WebDPF: A Web-based Metamodelling and Model Transformation Environment

Fazle Rabbi[1,2], Yngve Lamo[1], Ingrid Chieh Yu[2] and Lars Michael Kristensen[1]
[1]*Bergen University College, Bergen, Norway*
[2]*University of Oslo, Oslo, Norway*

Keywords: Model Driven Engineering, Metamodelling, Model Transformation, Domain Specific Modelling, Model Completion.

Abstract: Metamodelling and model transformation play important roles in model-driven engineering as they can be used to define domain-specific modelling languages. During the modelling phase, modellers encode domain knowledge into models which may include both structural and behavioral aspects of a system. The contribution of this paper is a new web-based metamodelling and model transformation tool called WebDPF based on the Diagram Predicate Framework (DPF). WebDPF supports multilevel diagrammatic metamodelling and specification of model constraints, and it supports diagrammatic development and analysis of model transformation systems. We show how the support for model transformation systems in WebDPF can be exploited to (i) support auto-completion of partial models thereby enhancing modelling efficiency, and (ii) provide execution semantics for workflow models. Furthermore, we illustrate how WebDPF incorporates a scalable model navigation facility designed to enable users to inspect and query large models.

1 INTRODUCTION

Software development productivity can be greatly enhanced by applying Model Driven Engineering (MDE) (Schmidt, 2006). MDE is considered to be an effective way of reducing repetitive, error-prone, and time-consuming tasks through automation, without sacrificing software quality. Although MDE was introduced to reduce the complexity of system development, it adds in many cases accidental complexity (Whittle et al., 2013). This problem gets worse when software models become larger. One of the main reason of MDE not being adopted in the software development process, as pointed out by many authors, is inadequate tool support (Tomassetti et al., 2012). There are reasons to believe that with advanced tool support, MDE can be successfully applied to model and develop large-scale software applications.

There exists several MDE tools such as DIAGEN (Mazanek et al., 2008), WebGME (Maróti et al., 2014), AToMPM (Syriani et al., 2013), Melanie (Kennel, 2012), AGG (Ehrig et al., 2006), eMoflon (Anjorin et al., 2011) for modelling and development of domain specific modelling languages (DSML). These tools include various features which are essential for developing DSMLs. In this paper, we present WebDPF which is a web-based tool inspired by the tools listed above yet incorporating a distinguished set of features including support for scalability concerning the size of models, completion of partial models, and the construction of reusable components for structural and behavioral modelling of systems.

WebDPF has been developed using HTML5 and JavaScript.

Any HTML5 and JavaScript enabled web browser can be used for metamodelling with WebDPF. Algorithms related to model transformation and analysis in WebDPF are written in JavaScript and therefore executes on the client machine.

The WebDPF metamodelling environment is based on the Diagram Predicate Framework (DPF) (Rutle, 2010) which supports multilevel metamodelling. Earlier in (Lamo et al., 2012), an Eclipse-based diagrammatic workbench, DPF-WB has been presented. The WebDPF is built on the foundation of the DPF yet incorporating a number of new features to ease metamodelling.

In WebDPF, one can graphically specifiy constraints and model transformation rules. Transformation rules have been introduced in WebDPF for two purposes: i) automatic rewriting of partial (incomplete) models so that they can be made to conform

Rabbi, F., Lamo, Y., Yu, I. and Kristensen, L.
WebDPF: A Web-based Metamodelling and Model Transformation Environment.
DOI: 10.5220/0005686900870098
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 87-98
ISBN: 978-989-758-168-7

to the underlying metamodel; ii) modelling the behavior of systems. In the process of software development, modellers are often confronted with a variety of inconsistencies and/or incompleteness in the models under construction (Mens and Van Der Straeten, 2007). In particular, the modeller will most of the time be working with a *partial model* not conforming to the metamodel that defines the modelling language being used (Sen et al., 2007), i.e., it is not typed by and/or not satisfying modelling constraints. One aspect of using model transformation rules in WebDPF is to reduce the modelling effort of the modeller by means of model completion[1]. Therefore, the model transformation rules that are intended to automatically complete a partial model are referred to as *completion rules*. Another aspect of using model transformation rules in WebDPF is to model the behavior of a system. While designing the behavior of a transition system, the modeller can take advantage of reusing rules. Rules that are intended to model the behavior are referred to as *production rules*. The tool exploits the locality of model transformation rules for termination analysis and provides a foundation that enables automated tool-support to increase modelling productivity.

Scalability is a general problem for graphical languages as graphical models occupy more space than text. In order to tackle this issue, WebDPF implements a model navigation feature that allows a designer to search for desired model elements and display a small fragment of a model in the modelling editor. This approach provides better visualization support for the designer to focus on essential parts of his/her work.

The foundation of the paper is based on category theory and graph transformation systems (Barr and Wells, 1995), (Ehrig et al., 2006). The rest of this paper is organised as follows. Section 2 provides a description of multilevel metamodelling and background knowledge on DPF, section 3 provides an introduction to the WebDPF tool. In section 4 we provide a description of the model navigation facility. In section 5 we present the underlying diagrammatic transformation system of WebDPF. In section 6 and section 7 we provide a detailed description of model completion and production rules with examples. Sections 8 and 9 present some related work and conclude the paper.

2 BACKGROUND ON DPF

Multilevel metamodelling offers a clean, simple and coherent semantics for metamodelling (Atkinson and Kühne, 2001) which is an essential requirement for the development of domain specific modelling languages. DPF is a language independent formalism for defining metamodelling hierarchies. DPF allows us to construct a metamodelling hierarchy in which a model at any level can be considered as a metamodel wrt. the models at the level below it. Models at each level in a metamodelling hierarchy are specified by a modelling language at the level above and conform to the corresponding metamodel.

In the DPF approach, models at any level are formalised as diagrammatic specifications which consist of type graphs annotated with diagrammatic constraints. To briefly introduce DPF, we consider requirements R1-R5 for an envisioned system in the healthcare domain:

R1. *A Caregiver (e.g., nurse, doctor) must work for at least one Department (possibly more).*

R2. *A Ward must be controlled by exactly one Department.*

R3. *A Caregiver who is involved in a Ward, must work for its controlling Department.*

R4. *All Caregivers assigned to a Ward have access to the patient information who are admitted to the same Ward.*

R5. *A Caregiver who works for a Department must be registered for that Department.*

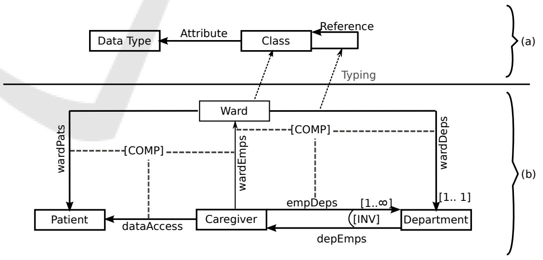


Figure 1: a) Metamodel $\mathfrak{S}_0$, b) Model $\mathfrak{S}_1$.

Figure 1 shows a DPF model (specification) $\mathfrak{S}_1$ with its metamodel (specification) $\mathfrak{S}_0$ developed from the above requirements. The type graphs in Figure 1 represent the structure of the models; constraints are added into the structure by predicates. Table 1 lists the predicates used for constraining the specification $\mathfrak{S}_1$. Each predicate has a name (p), a shape graph (arity, $\alpha(p)$), a visualisation, and a semantic interpretation. In Table 1, we used set theoretic notation to describe the semantics of the predicates. For example the intended semantics of $[mult(n,m)]$ is that: f must have at least n and at most m instances.

[1] Model completion is also known as model repair technique in some literature (Macedo et al., 2015).

Table 2 shows how the predicates are constraining the specification $\mathfrak{S}_1$ by means of graph homomorphisms $\delta : \alpha(p) \to S$ from the shape graph to the specification. Multiplicity constraints are added into the specification by graph homomorphisms from the shape graph $(1 \xrightarrow{f} 2)$ of $[mult(n,m)]$ predicate to the model elements (e.g., $Caregiver \xrightarrow{empDeps} Department$) . The graph homomorphism is indicated by subscripts in Table 2. 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. R1 is encoded in $\mathfrak{S}_1$ by the $[mult(1,\infty)]$ predicate on the edge *empDeps*; R2 by the $[mult(1,1)]$ predicate on the edge *wardDeps*; R3 by the $[composite]$ predicate on edges *wardEmps*, *wardDeps* and *empDeps*; R4 by the $[composite]$ predicate on edges *wardEmps*, *wardPats* and *dataAccess*; and R5 by the $[inverse]$ predicate on edges *empDeps*, *depEmps*.

Table 1: Predicates of a sample signature Σ_0.

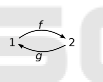
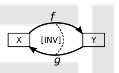
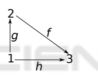
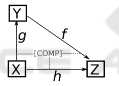
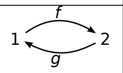
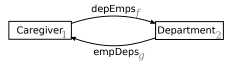
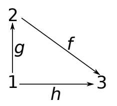
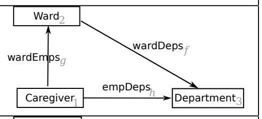
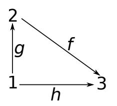
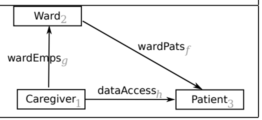
p	$\alpha(p)$	visualisation	Semantic Interpretation
[mult(n,m)]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[\text{[n..m]}]{f} \boxed{Y}$	f must have at least n and at most m instances. Formally, $\forall x \in X : n \leq \lvert f(x) \rvert \leq m$, with $0 \leq n \leq m$ and $n \geq 1$
[inverse]			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, $\forall x \in X : \bigcup \{f(y) \mid y \in g(x)\} \subseteq h(x)$

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

(p,δ)	$\alpha(p)$	$\delta(\alpha(p))$
$([mult(1,\infty)],\delta_1)$	$1 \xrightarrow{f} 2$	$\boxed{\text{Caregiver}_1} \xrightarrow{\text{empDeps}_f} \boxed{\text{Department}_2}$
$([mult(1,1)],\delta_2)$	$1 \xrightarrow{f} 2$	$\boxed{\text{Ward}_1} \xrightarrow{\text{wardDeps}_f} \boxed{\text{Department}_2}$
$([inverse],\delta_3)$		
$([composite],\delta_4)$		 Ward 2 wardEmps g wardDeps f Caregiver 1 empDeps h Department 3
$([composite],\delta_5)$		 Ward 2 wardEmps g wardPats f Caregiver 1 dataAccess h Patient 3

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}.

3 INTRODUCTION TO WEBDPF

The WebDPF editor (see Figure 2) consists of four resizable windows. The windows are arranged in a single view which provides the modeller with an overview of different modelling artefacts. The control panel on the left allows the user to select metamodels from the metamodel stack and also provides options to perform analysis such as conformance checking and termination analysis of model transformation (see section. 5). The conformance checking is used for validating whether a model conforms to its metamodel and the termination analysis is used for checking whether the application of transformation rules are terminating. While designing a model using the WebDPF model editor, the metamodel viewer displays the metamodel to help the modeller choose types for modelling elements. The signature editor is used to graphically define the arity and visualization of predicates. An atomic constraint can be defined by selecting a predicate in the signature editor and binding the arity of the predicate to the model elements from the model editor.

The semantics of the predicates are provided in a fibred manner (Diskin and Wolter, 2008). That is, the semantics of a predicate is given by the set of its instances. In WebDPF, the semantic of a predicate is defined by writing a validation method in JavaScript. The validation method is used to check if an instance of a model satisfies its atomic constraints. For each predicate, the tool produces a JavaScript object from the given arity. The language designer has to provide a validation method that will be used by the tool to determine if a constrained model is conforming to its metamodel or not.

Instances that violate the constraints are then highlighted in the model. In future we will incorporate an OCL validator for checking constraints. A model that does not satisfy its constraints is referred to as a partial model. An important observation regarding partial models is that in many situations partial models can be automatically repaired using model completion. In section 6 we provide a way to repair partial models by means of model transformations.

Predicates can be parameterized to support syntactic variation of model constraints. For instance, the multiplicity predicate $[mult(n,m)]$ has two parameters (see Table 1) to indicate the minimum and maximum cardinality of the instances of an edge. Two different variations of a multiplicity predicate have been used to constraint the model $\mathfrak{S}_1$ in Figure 1. The atomic constraint $(mult[1,1],\delta_2)$ on the edge *wardDeps* specifies that a *Ward* must be controlled by exactly one *Department* (R2); the atomic constraint

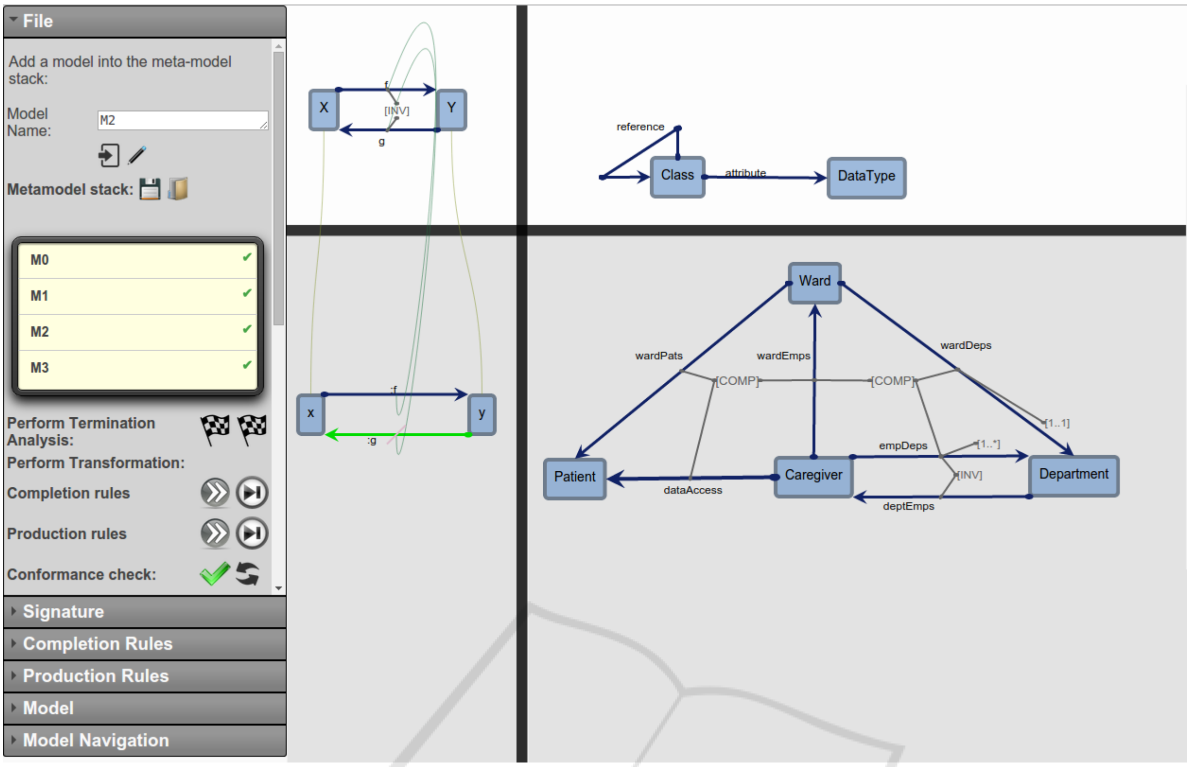


Figure 2: The WebDPF editor with a control panel (left), a metamodel viewer (top right), a model editor (bottom right), a signature editor (top left), and a rule editor (bottom left).

$([mult(1,\infty)],\delta_1)$ on the edge *empDeps* specifies that a *Caregiver* must work for at least one *Department* (R1). The main motivation behind using graphical signatures for constraining models is that they provide an abstraction to the modeller. Once a signature is defined by a language designer, the domain experts do not need to see the implementation giving the semantics of the predicates. Having an understanding of the intended semantics of the predicates suffices to use them for modelling.

4 MODEL NAVIGATION AND QUERY SUPPORT

Support for model navigation and query over modelling elements may come in handy when dealing with larger graphical models. In our approach, one can select a meta-model and a model from a metamodel stack to visualize the model elements in the WebDPF editor (Figure 2). WebDPF supports a query format that one can use to filter meta-model and model elements. Filter criterias can be specified over meta-model elements as well as model elements.
A query is built from eight tuples $(m_1, m_2, src_1, e_1, trg_1, src_2, e_2, trg_2)$ where

- m_1 denotes the meta-model,
- m_2 denotes the model,
- src_1 denotes a node in m_1,
- e_1 denotes an edge in m_1 with source src_1,
- trg_1 denotes the target of edge e_1 in m_1,
- src_2 denotes a node in m_2,
- e_2 denotes an edge in m_2 with source src_2,
- trg_2 denotes the target of e_2 in m_2,

If all eight parameters are specified, it means that the user is looking for nodes src_2 of type src_1 where src_2 has outgoing edges that matches with e_2 of type e_1 and e_2 has a target that matches with trg_2 of type trg_1. If any of the parameters from $src_1,e_1,trg_1,src_2,e_2,trg_2$ are not specified, a wildcard '%' is assumed for the query.

Figure 3 shows a screenshot of the WebDPF editor with a model and its instance. The model instance consist of three patients, two wards, six caregivers, two departments and the relations among them. It is evident from the diagram that for large models it is difficult to accomodate all modelling elements in a small viewing area. But in most cases, a modeller is interested in viewing only a small fragment of a model rather than the entire model. Let us consider

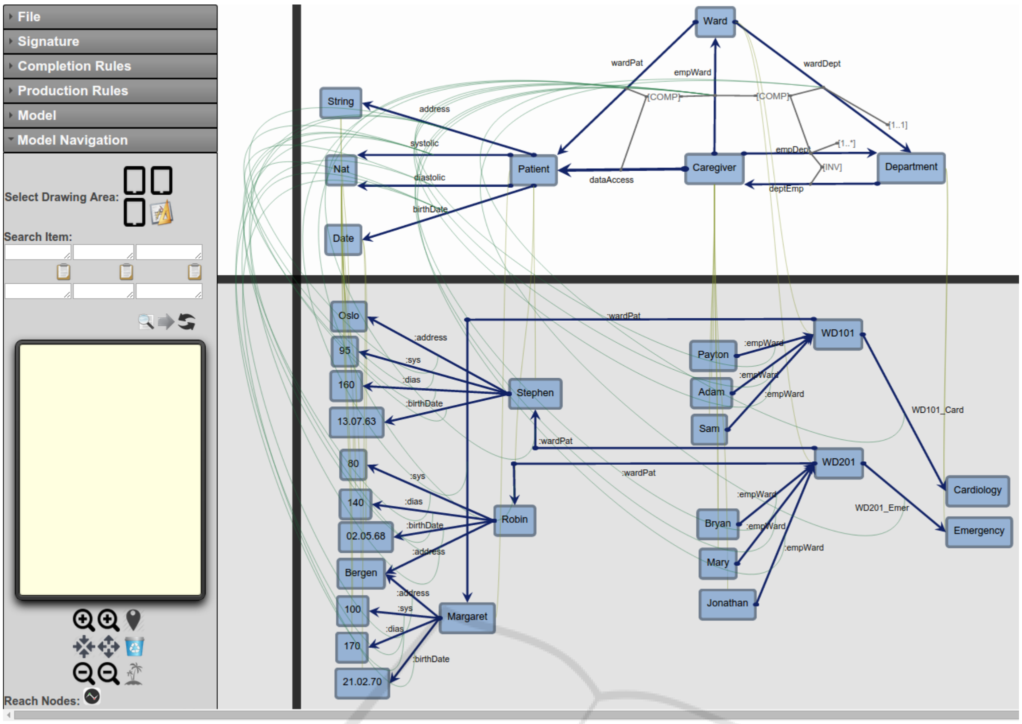


Figure 3: A large model and its instance that illustrates filtering.

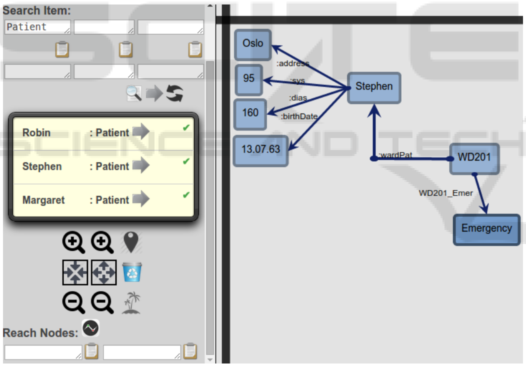


Figure 4: Query result.

that the modeller wishes to visualize the information related to the patient ‘Stephen’. The modeller can perform a query in WebDPF and can visualize the data related to the patient ‘Stephen’. Figure 4 shows the query on the left and the search result on the right in the WebDPF editor. Search parameters can be specified by either selecting model elements from the editor or by typing inside the *Search Item* text-fields. Query results are listed in a table where the model elements can be browsed in the editor. WebDPF provides the option for exploring and hiding nodes. The neighbouring nodes at a model element can be made visible in the editor or it can be made invisible with all its neighbour nodes. WebDPF also provides the option of searching for paths between model elements. The user can specify a source node and a target node, and investigate if there exists a path. This feature is useful to encode queries such as “Is there any patient in the Cardiology department who lives in Bergen”.

5 MODEL TRANSFORMATION

There are various applications of model transformations such as conversion of model elements and model fragments from one domain to another, model migration for meta-model evolutions (Taentzer et al., 2013), and behavioural modelling of systems (Rutle et al., 2012). In this article we use model transformations for representing the dynamic aspect of systems. WebDPF extends the DPF approach by attaching model transformation rules to predicates. The purpose of this extension is twofold: it aids the software designer to complete partial models and thereby reduce modelling effort, and it allows a modeller to reuse model transformation rules and design the behavior of a system using WebDPF.

Our diagrammatic model transformation system is based on graph transformation rules. The rules are linked to predicates and we use the standard double-pushout (DPO) approach (Ehrig et al., 2006) for defining transformation rules. We omit the technical details of the application of (coupled) transformation rules in this paper and provide an informal description instead.

Attached transformation rules for a predicate p, is given by a set of coupled transformation rules $\rho(p)$ where the meta-models remain unchanged. A rule $r \in \rho(p)$ of a predicate p has a matching pattern (L), a gluing condition (K), a replacement pattern (R), and an optional negative application condition, NAC $(n : L \rightarrow N)$ where L, N, K, R are all typed by the arity $\alpha(p)$ of the predicate p. The matching pattern and replacement pattern are also known as left-hand side and right-hand side of a rule, respectively. Figure 5 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.

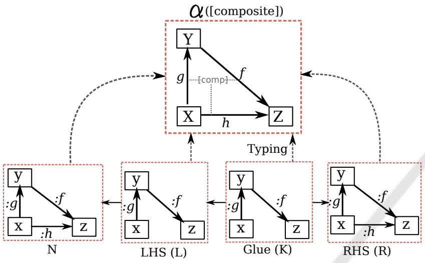


Figure 5: A transformation rule linked to the $[composite]$ predicate.

Since the rules are not directly linked to a model, they can be reused by constraining a model with appropriate predicates. A match (δ, m) of a rule is given by an atomic constraint $\delta : \alpha(p) \rightarrow S_{i-1}$ and an injective morphism $m : L \rightarrow S_i$ such that constraint δ and an injective morphism m together with typing morphisms $\iota_L : L \rightarrow \alpha(p)$ and $\iota_{S_i} : S_i \rightarrow S_{i-1}$ constitute a commuting square: $\iota_L;\delta = m;\iota_{S_i}$. This is illustrated in the following diagram.

$$\begin{tikzcd}
\text{Arity } \alpha(p) \arrow[r, "constraint{,}\delta"] \arrow[dr, phantom, "="] & \text{Graph } S_{i-1} \text{ of } \mathfrak{S}_{i-1} \\
\text{LHS (L)} \arrow[u, dashed, "\iota_L"] \arrow[r, "injective-morphism{,}m"] & \text{Graph } S_i \text{ of } \mathfrak{S}_i \arrow[u, dashed, "\iota_{S_i}"']
\end{tikzcd}$$

Figure 6 shows the left hand side of a rule attached to the $[composite]$ predicate and its match is given by a constraint and an injective morphism. A rule is applied as long as it satisfies its negative application condition (NAC) which is typically expressed by a graph structure in a graph transformation system. Negative application conditions are typically used in graph transformation to prohibit an infinite number of rule applications. A rule transforms a model (S_i, ι_{S_i}) to $(S_i^*, \iota_{S_i^*})$ if there exists a match (δ, m) where (δ, m) satisfies the NAC.

WebDPF supports the execution of rules in two different ways. One can apply the rules interactively

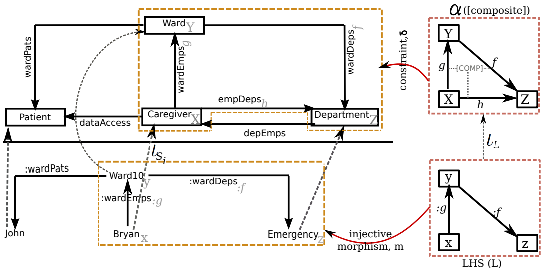


Figure 6: An example match (δ, m) of a rule.

where the user guides the execution order; or automatically considering all possible orderings of rule execution. If the rules are applied automatically, then it is important to perform an analysis of the termination of the transformation system.

WebDPF performs termination analysis based on principles adapted from layered graph grammars (Ehrig et al., 2006). 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. WebDPF generalizes the layer conditions from (Ehrig et al., 2006) allowing deleting and non-deleting rules to reside in the same layer as long as the rules are loop-free. Furthermore, it permits a rule to use newly created edges allowing us to perform transitive closure operations (Levendovszky et al., 2007). A loop detection algorithm is implemented in WebDPF that overestimates the existance 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. In order to be loop free, all the rules in R_k must satisfy the following conditions:

1. 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,
2. 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,

Informally, a rule r_i that creates an element x of type t, becomes disabled as soon as the element x is created according to condition (1). Therefore rule r_i cannot execute in a loop unless a rule r_j deletes the element x. Deletion of the element x would enable rule r_i. This is however impossible because of condition (2). So the bottomline is: rules that create and delete an element x of type t cannot be executed in the same layer. Note that we are assuming, a finite number of rules in each layer and that the rules are applied on a finite input graph. The algorithm in WebDPF guarantees termination if all the rules for each layer satisfies the above mentioned conditions. WebDPF provides a warning to the modeller if it detects a potential loop.

6 MODEL COMPLETION

A diagrammatic approach to model completion were proposed in (Rabbi et al., 2015a) where the completion rules defined in a metamodel are used to complete a partial model. As an example, consider the requirements R1-R5 from section 3. Figure 7(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 1(a). Figure 7(b) shows a candidate instance $\iota : I \rightarrow S$ of $\mathfrak{S}_1$ (also represented as (I,ι)) where S is the graph of specification $\mathfrak{S}_1$. In this example, Bryan works at Ward10; Ward10 is controlled by the Emergency department. Even though I is typed by S, (I,ι) is not a valid instance of $\mathfrak{S}_1$ since it does not satisfy the $[composite]$ and $[mult(1,\infty)]$ constraints.

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 $\iota : I \rightarrow S$ of $\mathfrak{S}_1$. Figure 8 shows a screenshot of the WebDPF editor that highlights which part of the model instance violates the constraint.

In the above example, the partial model (I,ι) can be automatically repaired by adding missing arcs such as the missing arc from Bryan to the Emergency department which is required in order to satisfy the constraints and thereby conform to the metamodel $\mathfrak{S}_1$. WebDPF improves the productivity of the modeller by providing editing support that either automatically creates such missing model elements or make suggestions 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 is reduced by providing editing support for automated rewriting of models so that they conform to the modelling language used.

We augment the above mentioned example with completion rules at the meta-model level that can be used as a basis for automatically fixing the inconsistencies of the partial model (I,ι). Table 3 shows completion rules linked to predicates. The semantics of completion rules are defined by coupled transformation rules (Schulz et al., 2011) (Becker, 2008) with negative application conditions as in Table 3. 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.

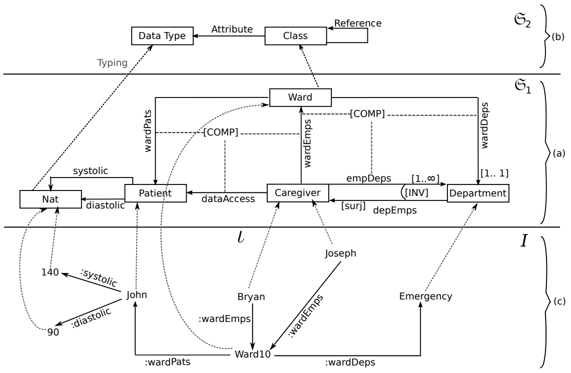


Figure 7: a) A DPF model $\mathfrak{S}_1 = (S, C^{\mathfrak{S}_1} : \Sigma_2)$, and b) an instance (I,ι) of $\mathfrak{S}_1$.

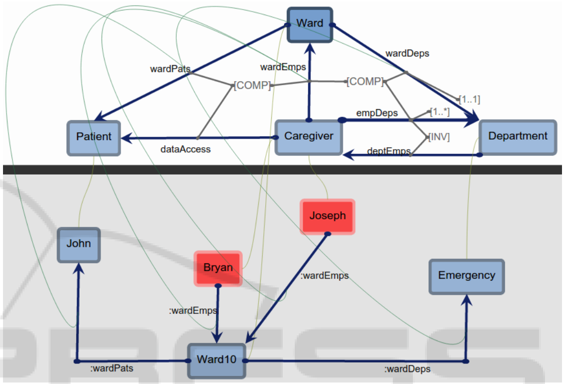


Figure 8: An inconsistent instance (I,ι) of $\mathfrak{S}_1$.

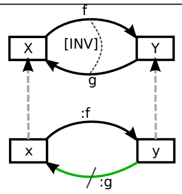
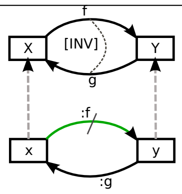
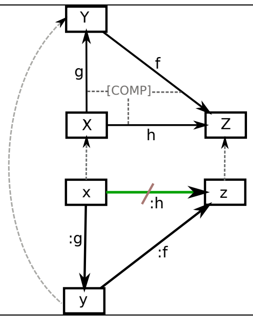
By applying the completion rules from Table 3 on (I,ι) from Figure 8, the partial model is updated to become a model, (I^*,ι) which conforms to its metamodel.

Once we apply the completion rules for $[inverse]$ and $[composite]$ on (I,ι), we obtain the instance of specification $\mathfrak{S}_1$ shown in Figure 9. Thick arcs (also shown with a green color) represent the additional edges derived from the application of completion rules.

7 BEHAVIORAL MODELLING

In order to illustrate the general application of transformation systems as supported by WebDPF, we model the behavior of a system by developing a multilevel description of a workflow modelling language. We use coupled transformation rules with negative application conditions for developing a diagrammatic transformation systems in WebDPF. Since the purpose of these transformation rules are to build a transformation system, they are referred to as production rules.There is no fundamental difference be-

Table 3: Completion predicates and rules of $\mathfrak{S}_1$.

p	$\rho(p)$	Interpretation
[inverse]	 	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.
[composite]		derive an edge $x \xrightarrow{:h} z$ (if it does not exist) from the existence of edges $x \xrightarrow{:g} y$ and $y \xrightarrow{:f} z$.

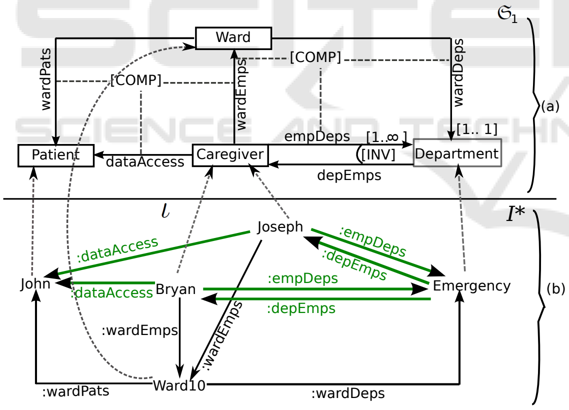


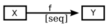
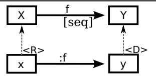
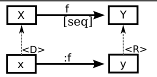
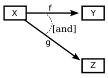
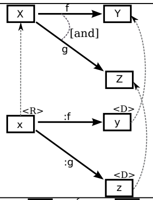
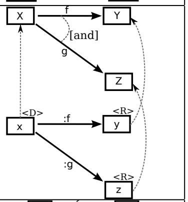
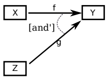
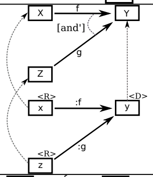
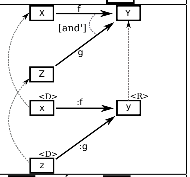
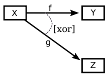
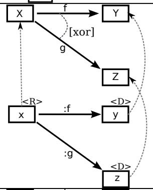
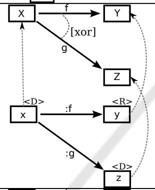
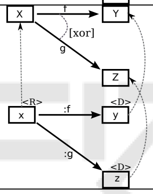
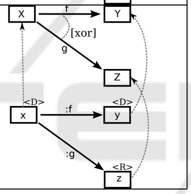
Figure 9: a) A specification $\mathfrak{S}_1 = (S, C^{\mathfrak{S}_1} : \Sigma_0)$ and b) an instance (I^*, ι) of $\mathfrak{S}_1$.

tween production rules and completion rules other than their purpose. Similar to completion rules, production rules are also attach to predicates and therefore they can be reused by binding predicate structures to the model elements. This essentially means that a modeller can graphically design rules and can use them as a library of rules. This feature is useful for building model transformation languages.

Rutle et al. proposed a metamodelling approach for behavioral modelling in (Rutle et al., 2012) and developed a workflow modelling langauge named DERF using coupled model transformation rules. In this section we generalize the concepts of DERF and show that domain specific languages for workflow modelling such as DERF can be constructed using WebDPF. In general, a workflow or process modelling language requires different types of routing operators to control the execution flow among processes. The state of a workflow is determined by the states of task instances (i.e., processes) in DERF. Each task instance is annotated with a predicate from [*running*] and [*disabled*] (in short <*R*>, <*D*>, resp.) to indicate its state. A task instance annotated with the predicate [*running*] indicates that the task instance is in the running state and may transit to the disabled state (e.g., become annotated with [*disabled*]) when it is finished its execution. Table 4 shows a set of selected routing operators and their production rules (adapted from (Rutle et al., 2012)) that can be used to construct a workflow model. The execution flow of a workflow instance is controlled by the production rules.

Figure 10 shows an overview of a hypertention management workflow model $\mathfrak{S}_1$ with its meta-model $\mathfrak{S}_0$ and an instance (I, ι) of $\mathfrak{S}_1$. We have formalized this workflow from a clinical guide for hypertension management (Chi, 2006). The workflow model in Figure 10 is however shows an abstract process model which requires further enhancement with pre/post conditions (Rabbi and MacCaull, 2014) (Rabbi et al., 2015b). All the nodes (and edges) in $\mathfrak{S}_1$ are typed by the 'Task' (and the 'Flow' respectively) from $\mathfrak{S}_0$. Routing predicates from Table 4 have been used to constraint the workflow model $\mathfrak{S}_1$. In the instance I, we use $: X$ to denote a typing morphism $\iota : x \to X$. Initially patients' blood pressure (BP) is measured at the ':Initial BP Measure' task which may prompt the clinical hypertension management procedures. If the initial BP is found normal, the workflow terminates. Otherwise the clinical procedure (i.e., ':Visit1', ':Visit2', and all other subsequent task instances in Figure 10) starts to manage patients' hypertension. The production rules attached to the [*xor_split*] predicate ensures that when the ':Check Eligibility' task instance is finished running, either ':Visit1' or ':End' task instance transits to the running state. Patients with high BP have risk of organ failures and/or other chronic illness. During the first visit at the doctors office ('Visit1') the BP is measured twice, an initial assessment is done, and an investigation is started with diagnostic tests. During ':Visit2' BP is measured, diagnosis is performed and followup visits are scheduled. The workflow executes ':Ambulatory blood pressure monitoring' (':ABPM') or ':Clinical BPM' (':CBPM') depending on the availability. ':ABPM' and ':CBPM' tasks are rescheduled if required otherwise the workflow terminates.

Table 4: Production rules for workflow modelling.

p	Visualization	$\rho(p)$	
		LHS	RHS
[sequence]			
[and_split]			
[and_merge]			
[xor_split]			
			

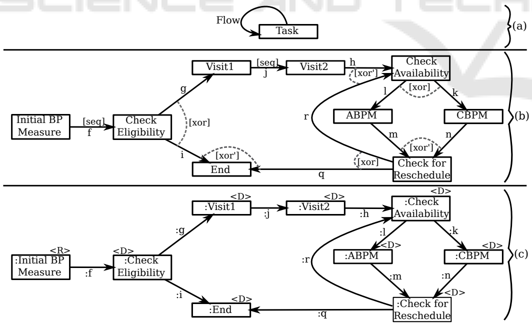


Figure 10: (a) A workflow meta-model $\mathfrak{S}_0$, (b) a workflow model $\mathfrak{S}_1$, and (c) an instance (I,ι) of $\mathfrak{S}_1$.

The behavior of a DERF workflow is determined by the set of states reachable from an initial state. The initial state for the workflow model $\mathfrak{S}_1$ is in this situation $s_0 := (I,\iota)$ which represents the start state of the system. Other states are reached by applying production rules from Table 4. At state s_0, the production rule attached to the [*sequence*] predicate is applicable and the transition $s_0 \xRightarrow{\rho[sequence]} s_1$ is given by an application of production rule. At state s_1 (see Figure 11), ':Initial BP Measure' is annotated with the predicate [*disabled*] and ':Check Eligibility' is annotated with the predicate [*running*]. At state s_1, two production rules attached to the [*xor_split*] predicate are applicable giving two choices. For each choice we get a different execution path (to s_2 or s_3) for a transformation system which is shown in Figure 11.

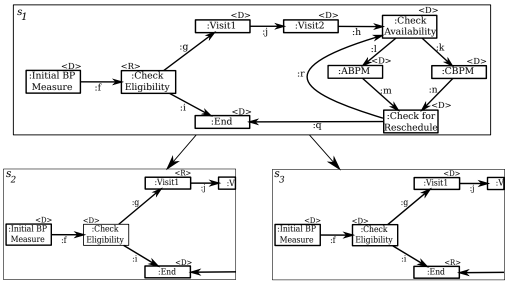


Figure 11: Two different execution paths from state s_1.

8 RELATED WORK

In this section we compare the features of WebDPF to related works found in the literature. Table 5 provides an overview comparing WebDPF features with other modelling tools.

WebGME is a web and cloud-based scalable metamodelling tool (Maróti et al., 2014) that supports the development of domain specific modelling languages (DSML). Online collaboration is the main focus of the WebGME tool. In WebGME, changes are immediately broadcast to all the client machines and every modeller sees the same state. WebGME provides a visualization toolkit that the DSML designer can efficiently use to control the visualization aspects of the concrete syntax. WebGME supports filtered views showing only a subset of the elements of the model, but a query functionality on modelling elements is missing. AToMPM is another web-based modelling framework for designing DSMLs, performing model transformations, manipulating and managing models (Syriani et al., 2013).

The tool features collaborate development and share modeling artifacts among users. Even though WebGME and ATomPM have several advantages over desktop based (meta)modeling tools they have limitations on defining constraints and transformation rules in comparison to WebDPF.

Metamodelling in WebDPF is similar to the modelling environment of Melanie (Kennel, 2012) as the metamodels and models are displayed in a single view. In the Melanie editor, all metamodels are open at the same time focusing on the whole ontology at a time. This perspective works well as long as the models are small. This approach does not scale to larger

Table 5: Comparison of WebDPF features with other modelling tools.

Tool	Multilevel metamodelling	Web-based UI	Constraint language	Model navigation (query facility)	Rule Reusability	Model completion	Termination analysis
DIAGEN						✓	
WebGME	✓	✓	OCL				
AToMPM	✓	✓	JavaScript				
Melanie	✓		OCL				
AGG			Graph constraints				✓
eMoflon	✓		OCL				✓
DPF-WB	✓		OCL, Java, Alloy				
WebDPF	✓	✓	JavaScript	✓	✓	✓	✓

models as it becomes difficult to get an overview of the system. WebDPF overcomes this problem in two ways: it only shows two levels of metamodels at the same time instead of the entire metamodel stack, and it supports filtering over the displayed items via a model naviagation facility. Although Melanie comes with an adapter for the ATL transformation language (Atkinson et al., 2013), it does not provide support for designing transformation rules graphically. Also, the Melanie tool is not web-based.

AGG is a visual tool for algebraic graph transformation (Ehrig et al., 2006). The tool can also be used as a graph transformation engine for Java applications. Layered graph transformation system can be developed with AGG and the tool supports various analysis techniques, such as critical pair analysis, consistency checking, and facilitating validation of graph transformations. eMoflon (Anjorin et al., 2011) 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. WebDPF utilizes many concepts and theories of AGG and provides a web-based tool for metamodelling. Transformation rules are attached to predicates in WebDPF that gives the modeller opportunity to reuse them by constraining the predicates inside the models. The query feature to manage large models in the WebDPF editor is also unique. AGG, eMoflon and other model transformation tools such as Henshin (Arendt et al., 2010) do not have these features.

Mazanek et al. presented an editor generator framework called DIAGEN that is based on the graph grammar approach (Mazanek et al., 2008). The framework is used for graph completion and the technique has been applied to realize editors for Nassi-Shneiderman diagrams, flowcharts, and trees. However, the framework does not support features that has been presented in this paper such as multilevel metamodelling, predicate constraints, and behavioral modelling.

Work similar to our was also presented in (Sen et al., 2007) 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. In our situation, we are given a model specification with a metamodel, a set of predicate constraints, a constrainted partial model instance, and a set of completion rules. In WebDPF, all these artefacts are diagrammatically specified. We use completion rules to derive a complete model instance. Another possible application of our 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 (Taentzer et al., 2013). The concept of partial models is applicable also in this setting if accompanied by completion rules and would potentially be able to automate the model migration process.

9 CONCLUSIONS

In this paper we have introduced the WebDPF tool which is based on a rigorous mathematical foundation in the form of DPF. The main features of WebDPF are: a model navigation facility, a JavaScript editor for writing predicate semantics, partial model completion, reusability of transformation rules, and termination analysis. A prototype implementation of WebDPF may be found from *http://dpf.hib.no* where one can graphically design modelling artefacts in a web browser and apply the transformation rules.

We are currently investigating the metamodelling approach for modelling distributed systems. We are developing a prototype of a blood transfusion application in collaboration with the local health authority. The goal of the project is to develop an application for safe blood transfusion. We are modelling the process flow and developing the data models as well as modelling the interaction between different subsystems. WebDPF has been used for modelling the system and simulating the process flow. Initial results have been published in (Rabbi et al., 2015b) where we provide different level of abstraction for modelling.

REFERENCES

(2006). *Hypertension. Building Healthy Lifestyles.* www.chinookprimarycarenetwork.ab.ca/ extranet/docs/guides/7.pdf.

Anjorin, A., Lauder, M., Patzina, S., and Schürr, A. (2011). eMoflon: Leveraging EMF and Professional CASE Tools. In *3. Workshop Methodische Entwicklung von Modellierungswerkzeugen (MEMWe2011)*.

Arendt, T., Biermann, E., Jurack, S., Krause, C., and Taentzer, G. (2010). Henshin: Advanced Concepts and Tools for In-place EMF Model Transformations. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part I*, MODELS'10, pages 121–135. Springer-Verlag, Berlin, Heidelberg.

Atkinson, C., Gerbig, R., and Tunjic, C. (2013). Enhancing classic transformation languages to support multi-level modeling. *Software & Systems Modeling*, pages 1–22.

Atkinson, C. and Kühne, T. (2001). The essence of multi-level metamodeling. In *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, UML'01, pages 19–33. Springer-Verlag.

Barr, M. and Wells, C., editors (1995). *Category Theory for Computing Science, 2nd Ed.* Prentice Hall International (UK) Ltd., Hertfordshire, UK.

Becker, S. (2008). Coupled model transformations. In *Proceedings of the 7th International Workshop on Software and Performance*, WOSP '08, pages 103–114.

Diskin, Z. and Wolter, U. (2008). A diagrammatic logic for object-oriented visual modeling. *Electronic Notes in Theoretical Computer Science*, 203(6):19 – 41. Proceedings of the Second Workshop on Applied and Computational Category Theory (ACCAT 2007).

Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. (2006). *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer.

Kennel, B. (2012). *A Unified Framework for Multi-Level Modeling*. PhD thesis, University of Mannheim.

Lamo, Y., Wang, X., Mantz, F., MacCaull, W., and Rutle, A. (2012). DPF Workbench: A diagrammatic multi-layer domain specific (meta-)modelling environment. In Lee, R., editor, *Computer and Information Science 2012*, volume 429 of *Studies in Computational Intelligence*, pages 37–52. Springer.

Levendovszky, T., Prange, U., and Ehrig, H. (2007). Termination criteria for dpo transformations with injective matches. *Electr. Notes Theor. Comput. Sci.*, 175(4):87–100.

Macedo, N., Tiago, J., and Cunha, A. (2015). A feature-based classification of model repair approaches. *CoRR*, abs/1504.03947.

Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levendovszky, T., and Lédeczi, Á. (2014). Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure. In *Proceedings of the 8th Workshop on Multi-Paradigm Modeling, MPM@MODELS 2014*, pages 41–60.

Mazanek, S., Maier, S., and Minas, M. (2008). Auto-completion for diagram editors based on graph grammars. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 242–245.

Mens, T. and Van Der Straeten, R. (2007). Incremental resolution of model inconsistencies. In *Recent Trends in Algebraic Development Techniques*, volume 4409 of *LNCS*, pages 111–126. Springer.

Rabbi, F., Lamo, Y., Yu, I., and Kristensen, L. (2015a). A diagrammatic approach to model completion. In *Proceedings of the 4th Workshop on the Analysis of Model Transformations co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015)*, volume 1500 of *CEUR Workshop Proceedings*, pages 56–65. CEUR-WS.org.

Rabbi, F., Lamo, Y., Yu, I. C., and Kristensen, L. M. (2015b). Towards a Multi Metamodelling Approach for Developing Distributed Healthcare Applications. . *NIK: Norsk Informatikkonferanse, ISSN: 1892-0721*.

Rabbi, F. and MacCaull, W. (2014). User-Friendly UIs for the Execution of Clinical Practice Guidelines. In *2014 IEEE 27th International Symposium on Computer-Based Medical Systems, 2014*, pages 489–490. IEEE Computer Society.

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

Rutle, A., MacCaull, W., Wang, H., and Lamo, Y. (2012). 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. ACM.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. 39(2):25–31.

Schulz, C., Löwe, M., and König, H. (2011). A categorical framework for the transformation of object-oriented systems: Models and data. *J. Symb. Comput.*, 46(3):316–337.

Sen, S., Baudry, B., and Precup, D. (2007). Partial model completion in model driven engineering using constraint logic programming. In *INAP'07*, Germany.

Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S. V., and Ergin, H. (2013). AToMPM: A web-based modeling environment. In *Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session,* *Poster Session, and ACM Student Research Competition*, volume 1115, pages 21–25. CEUR-WS.org.

Taentzer, G., Mantz, F., Arendt, T., and Lamo, Y. (2013). Customizable model migration schemes for metamodel evolutions with multiplicity changes. In *16th International Conference, MODELS 2013*, volume 8107 of *LNCS*, pages 254–270. Springer.

Tomassetti, F., Torchiano, M., Tiso, A., Ricca, F., and Reggio, G. (2012). Maturity of software modelling and model driven engineering: A survey in the italian industry. In *16th International Conference on Evaluation & Assessment in Software Engineering, EASE 2012*, pages 91–100. IET - The Institute of Engineering and Technology / IEEE Xplore.

Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., and Heldal, R. (2013). Industrial adoption of model-driven engineering: Are the tools really the problem? In *16th International Conference, MODELS 2013*, volume 8107 of *LNCS*, pages 1–17. Springer.