Towards Rigorous Metamodeling

Benoît Combemale[1], Sylvain Rougemaille[1], Xavier Crégut[1], Frédéric Migeon[1], Marc Pantel[1], Christine Maurel[1] and Bernard Coulette[2]

[1] FéRIA-IRIT-LYRE
118, route de Narbonne
F-31062 Toulouse Cedex 9

[2] GRIMM ISYCOM
5, allée Antonio Machado
F-31058 Toulouse Cedex 9

Abstract. MDE has provided several significant improvements in the development of complex systems by focusing on more abstract issues than programming. However, improvments are needed on the semantic side in order to reach high-level certification such as the one currently required for critical embedded systems (which will also probably be required in the near future for Information Systems as application of Basel II kind of agreements). This paper presents different means to specify models semantics at the metamodel level. We will focus on the definition of executable SPEM-based development process models (workflow related models) using an approach defined for the TOPCASED project.

1 Introduction

Model-Driven Engineering (MDE) has succeeded in establishing a new, more abstract, approach to large scale system development. A system can be described using many different models which are related to each other using model transformations. The key point is to use as many different models as life-time or technology aspects in the system. The main difference with programs is that model focus on the abstract syntax whereas programs focus on the concrete syntax. A single model can then be represented using different graphical or textual concrete syntaxes. For data-centred systems, the level of abstraction thus provided led to significant improvements and seems to correspond to an adequate semantic level. For computation-centred systems, further steps are required in order to give a precise enough account of the dynamic aspects of the models.

This contribution gives some insights on approaches for defining metalevel model semantics derived from the work done by the programming language community. The evocated experiments take place in the TOPCASED project [1] whose purpose is to define and implement a MDE-centred CASE tool for critical embedded software and hardware systems. The certification authorities for TOPCASED application domain (aeronautic, space, automotive...) require high quality system validation approaches which are currently based on formal tools. Currently, Information Systems do not require this

Combemale B., Rougemaille S., Crégut X., Migeon F., Pantel M., Maurel C. and Coulette B. (2006).
Towards Rigorous Metamodeling.
In *Proceedings of the 2nd International Workshop on Model-Driven Enterprise Information Systems*, pages 5-14
DOI: 10.5220/0002501500050014

kind of certification. However, we can guess that, in the near future, the software developed for Basel II level IS will also follow this level of requirements.

The TOPCASED toolkit aims at easing the definition of new DSL or modeling languages by providing metalevel technologies such as concrete syntax (both textual and graphical) editor generators, static validation and dynamic execution of models. This contribution will describe how semantic considerations designed for programming languages can be integrated in the MDE approach. We will focus on one of the available technologies for creating executable models; the other ones will be reported in forthcoming publications.

As an example, we apply our proposal for the modeling of a very simplified process description language (PDL). This PDL provides the concept of processes (*Process*) composed of activity (*Activity*) sequences representing the various tasks which must be realized during the development. These activities may be connected using a relation of precedence (*Precedes*) which makes it possible to indicate a partial order *start-to-start*, *finish-to-start* and *finish-to-finish* (*PrecedenceKind*). This kind of example is very close to the workflow-based modeling of IS.

2 Syntax in Metamodeling

2.1 Abstract Syntax Definition

The abstract syntax of a modeling language is the structural expression of its concepts and the relations which bind them. Metamodeling languages such as the OMG standard MOF (Meta Object Facility) [2], provide sets of elementary entities and relations in which terms we can describe our own metamodel. Nowadays, the definition of this syntax is well mastered and supported by many metamodeling environments (Eclipse/Ecore [3], GME/MetaGME [4], AMMA/KM3 [5] and XMF-Mosaic/Xcore [6]).

To describe the abstract syntax of our SimplePDL, we use the Ecore editor from the TOPCASED project. It is a graphical editor that allows the description of the abstract syntax. For SimplePDL, we draw the metamodel of figure 1. *Process*, *Activity* and *Precedes* are instances of the EClass metaclass of Ecore. Their characteristics, such as, e.g., *name*, are described as EAttribute and their relationships are defined as EReference. This metamodel will be used as a basis for the various experiments. First of all, we would like to have a concrete syntax to be able to define models conforming to SimplePDL.

2.2 Concrete Syntax Design

Concrete syntax provides a formalism, graphical or textual, to handle concepts of the abstract syntax and thus to describe “instances” of the abstract syntax. The definition of ad hoc concrete syntaxes is well mastered, indeed many projects exist for this purpose which are mainly built upon EMF (Eclipse Modeling Framework) : GMF[3], Merlin

[3] Generic Modeling Framework, `http://www.eclipse.org/gmf/tutorial/`

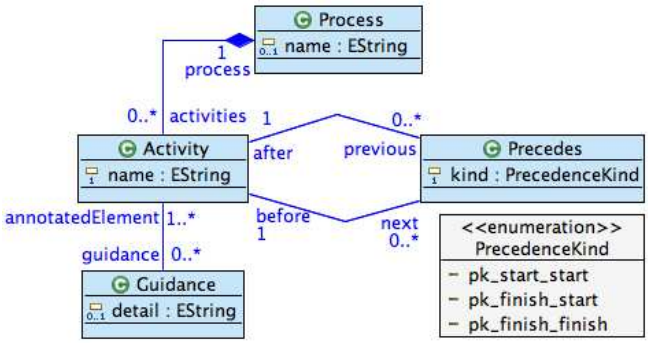


Fig. 1. The Ecore metamodel of our Simple PDL.

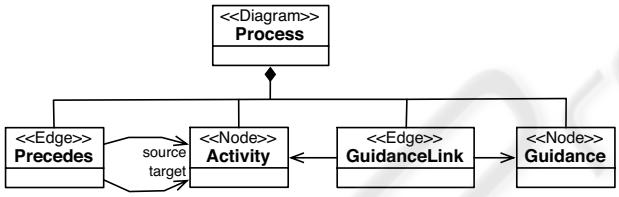


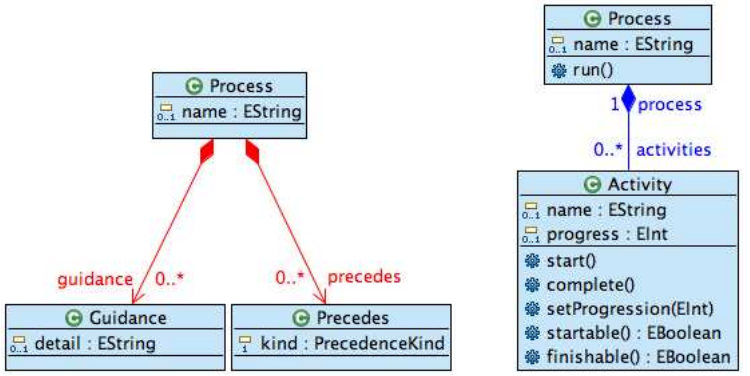
Fig. 2. Configurator model.

Generator[4], GEMS[5], TIGER [7], etc. The current challenge aims at being able to generate automatically a concrete syntax from an already defined abstract syntax [4, 6]. This generative approach, in addition to its generic qualities, would allow to standardize the construction of concrete syntaxes.

The TOPCASED environment offers a tool called "graphical editor generator" that allows to define a graphical concrete syntax and the associated editor for an Ecore model. The generation process takes place after the generation of the textual syntax (XML) provided by EMF [3]. It is based on the definition of the items of the graphical formalism (concrete syntax) and its mapping to the base Ecore model (abstract syntax). These two things are described in the *configuration model* (*configurator*) that offers strong possibilities for the personalization of this concrete syntax.

For SimplePDL, we first defined the model of our concrete syntax (figure 2). *Activity* and *Guidance* are defined as *Node* (boxes). *Precedes* is defined as an *Edge*, a connection between two nodes. *Process* is represented as a *Diagram*, which is a package that will contain the other items. The concrete syntax may need additional items that do not correspond to any abstract concept. For example, we need to add *GuidanceLink* as an *Edge* to connect a *Guidance* to the descrided *Activity*. *GuidanceLink* do not correspond to any concept of SimplePDL but is required to link a guidance to an activity (EReference named *guidance* on the base metamodel, figure 1). Please note that concepts of abstract syntax (figure 1) and concepts of concrete syntax (figure 2) are different con-

[4] `http://sourceforge.net/projects/merlingenerator/`
[5] Generic Eclipse Modeling System, `http://sourceforge.net/projects/gems/`



(a) Concrete syntax (b) Operational semantics

Fig. 3. Extensions of the SimplePDL abstract syntax.

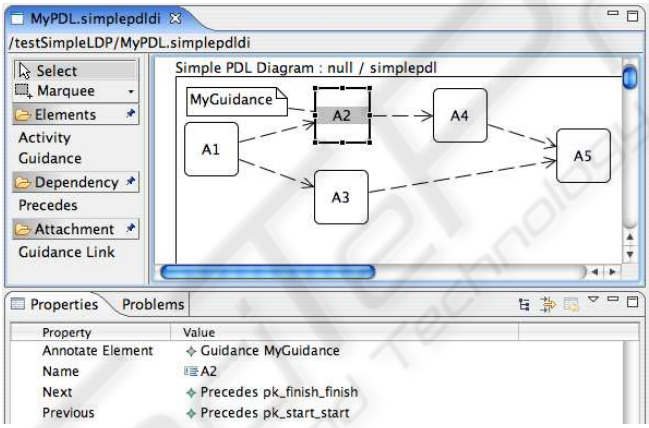


Fig. 4. Generated editor to provide a graphical concrete syntax to SimplePDL.

cepts that have to be mapped to each other. We used the same name when the mapping was obvious.

To be able to use the TOPCASED editor generator, we had to extend our abstract syntax (fig. 1) to add two containment references, one between *Process* and *Guidance*, and the other between *Process* and *Precedes* (fig. 3a). This is required to be able to put the corresponding graphical items (*Activity* and *Guidance*) in the *Process* package.

Figure 4 shows the generated editor. All the concepts of the configurator model are available using the palette. Clicking on a palette item and adding it on the diagram, creates a graphical feature (node, edge) and instantiates the corresponding SimplePDL metaclass according to the configurator model.

3 Semantics in Metamodeling

In the scope of MDE there are currently a lot of languages and techniques to define the abstract and concrete syntax of a modeling language. However, these techniques do not take into account the description of the precise meaning of the concepts provided by a modeling language. Consequently, the semantics of these languages has to be defined by the use of additional techniques allowing to enrich their abstract syntax. These problems have previously been handled by the programming language community. It is not surprising that the same approaches can also be used here. We can separate modeling language semantics into three categories (as defined for programming languages): axiomatic, operational and denotational (left for further work). Each of them can be applied at different levels.

3.1 Axiomatic Semantics

The axiomatic semantics is based on mathematic logic and allows to define correctness for the use of programming language constructions. The principle is to define axioms and deduction rules to express the meaning of such constructions according to invariants, pre and post-conditions. Thus it gives the precise semantics of every program written in this language along with a correctness proof scheme. In a model-driven approach, we restrict this semantics to models static analysis, which allow us to check the correctness of models structure. This vision of axiomatic semantics can be added by means of Well-Formed-Rules (WFR), which are expressed on the metamodel and have to be respected by the models. The OMG recommends the use of OCL (Object Constraint Language) [8] for the expression of WFR on metamodels. The metamodel WFR can be seen as a mean to reduce the number of valid models. Thus, one can use OCL checkers (e.g., Use [9]) to verify the correctness of a model in accordance with each of the WFR expressed on its metamodel.

One can also check whether a model satisfy its WFR or not by means of a declarative transformation language such as ATL (Atlas Transformation Language) [10]. The idea is to define transformation rules that match errors and generate a diagnostic model containing much more details than the Boolean return value of an OCL checker. The details concerning the errors depend on the diagnostic metamodel. This technique has been proposed by the ATLAS team and carried out thanks to ATL [11]. ATL transformation rules are defined to detect the negation of a WFR and generate the corresponding diagnostic model (fig. 5).

In the scope of our language (SimplePDL), several constraints have to be defined to guarantee the consistency of the models which conform to the metamodel. As an example we proposed the following rules :

“ An activity must not precede itself ” :

```
context Precedes inv :
       self.before <> self.after
```

“ A process must not contain two activities with the same name ” :

```
context Process inv :
       self.activities->forAll(a1, a2 : Activity |
                        a1 <> a2 implies a1.name <> a2.name)
```

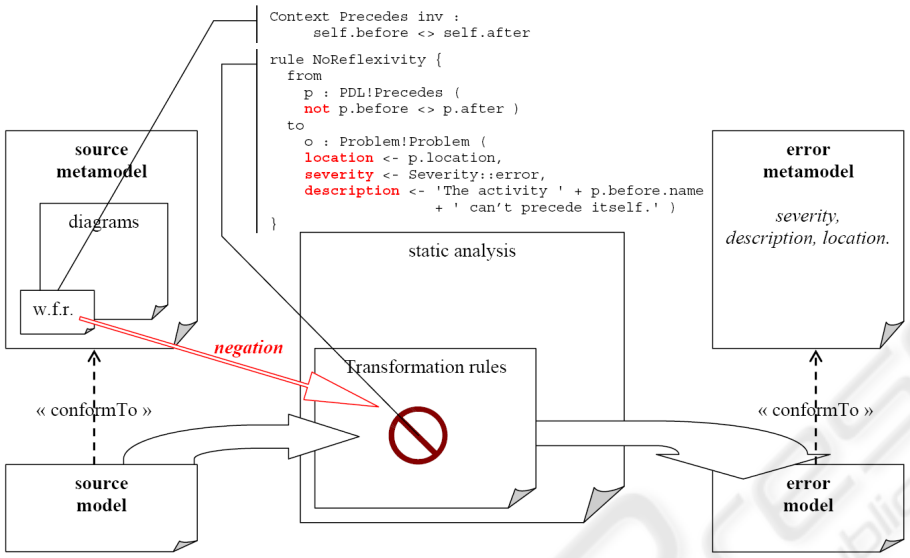


Fig. 5. Checking model with ATL.

3.2 Operational Semantics

The operational semantics allows to precisely describe the dynamic behavior of the constructions of a language. In MDE, it aims to express the behavioral semantics of a metamodel and thus build executable conforming models. For this purpose, two approaches are available. First of all, the one which is closer to the operational semantics in programming languages consists in the definition of transformations between two execution states of a model. The whole set of transformations, written in conformance to the metamodel, defines the behavior of models. The second one consists of describing the behavior of each concept of the metamodel in an imperative way using metaprogramming languages such as Kermeta [12], xOCL [6] or an action language such as AS-MOF [13].

Our first experimentation is related to Kermeta which is defined as an executable metamodeling language, or as an object oriented metaprogramming language, i.e., it allows to describe metamodels whose models are executable. Kermeta relies on the Ecore metamodeling language, it has been defined as a ”weaving” between a behavioral model and the Ecore metadata model [12]. The Kermeta metamodel is composed of two packages. The first one called *core* corresponds to Ecore. The second one called *behavior* is built as a metaclass hierarchy representing the expressions that constitute the body of the *operation* features defined in the *core* package. Thus, Kermeta allows to specify the structure of a metamodel as well as its behavior.

Kermeta is integrated as a plug-in to the Eclipse IDE, and it provides a generation tool *Ecore2Kermeta* which has allowed us to translate our SimplePDL metamodel (fig. 1) to a Kermeta version. This version of our metamodel has been used as a basis

for the programming of the SimplePDL models behavior. In order to code this behavior, we have had to define precisely what is the execution of a SimplePDL model.

A *Process* is composed of *Activities*, which goes through different states during the enactment of the *Process*: not started, in progress and completed. In order to represent those states we have added the *progress* attribute to the *Activity Eclass*. Thus, its progression rate value corresponds to its three possible states : -1: not started ; [0..99]: in progress and 100: complete. A *Process* have been executed when all the contained activities are completed. The behavior of our SimplePDL *process* consists of authorizing users to set the values of activities progression rate, according to *precedes* relation, until they are all finished. The handling of the progression rate and the *precedes* link for each *Activity* implies the extension of our metamodel (fig. 1) in order to add the necessary operations (fig. 3b). Thus, the execution of SimplePDL processes was implemented in Kermeta as a loop proposing to the user the following choices :

– *Stop the process execution:* Quit the loop.

– *Start an enactable activity:* One selects the activity which can be started. An activity can start if the *startable* operation returns *True*, i.e., if it is an initial one, or if its preceding activities and the *Precedes* link which bind them to it allows to.

```
operation startable() : Boolean is do
    var start_ok : kermeta::standard::Boolean
    var previousActivities : seq Activity [0..*]
    var prevPrecedes : seq Precedes [0..*]

    if progress==-1 then
        // Getting the activities which have to be started
        prevPrecedes := previous.select{p | p.kind ==
                                    PrecedenceKind.pk_start_start }
        previousActivities := prevPrecedes.collect{p | p.before}
        start_ok := previousActivities.forAll{a | a.progress >= 0}
        // Getting the activities which have to be finished
        prevPrecedes := previous.select{p | p.kind ==
                                   PrecedenceKind.pk_finish_start }
        previousActivities := prevPrecedes.collect{p | p.before}
        start_ok := start_ok and
                    (previousActivities.forAll{a | a.progress==100})
        result := start_ok or (previous.size() == 0)
    else
        result := false
    end
end
```

The user chooses the activity he wants to start, then its *progress* is set to 0.

```
operation start() : Void is do
    progress := 0
end
```

– *Make the progression rate of a started activity evolve:* One selects the activities whose progression rate can evolve. Then, the user chooses the one whose progression he wants to increase and gives the progression percentage that will be added to the current rate (operation *setProgression*).

– *Finish an activity:* One selects all the activities that can be stopped, i.e., those whose *finishable* operation return *True*. *finishable* evaluate whether an activity can be stopped or not according to the precedences rules to which it is subjected (relation *Precedes*).

```
operation finishable() : Boolean is do
    var finish_ok : kermeta::standard::Boolean
    var previousActivities : seq Activity [0..*]
    var prevPrecedes : seq Precedes [0..*]
    // Activities must be started
    if progress < 100 and progress >= 0 then
        // Testing previous activities
        prevPrecedes :=
            previous.select{p | p.kind == PrecedenceKind.pk_finish_finish }
        previousActivities := prevPrecedes.collect{p | p.before}
        finish_ok := previousActivities.forAll{a | a.progress==100}
        result := (finish_ok or previous.size()==0)
    else
        result := false
    end
end
```

Then the user selects the one he wants to be finished.

```
operation complete() : Void is do
    progress := 100
end
```

This loop and the choices proposals are implemented in the body of the *run()* operation of the *Process* metaclass. This execution model describes the behavior of all the models which conform to our Kermeta metamodel (SimplePDL); it represents the operational semantics of our Process Description Language.

4 Related Work

The definition of a rigorous semantics for modeling languages is currently a crucial issue in the ”Model-Driven” world. We can note two works that deal with this particularly important problem.

The ISIS laboratory from the Vanderbilt University has been involved in model engineering for many years. They promote the principles of MIC (Model-Integrated Computing), which places models as center piece for the integrated software development. They are developing the GME tool [4], which allows to describe DSL for multi-aspect and hierarchical models. In this scope they face the same problem concerning the definition of precise semantics. They recently proposed to ”anchor” the semantics of a particular DSL into a well-defined and formal semantics model [14]: the ASM (Abstract State Machine) [15] using their transformation modeling language GReAT (Graph Rewriting And Transformation language) [16].

Xactium[6] is a company founded in 2003 whose objective is to provide practical solutions for the development of large software system based on model-driven principles. They developed the XMF-Mosaic tool [6], which allows to define DSL, to simulate and validate models thanks to an extension of the OCL language called xOCL (eXecutable OCL). It provides means to transform models and to define mapping between them and other features for handling models.

These works are very close to the objectives of the TOPCASED environment, i.e., proposing an adaptive modeling environment based on a generative approach (as GME, XMF), offering means of simulation, validation of models by the definition of rigorous semantics.

[6] `http://www.xactium.com`

5 Conclusion and Future Work

This paper advocates the need for more semantic consideration in MDE. We then present several approaches for the integration of these points which are derived from previous work from the programming languages community. We focus on the definition of executable models for a very small subset of the SPEM development process modeling language. This work was based on the use of the Kermeta tool which weaves the model semantics with the metamodel. Further work will detail the other approaches in order to gather engineering knowledge around the semantic MDE. For instance, we are studying the possibility to define denotational semantics. In the programming languages scope, this semantics describes instructions as mathematical objects (i.e., function, integer, tuples, truth value etc.). The main idea of denotational semantics is to associate each phrase of the language with the appropriate mathematical object and thus, to map syntactic domain to a well-defined semantic domain. Mathematical objects are called the *denotation* of syntactic phrases, which are themselves said to *denote* objects. We can say that this denotation is a kind of translation to the mathematics world.

We are foreseeing a similar approach to provide a rigorous definition of DSL semantics. The idea is to target a well-known and well-defined formal language instead of mathematical objects. The challenge is to define transformation from DSL to another language owned by a different technological space and that has a rigorous semantics. This is often called translational semantics [6]. Those technological bridges allow to profit from simulation, checking and execution tools provided by the targeted technological spaces. We are considering to us ATL to define transformations from our DSL to semantics models such as Petri nets, timed automata or transition systems.

We are also expecting to use model transformations to describe rewriting rules over models. Thus, we will be able to express operational semantics in a closer way to former Structural Operational Semantics defined for programming languages by Plotkin [17]. The main profit of this method is that semantics of a language is expressed in its own terms, i.e., there is no need of additional concepts except those related to the transformation language.

This work puts forward the fact that many different metamodels need to be defined in order to manage the various aspects of a system. All these models are differents but related. These relations must be managed in order to reduce the amount of work required for the definition of their semantics.

References

1. Farail, P., Gaufillet, P., Canals, A., Camus, C.L., Sciamma, D., Michel, P., Crégut, X., Pantel, M.: the TOPCASED project: a toolkit in open source for critical aeronautic systems design. In: Embedded Real Time Software (ERTS), Toulouse (2006)
2. Object Management Group, Inc.: Meta Object Facility (MOF) 2.0 Core Specification. (2003)
3. Budinsky, F., Steinberg, D., Ellersick, R.: Eclipse Modeling Framework : A Developer's Guide. Addison-Wesley Professional (2003)
4. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., IV, C.T., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The generic modeling environment. In: Workshop on Intelligent Signal Processing, Budapest, Hungary (2001)

5. ATLAS: KM3 : Kernel metametamodel. Technical report, LINA & INRIA, Nantes (2005)
6. Clark, T., Evans, A., Sammut, P., Willans, J.: Applied metamodelling - a foundation for language driven development. version 0.1 (2004)
7. Ehrig, K., Ermel, C., Hänsgen, S., Taentzer, G.: Towards graph transformation based generation of visual editors using eclipse. Electr. Notes Theor. Comput. Sci **127** (2005)
8. Object Management Group, Inc.: UML Object Constraint Language (OCL) 2.0 Specification. (2003) Final Adopted Specification.
9. Richters, M., Gogolla, M.: Validating UML models and OCL constraints. In Evans, A., Kent, S., Selic, B., eds.: UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference. Volume 1939 of LNCS., Springer Verlag (2000) 265–277
10. Jouault, F., Kurtev, I.: Transforming models with atl. In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica (2005)
11. Bézivin, J., Jouault, F.: Using atl for checking models. In: GraMoT. (2005)
12. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented metalanguages. In: LNCS, Montego Bay, Jamaica, MODELS/UML'2005, Springer (2005)
13. Breton, E.: Contribution à la représentation de processus par des techniques de métamodélisation. PhD thesis, Université de Nantes (2002)
14. Chen, K., Sztipanovits, J., Abdelwalhed, S., Jackson, E.: Semantic anchoring with model transformations. In LNCS 3748, S.V., ed.: Model Driven Architecture - Foundations and Applications, First European Conference (ECMDA-FA). (2005) 115–129
15. Gurevich, Y.: The abstract state machine paradigm: What is in and what is out. In: Ershov Memorial Conference. (2001)
16. Agrawal, A., Karsai, G., Kalmar, Z., Neema, S., Shi, F., Vizhanyo, A.: The design of a language for model transformations. Technical report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA. (2005)
17. Plotkin, G.: A structural approach to operational semantics. Technical Report DAIMI FN-19, Department of Computer Science, Aarhus University, Denmark (1981)