S}_0^*, \mathfrak{T}_0^*$ of $\mathfrak{S}$ and $\mathfrak{T}$ after applying rewriting rules over $\mathfrak{S}_0', \mathfrak{T}_0'$ may be integrated again and be used for further derivation. This derivation process can run again and again until no further rewriting rule is applicable:

$$\begin{tikzcd}
\mathfrak{S}_0 \arrow[d, hook] & \mathfrak{S}_0^1 \arrow[d, hook] \arrow[r] & \mathfrak{S}_0^{1*} \arrow[d, hook] & \mathfrak{S}_0^{n*} \arrow[d, hook] \\
\mathfrak{J}_0 \arrow[r] & \mathfrak{J}_0^* & \mathfrak{J}_0^1 \arrow[r, dotted] & \mathfrak{J}_0^n \\
\mathfrak{T}_0 \arrow[u, hook] & \mathfrak{T}_0^1 \arrow[u, hook] \arrow[r] & \mathfrak{T}_0^{1*} \arrow[u, hook] & \mathfrak{T}_0^{n*} \arrow[u, hook]
\end{tikzcd}$$

As an example for the integration of multiple metamodels we take a DERF process model and an entity model from a healthcare domain and present a data aware workflow model. Fig. 4

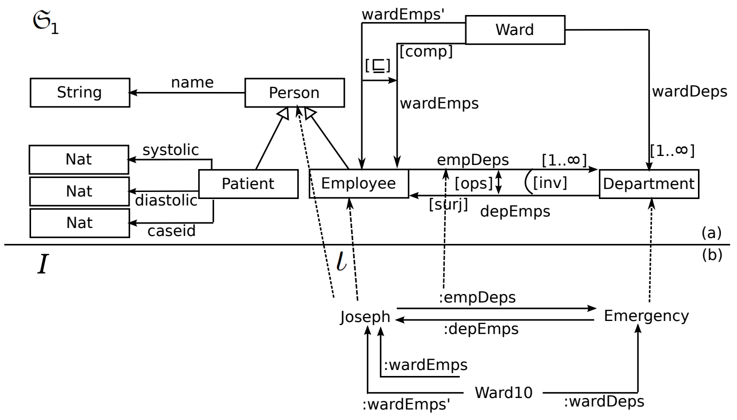


Figure 3: a) Entity specification $\mathfrak{S}_1 = (S, C^{\mathfrak{S}_1} : \Sigma_2, R^{\mathfrak{S}_1} : \Sigma_2)$ and b) a complete instance I of $\mathfrak{S}_1$

shows a hypertension management workflow model with its meta model. The workflow model is developed from a clinical practice guideline [HTN].

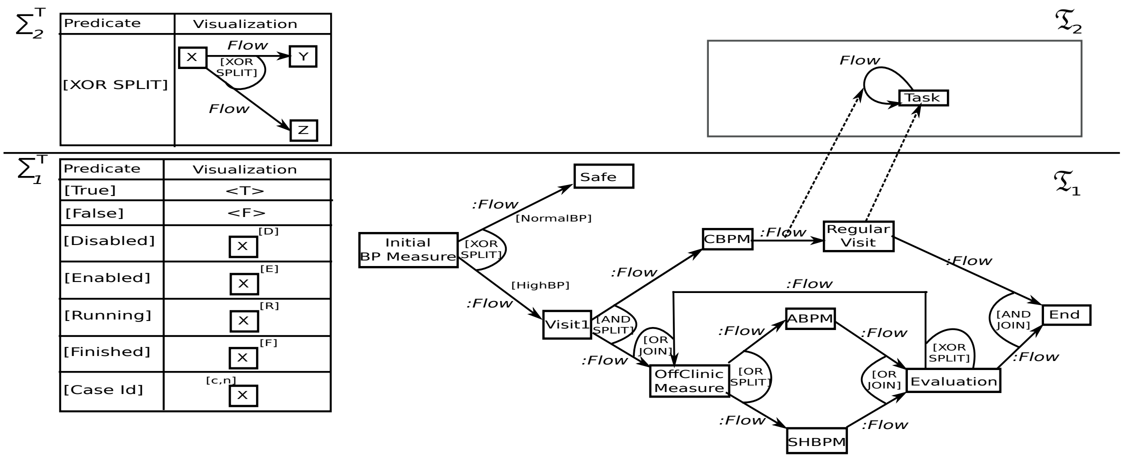


Figure 4: Example DERF process model specification: Hypertension Workflow Model

The behavior of DERF process models are given by a coupled transition system. The hypertension workflow model consists of some predicates *[XOR SPLIT]*, *[AND SPLIT]*, *[OR SPLIT]*, etc,. which describes the control flow of the process model. The dynamic semantics of DERF workflow models are provided in [RMWL12]. The predicates *[Disabled]*, *[Enabled]*, etc,. from predicate signature Σ_1^T are annotations and they do not have a semantic counterpart. Task instances with annotations in specification $\mathfrak{S}_0$ give us a state of a workflow instance. And the coupled transition systems produce reachable instances of a workflow model from an initial instance.

Fig. 5 shows two specifications $\mathfrak{T}_0$ and $\mathfrak{T}_0'$ which are reachable instances of $\mathfrak{T}_1$. The predicate $[c, 11]$ denotes that they are the instance of case no 11. For every patient there are different instances with different case numbers. In $\mathfrak{T}_0$ the task instance *:Initial BP Measure* is finished (annotated with $< F >$), the branch *:f* and *:g* are evaluated to true and false recursively and both task instances *:Safe* and *:Visit1* are disabled (annotated with $< D >$). In $\mathfrak{T}_0$ the task instance *:Safe* is enabled (annotated with $< E >$).

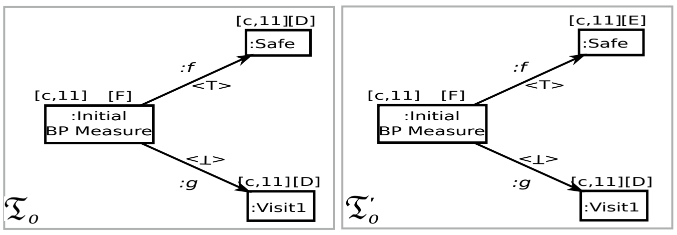
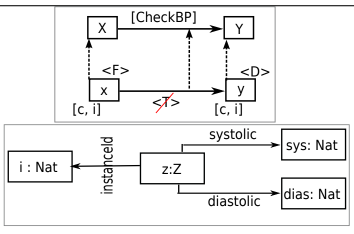
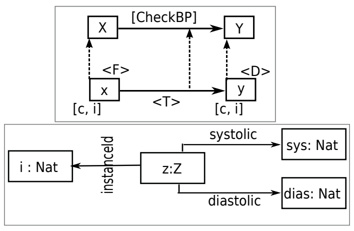
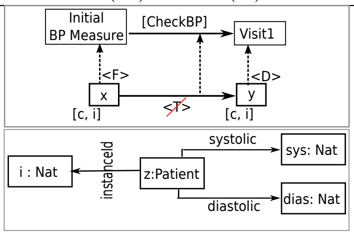
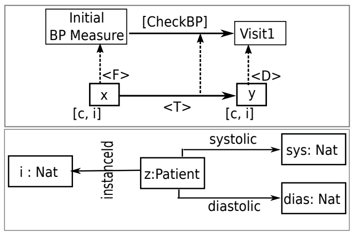


Figure 5: Two reachable instances of the hypertension workflow model

We design a data aware process model specification $\mathfrak{J}_1$ by integrating the hypertension workflow model specification $\mathfrak{T}_1$ and the entity specification $\mathfrak{S}_1$ from Fig. 1. For the integrated specification $\mathfrak{J}_1$ we need to specify a set of atomic constraints and a set of rewriting rules. We may use the predicate symbols associated with the shape graphs S, T from Σ_2^S, Σ_2^T and from the predicate signatures Σ_1^S, Σ_1^T of specifications $\mathfrak{S}_1, \mathfrak{T}_1$. For this particular example, we defined a set $R^{\mathfrak{J}_1}$ of rewriting rules (r, δ) for the integrated specification. Table. 5 shows the transformation rule of predicate $[HighBP]$. In the transformation rule we included a *condition* of the form $(\forall X)\varphi : \bigwedge_i p_i = q_i$ where all the variables in p_i, and q_i are in $\mathfrak{L}$.

Table 5: The rewriting predicate $[HighBP]$ from $R^{\mathfrak{J}_1}$ of $\mathfrak{J}_1$

(r,δ)	$\pi^{\Sigma_1}(r)$	
	$\mathfrak{N} \xleftarrow{n} \mathfrak{L}$	$\mathfrak{R}$
$([HighBP], \delta_1)$	 φ: (sys > 140 && dias > 90)	
	$\delta(\pi^{\Sigma_1}(r))$	
	$\delta(\mathfrak{N}) \xleftarrow{n} \delta(\mathfrak{L})$	$\delta(\mathfrak{R})$
	 φ : (sys > 140 && dias > 90)	

4 Algebraic Specification for Metamodel Integration

One of our intension to use algebraic specification for DPF metamodelling is to develop an execution environment for DPF framework where one could design and simulate a software model with multiple metamodels. Fig. 6 shows a proposed system architecture for DPF simulate tool where maude tool has been used as an engine to perform 1) Type checking, 2) Conformance checking, and 3) Reasoning / derivation. During the design phase, the tool may be used to provide feedback to the designer if the model he is working on satisfies its metamodel or not. We intend to use a user-friendly user interface (UI) to support better user experience for the developer to simulate a model instance. During the simulation phase, maude tool will be used to produce the instances of a software model.

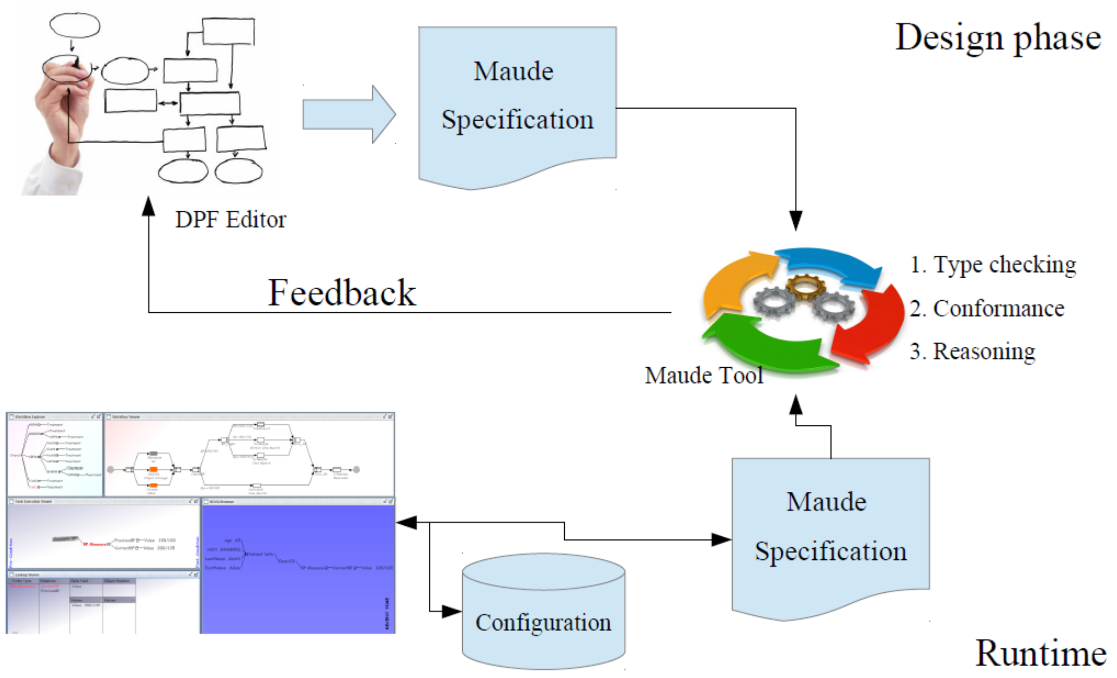


Figure 6: System architecture for DPF simulation tool

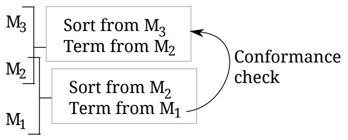
Since DPF is based on metamodelling, a natural choice would be to use the META-LEVEL programming capability of Maude specification. A detailed description about maude metalevel programming and metalanguage applications may be found from Chapters 14 and 17 of [CDE+07]. Maude metalevel programming is based on a “reflective tower” with an arbitrary number of levels of reflection. We can represent in a rewrite theory U, any finitely presented rewrite theory R (including U itself) as a term $\overline{R}$, any terms t,t' in R as terms $\overline{t},\overline{t}$, and any pair (R,t) as a term $\langle \overline{R},\overline{t} \rangle$.

$$R \vdash t \rightarrow t' \Leftrightarrow U \vdash \langle \overline{R},\overline{t} \rangle \rightarrow \langle \overline{R},\overline{t'} \rangle \Leftrightarrow U \vdash \langle \overline{U}, \overline{\langle \overline{R},\overline{t} \rangle} \rangle \rightarrow \langle \overline{U}, \overline{\langle \overline{R},\overline{t'} \rangle} \rangle \ldots$$

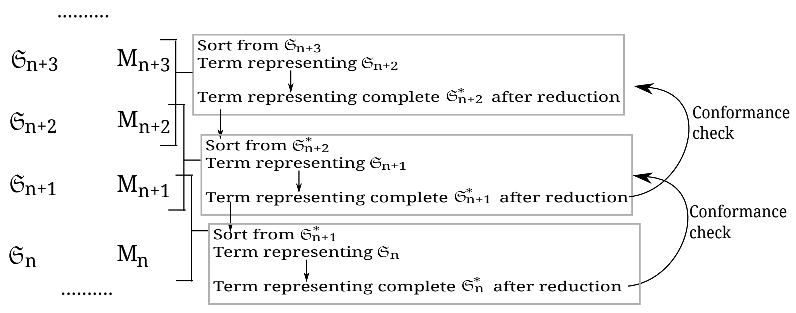
In this chain of equivalences the first computation takes place at level 0, the second computation at level 1, and so on. This is a barrier to use maude metalevel programming for DPF as the computation of a single step requires considerable computational cost that involves many rewriting steps one level up. For this reason it was suggested for maude to lower the number of levels of reflective computation as much as possible, so that a rewriting subcomputation happens at a higher level in the tower only when this is strictly necessary [CDE+07]. But the semantic of DPF metamodel specifications are provided in the opposite way. Once a specification $\mathfrak{S}_i$ is defined in DPF, the computation regarding type checking, conformance checking and reasoning of the instance $\mathfrak{S}_{i+1}$ of the specification $\mathfrak{S}_i$ does not involve any computation regarding the type checking, conformance checking or reasoning of $\mathfrak{S}_i$ or any of its metamodel $\mathfrak{S}_{i-1}$, $\mathfrak{S}_{i-2}, etc,$. anymore.

Another issue regarding the use of maude metalevel programming is the complex sort infrastructure provided by the module META-TERM. In order to write a rewriting rule at a higher level, requires the use of metarepresentation of terms. The metarepresentation of terms, when they are iterated, becomes very complex; for example, the metarepresentation of a natural number such as, e.g., 42 is $'s\_\hat{}42['0.Zero]$. If we wish to make a reusable DPF specification, this would become a big issue as the rewriting theories metarepresented in one signature may not match in another metalevel hierarchy. In general, for DPF, while we are specifying semantics for a higher level specification, we should not practice using a concept/name from a specification at lower metalevel.

An algebraic semantics for MOF was presented in [BM10] which is very relevant with our situation. In that paper the authors proposed an algebraic, mathematical semantics for MOF in membership equational logic (MEL). The algebraic semantics that they proposed, exploits the reflective features of both MOF and MEL and they used a modular and stepwise approach in the definition of the semantic function. An abstraction of the reflective bootstrapping strategy to define the semantic function is shown here:



A straight forward translation of DPF specifications to maude using this approach would not work since the specifications in DPF have dynamic semantics; so the *sort* declaration for any meta level n requires parsing of the reduced term at level $n+1$. This is pictorically represented in the following diagram:



Another complexity is to deal with the integrated specification since it requires to perform a disjoint union operation over the underlying graphs of two specifications and the pullback operation after reduction over the term represnting the integrated specification. We present an object oriented stepwise approach in maude specification where each object represent a specification consisting of typed graphs and their association with predicates. Predicate constraints, rewriting predicates, and type checking using graph homomorphism are encoded as conditional rewriting rules. We control the execution of the rewriting rules by an external program written in Java. Terms of sort $Object$ are defined by means of the constructor in maude specification:

$$\text{op} < \_ : \_|\_ > : \text{Oid Cid AttributeSet} \rightarrow \text{Object [ctor object] .}$$

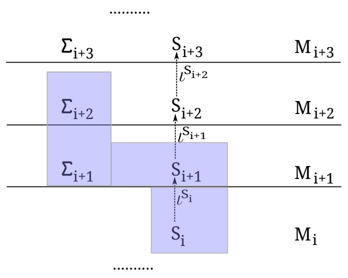
Part of *sorts*, *subsorts* and *operator* declarations of maude specification related to the *AttributeSet* is given below:

```
sorts Attribute AttributeSet Atom SigAtom DpfGraph SignatureMaps SIGID .
subsort Atom < DpfGraph .
subsort SigAtom < SignatureMaps .

op _,_ : AttributeSet AttributeSet  -> AttributeSet
       [ctor assoc comm id: none] .
op relations  :_ : DpfGraph -> Attribute [ctor gather (&)] .
op sigMaps   :_ : SignatureMaps -> Attribute [ctor gather (&)] .

op rel   : String String String String -> Atom .
op rel   : String String String Nat -> Atom .
...
op sigmap   : SIGID String String -> SigAtom .
```

Our stepwise translation approach is depicted in the following diagram. Consider there are n number of meta levels: $M_0, M_1, ..M_{n-1}$, in every step we translate a metamodelling formalism $\Sigma_{i+1}, \mathfrak{S}_{i+1}, \Sigma_{i+2}$ and a specification $\mathfrak{S}_i$ where $\mathfrak{S}_{i+1} \blacktriangleright \mathfrak{S}_{i+2}$, and $\mathfrak{S}_i$ is a partial instance of $\mathfrak{S}_{i+1}$. After applying the rewriting rules from $R^{\mathfrak{S}_i}$ over $\mathfrak{S}_i$ we check if the derived specification is valid or not using the set $C^{\mathfrak{S}_i}$ of predicate constraints attached to $\mathfrak{S}_i$. The derived specification if found a valid instance of $\mathfrak{S}_{i+1}$ is then used as a meta model in the next step of translation.



Terms using the *rel* and *sigmap* operators presented in the above maude specification are used to hold a typed graph and a signature map respectively using multimap. The arguments of a *rel* term represents a *name*, *type*, *domain* and range respectively. Here *name* is either referring to the name of a vertex or a morphism depends on the *type*. The arguments of a *sigmap* term represents an *Id*, *name* of the vertex or morphism, and *predicate name* respectively. A sample maude specification is provided below that represents the specification $\mathfrak{S}_1$ and its instance I from Fig. 1. The subterm *rel("G0", "TypeOf", "Employee", "Class")* represents that *Employee* is a vertex of type *Class*, and the subterm *rel(":Typeof", "TypeOf", "Joseph", "Employee")* represents that *Joseph* is a vertex of type *Employee*. Here G0 denotes that *Employee* vertex will be treated as a metalevel node. The subterm *rel("empDeps", "G1", "Employee", "Department")* represents that *empDeps* is a morphism where the source is *Employee* and target is *Department*. The subterm *rel(":empDeps", "empDeps", "Joseph", "Employee")* represents an instance of the morphism *empDeps* from source *Joseph* to *Emergency*.

```
op initState : -> Configuration .
eq initState =
< DpfInstance : DPF |
  relations :
  rel("G0", "TypeOf", "Employee", "Class")
  rel("G0", "TypeOf", "Department", "Class")
  rel("G0", "TypeOf", "Ward", "Class")
  rel("G0", "TypeOf", "Person", "Class")
  ....
  rel("empDeps", "G1", "Employee", "Department")
  rel("depEmps", "G1", "Department", "Employee")
  ....
  rel(":TypeOf", "TypeOf", "Joseph", "Employee")
  rel(":TypeOf", "TypeOf", "Emergency", "Department")
  rel(":TypeOf", "TypeOf", "Ward10", "Ward")
  rel(":empDeps", "empDeps", "Joseph", "Emergency")
  ,
  sigMaps :
  sigmap(Sig3, "empDeps", "ops" ) sig(Sig3, "depEmps", "ops" )
  ....
> .
```

The following maude specification checks graph homomorphism for DPF instances if they are type correct. We use maude *search* command to check for valid and complete instances.

```
crl [Phi0] : < DpfInstance : DPF | relations : rel(R1, "TypeOf", X, Y) G , sigMaps : SG >
 => error (RULE-PHI0, rel(R1, "TypeOf", X, Y) )
if (R1 =/= "G0") /\ (not type-exists("TypeOf", Y, G) ) .

crl [Phi1] : < DpfInstance : DPF | relations : rel(R1, R2, W, Z) rel(R2, "G1", X, Y) G , sigMaps :  SG >
 => error (RULE-PHI1, rel(R1, R2, W, Z) rel(R2, "G1", X, Y) )
if (R2 =/= "G1") /\ ((not supertype-exists(W, X, G)) or (not supertype-exists(Z, Y, G))) .
```

The maude specification presented in this section can easily be used to integrate one specification with another one as the disjoin union and pullback operations do not require any parsing since each specification is represented as a seperate *Object*.

5 Related Work

6 Conclusion

Bibliography

[BM06] R. Bruni, J. Meseguer. Semantic Foundations for Generalized Rewrite Theories. *Theor. Comput. Sci.* 360(1):386–414, Aug. 2006.

[BM10] A. Boronat, J. Meseguer. An algebraic semantics for MOF. *Formal Asp. Comput.* 22(3-4):269–296, 2010.

[CDE$^+$07] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. L. Talcott (eds.). *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. Lecture Notes in Computer Science 4350. Springer, 2007.

[HTN] The Chinook Primary Care Network. www.chinookprimarycarenetwork.ab.ca/extranet/docs/guides/7.pdf.

[RLM14] F. Rabbi, Y. Lamo, W. MacCaull. A Flexible Metamodelling Approach for Healthcare Systems. *2nd European Workshop on Practical Aspects of Health Informatics (PAHI), Submitted*, 2014.

[RMWL12] A. Rutle, W. MacCaull, H. Wang, Y. Lamo. A Metamodelling Approach to Behavioural Modelling. In *Proceedings of BM-FA 2012: 4th Workshop on Behavioural Modelling: Foundations and Applications*. Pp. 5:1–5:10. ACM, 2012.

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

[RWM12] A. Rutle, H. Wang, W. MacCaull. A Formal Diagrammatic Approach to Compensable Workflow Modelling. In Weber and Perseil (eds.), *FHIES*. Lecture Notes in Computer Science 7789, pp. 194–212. Springer, 2012.

[WRM12] H. Wang, A. Rutle, W. MacCaull. A Formal Diagrammatic Approach to Timed Workflow Modelling. In *Proceedings of TASE 2012: 6th International Conference on Theoretical Aspects of Software Engineering*. Volume 0, pp. 167–174. IEEE Computer Society, 2012.