Towards a Formal Verification of Process Model's Properties - SimplePDL and TOCL Case Study

Benoît Combemale, Pierre-Loïc Garoche, Xavier Crégut, Xavier Thirioux, François Vernadat

Towards a Formal Verification of Process Model's Properties
SimplePDL and TOCL Case Study

Benoît Combemale, Pierre-Loïc Garoche, Xavier Crégut, Xavier Thirioux
Institut de Recherche en Informatique de Toulouse (CNRS UMR 5505), Toulouse, France
{ *first_name . last_name @ enseeiht.fr* }

Francois Vernadat
Laboratoire d'Analyse et d'Architecture des Systemes (CNRS), Toulouse, France
{ *last_name @ laas.fr* }

Keywords: Metamodelling, Properties Validation, Verification, Temporal OCL, Process Model, Petri Nets, LTL, Models Semantics, Model Transformation

Abstract: More and more, models, through Domain Specific Languages (DSL), tend to be the solution to define complex systems. Expressing properties specific to these metamodels and checking them appear as an urgent need. Until now, the only complete industrial solutions that are available consider structural properties such as the ones that could be expressed in OCL. There are although some attempts on behavioural properties for DSL. This paper addresses a method to specify and then check temporal properties over models. The case study is SIMPLEPDL, a process metamodel. We propose a way to use a temporal extension of OCL, TOCL, to express properties. We specify a models transformation to Petri Nets and LTL formulae for both the process model and its associated temporal properties. We check these properties using a model checker and enrich the model with the analysis results. This work is a first step towards a generic framework to specify and effectively check temporal properties over arbitrary models.

1 Introduction

Domain specific approaches tend to be the next approach for specifying complex systems, giving the appropriate abstraction. They can be easily built by domain experts and can then be integrated in generic toolkits and frameworks. Nowadays, there exists a bunch of environments allowing to define DSL (EMF[1], GME[2] ...) mainly focusing on abstract and concrete syntaxes.

Once a metamodel specific to a particular domain has been defined, one wants to express properties that have to be verified for models of this DSL. Such extensions are usually expressed in OCL and describe structural properties of the model. Initially OCL constraints were applied to UML models. Therefore many works and tools have been designed in order to verify these constraints. Tools developed for UML have been adapted to DSL.

However, for behavioural properties, there is a lack of effective works that define all the steps from the property specification to its effective verification. Numerous current projects, such as Topcased[3], consider these problematics as a main topic.

The paper introduces a property-driven approach for specifying and checking temporal properties. The case study is process engineering. Our approach can be described as simple steps. We first characterise the properties. Then process states must be identified. The DSL metamodel is then extended to represent these states. We adapt an OCL temporal extension, formalized using a LTL semantics (Chaki et al., 2004), to represent our temporal properties. The properties are then checked in another formalism: Petri nets. A model transformation to Petri nets is given and allows to apply model checking on an observational abstraction of the trace semantics of the given model with respect to the properties. Finally the result of the analysis is used to enrich the model with properties information.

[1] `http://www.eclipse.org/emf/`

[2] `http://www.isis.vanderbilt.edu/projects/gme/`

[3] *Toolkit In OPen source for Critical Applications and SystEms Development*, `http://www.topcased.org`

This paper gives the following contributions:

- we propose a property-driven approach to identify dynamic states of process models;
- we introduce a temporal extension of OCL based on process states;
- we translate temporal constraints into LTL constraints on the Petri nets;
- we propose an observational trace semantics for SIMPLEPDL;
- we define SIMPLEPDL denotational semantics though a mapping to Petri nets;
- we define a front end for the Tina model checker.

This paper is organised as follows: the second section introduces our DSL, a process metamodel, as well as the natural expression of the user needs for models validation. The third section develops our proposition on our case study. The fourth section considers related works then the last section concludes.

2 Case Study: Process Model Validation

Our contribution is introduced through a modelling language example on which we would like to express a set of properties that have to be verified on all possible models. Our DSL is a simple process description language: SIMPLEPDL.

We first introduce the domain concepts of SIMPLEPDL and then the kind of properties we want to check on models. The properties we are interested in are properties specific to our DSL that must be satisfied for every model of our metamodel. In fact, our approach of verification is driven by those properties. Properties allows to caracterise SIMPLEPDL models states and then refine the metamodel to capture them.

2.1 SIMPLEPDL

SIMPLEPDL is an experimental language for specifying processes. The SPEM standard (*Software Process Engineering Metamodel*) (omg, 2005) proposed by the OMG inspires our work[4], but we also take ideas from the UMA metamodel (*Unified Method Architecture*) used in the EPF Eclipse plug-in[5] (*Eclipse Process Framework*), dedicated to process modelling. It is simplified to keep the presentation simple.

[4]We propose an analysis of the SPEM 1.1 standard in (Combemale et al., 2006a)

[5]http://www.eclipse.org/epf/

The SIMPLEPDL metamodel is given in Figure 1. It defines the process concept (*Process*) composed of a set of work definitions (*WorkDefinition*) representing the activities to be performed during the development. One workdefinition may depend upon another (*WorkSequence*). In such a case, an ordering constraint (*linkType*) on the second workdefinition is specified, using the enumeration *WorkSequenceType*. For example, two workdefinitions WD1 and WD2 linked by a precedence relation of kind *finishToStart* specify that WD2 will be able to start only when WD1 is finished (and respectively for *startToStart*, *startToFinish* and *finishToFinish*). SIMPLEPDL does also allow to explicitly represent resources (*Resource*) that are needed in order to perform one workdefinition (designer, computer, server, ...) and also time constraints (*min_time* and *max_time* on *WorkDefinition* and *Process*) to specify the minimum (resp. maximum) time allowed to perform the workdefinition or the whole process.

One can remark that, for the sake of brevity, some concepts are not presented here such as products (*WorkProduct*) that workdefinitions handle, or roles (*Role*) that can be assimilated to resources.

2.2 Properties

We now present the different kinds of properties specific to the proposed metamodel: structural ones, temporal ones and quantitative ones. We will particularly develop the second kind as a core concept for the rest of the paper.

Structural properties: The expressivity of metamodelling languages (i.e. meta-metamodels) does not allow to formally capture the whole set of the language properties, i.e. the axiomatic semantics. They mainly capture the cardinalities constraints

In programming languages, the axiomatic semantics is usually based on mathematical logics and expresses a proof method for some construction properties of a language (Cousot, 1990). It can be very general, such as Hoare triples or restricted to ensure construction consistency (e.g. typing).

In a modelling language, this second kind of use is expressed using well formedness rules at the metamodel level. Such rules have to be realised by all models that are conform to this metamodel. One can check these rules by static analysis on models.

In order to express the rules, the OMG advocates the use of OCL (omg, 2003; Warmer and Kleppe, 2003). Applied at the metamodel level, OCL can add properties, mostly structural, that could not have been captured by the metamodel definition. It is a mean to

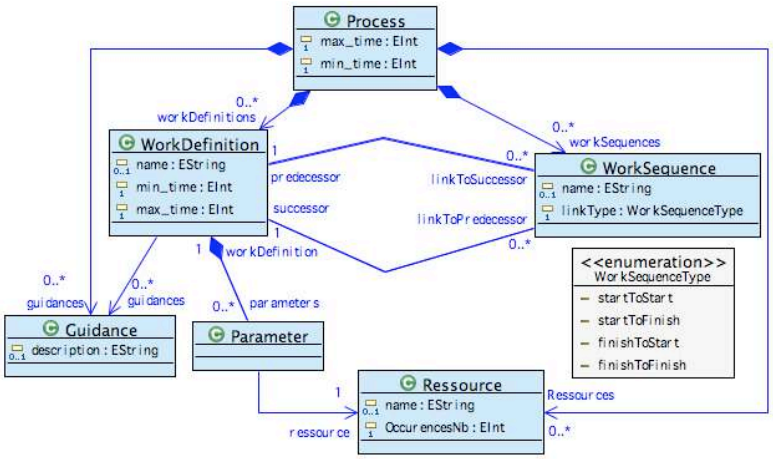

Figure 1: SIMPLEPDL metamodel

precise the metamodel semantics by limiting possible conforming models.

There is an example of such a constraint:

> One Worksequence could not have the same Workdefinition as source and target.

That can be formalised as

```
context WorkSequence inv :
  self.predecessor <> self.successor
```

In order to check that a particular model satisfies these constraints, one can use an OCL checker such as USE (Richters and Gogolla, 2000), OSLO[6], or EMFT[7].

Temporal properties: Many properties have to be satisfied in every model execution. The expert of the domain will formalise them when defining the metamodel. In our process metamodel, any workdefinition can be started and then be finished. One can then ask, and therefore check, whether a given process model effectively terminates, i.e. that every workdefinition in it finishes. Taking into account time and resources, some new properties appear that are independent of any model. For a given set of resources, described in the model, does the process terminate ? Is it possible to satisfy every real time constraints expressed on the workdefinition (attributes *min_time* and *max_time*) ? We could also express temporal properties depending on the capability for a workdefinition to suspend its work and free its resources to share them temporarily with other workdefinitions.

[6]Open Source Library for OCL, `http://oslo-project.berlios.de`.

[7]The Eclipse Modeling Framework Technology project, `http://www.eclipse.org/emft/`

Quantitative properties: The aim of such properties is to describe or compute critical paths of executions in terms of minimal or maximal resource consumption. WCET[8] or schedulability are typical examples of quantitative properties. For instance, time and memory are standard resources the usage of which one would wish to measure. In this respect, a consumption (and production) model has first to be set. In simple situations, a discrete and finite model may fit the needs, as it is the case when we focus on a single kind of resource, with a fixed and finite number of instances. Memory requirements alone usually fall in this simple class and could be checked with off-the-shelf model-checking techniques for discrete models, nevertheless with possible minor adaptations. Yet, for more involved models and resources, in order to precisely represent what is happening, we may find it mandatory to write down quite general arithmetical constraints or to handle continuous quantities (as in real-time systems specifications for instance). As discussions about relevance of such models and their verification issues are quite complex and out of the scope of this present work, we choose for the time being to simply rule out quantitative properties and postpone their introduction for future works. Thus, we stick to the presentation of the overall methodology without delving upon details.

2.3 Dynamic Informations & Property-Driven Approach

Expressing temporal properties that have to be checked on each model execution implies the existence of an operational semantics that is not expressed

[8]Worst Case Execution Time

within a metamodelling language such as the MOF.

In our case, the execution of a process model consists in performing the different work definitions of the process. When executing a model, every work definition must be started and the overall process must finally reach a state where all of them are in the state finished.

The real semantics can be arbitrary complex, and sometimes non computer-representable in case of complex continuous systems.

Our previous works have investigated the use of operational semantics (Combemale et al., 2006b) and translational semantics (Combemale et al., 2007). In this paper we present a generic approach to define the abstract dynamic semantics, a semantics of observable events, built upon the properties expressed at the metamodel level.

The temporal properties expressed for every model conform to the metamodel are built over a notion of states. The formal semantics associated to the system can be seen as the set of maximal finite traces which elements are model states. If the metamodel has a well defined operational semantics, it can be easily expressed as a modification of instances' attributes or a modification of the topology (dynamically creating or killing instances). On the contrary, if the associated semantics is not formally defined, the states characterised by properties allow to define an observable operational semantics. Following this idea, if state properties rely on notions that cannot be directly expressed in the model (classical OCL queries), then the metamodel must be enriched to express these notions. The dynamic operational semantics, i.e. the Kripke structure that allows to build trace semantics, must then be approximated by defining transition between characterised states. It is the work of the domain expert to describe them.

This approach has mainly three advantages:

- it gives a method to define a formal semantics for metamodel that could not always initially describe it;
- this approach is incremental: the domain expert can specify a property, that characterises new states. Then he will extend the metamodel to represent this new dynamic information. The expert can then introduce another property and extend again the metamodel.
- it allows to easily define an "observable" approximation of the trace semantics. One such approach allows to check the properties defined, because the semantics were defined depending on the needs expressed by these properties. It can also help in defining a minimal abstract semantics that gives access to formal tools allowing to check properties on a reasonably-sized state space.

3 An Approach to Validation through Petri Nets and LTL

In this section, we will follow all the steps that allow us to express temporal constraints on our SIMPLEPDL metamodel.

3.1 Characterising Properties

This first step must be realised by the expert. As expert of processes, we say that every SIMPLEPDL model must verify the following properties. We separate them in two classes: universal properties that have to be satisfied by every execution and existential properties that must be true in at least one execution.

Our universal properties are:

- every workdefinition must start,
- all started workdefinitions must finish,
- once a workdefinition is finished, it has to stay in this state,
- a workdefinition is able to start depending on worksequences constraints. All workdefinitions that are linked to it using a startToStart worksequence are started. Reciprocally all workdefinitions that are linked to it using a finishedToStart worksequence are finished.

  The same kind of properties apply for finishing each workdefinition.

Our existential properties are:

- every workdefinition must take more than *min_time* and less than *max_time* to be performed,
- the overall process is able to finish, i.e. when all workdefinitions are finished in a good state (i.e. between *min_time* and *max_time*).

3.2 Characterising States

The second step consists in characterising different states for the metamodel elements from the properties. From the aforementioned temporal properties, we can identify two orthogonal ideas for the workdefinition element. First, a workdefinition can be not started, started and finally finished. Secondly, there is a notion of time and clock associated to each workdefinition; but this time is only relevant for transition enabling conditions (in our case transitions that start and finish a workdefinition) and is not explicit in state

properties. Thus it can be represented into the finite set of states $\{$ *tooEarly*, *ok* , *tooLate* $\}$. This second orthogonal idea is only relevant when the progress is finished. Therefore we add a fourth state: *notFinished*.

3.3 Extending the Metamodel to Represent Dynamic Information

We now have to express these states by extending the WorkDefinition elements in order to introduce attributes that reflect dynamic information, i.e. the state of the current workdefinition. We choose to add three variables: $state \in \{notStarted,\ started,\ finished\}$, $time\_state \in \{notFinished,\ tooEarly,\ ok,\ tooLate\}$ and $clock \in \mathbb{R}^+$.

An observational abstraction of the operational semantics of our processes with respect to our properties can now be defined.

The expert has again to formalise the initial states and the transition relation. In our case, it is quite natural: the initial states are the singleton $\{w \mapsto (notStarted, notFinished) |\ w \in \mathcal{WD}\}$. We define the transition relation for one workdefinition in $\mathcal{WD}$ in Figure 2.

3.4 Expressing Temporal Properties : Temporal OCL

A few temporal extensions of OCL have already been proposed in a UML context (see related works section). We have chosen the proposal of (Ziemann and Gogolla, 2002) for two main reasons:

1. The semantics of the temporal expressions is formally defined on a trace semantics. Such traces are finite sequences of system states, describing a snapshot of the running system. Even if this work was initially defined on UML models, the trace semantics can be easily generalised to arbitrary state sequences while keeping the original semantics of temporal operators.
2. The syntax of this OCL extension is quite natural. It introduces usual future-oriented temporal operators such as *next*, *existsNext*, *always*, *sometimes* as well as their past-oriented duals. We will only use the future-oriented ones because we intend to effectively check properties using the *Tina* model checker (Berthomieu et al., 2004), which does not support past-oriented operators.

Let us go back to our process example to introduce our generalisation. A snapshot of our process has to describe precisely the state of each workdefinition. We take as given a finite set $\mathcal{S}$ of such states. Let $\mathcal{WD}$ be the set of workdefinitions of the model. Let Σ be the set of the process state: $\Sigma = \mathcal{WD} \mapsto \mathcal{S}$.

A trace $\hat{\sigma}$ of the process is a maximal finite sequence of process states $\langle \sigma_0, \ldots, \sigma_n \rangle, \sigma_i \in \Sigma$, where σ_0 denotes the initial process state. Semantically, we have two kinds of transitions. First, continuous time-passing transitions that are here unobservable and consist in incrementing all workdefinition clocks by a quantity *dt* simultaneously. Second, event-based transitions that change the states of workdefinitions as defined by the expert above. Two consecutive events in a sequence are related through a combination of the time-passing transition followed by an event-based transition.

In order to ease the definition of our properties we introduce the new operator *precedes*. Such an operator can be described using the previous ones:

$$e_1 \ precedes \ e_2 = always!(e_2) \ until \ e_1$$

Expressions of our TOCL extension are now OCL expressions over the model elements using these temporal operators. We also allow these expressions to be built over state names defined in the aforementioned set $\mathcal{S}$. The universal temporal properties can now be expressed as:

$always$ (`notStarted` $\implies$ $sometime$ `started`)
$always$ (`started` $\implies$ $sometime$ `finished`)
`finished` $\implies$ $always$ `finished`
$always$ (($preds_s.state =$ `started` &&
$pred_f.state =$ `finished` &&
`notStarted`) $\implies$ $sometime$ `started`)

The existential ones have to be rewritten in order to be checked: we will verify the negation of each formula. If the analysis gives a correct answer, there is no trace satisfying the property. On the contrary, if the analysis gives a negative answer with a counter-example, the existential property is verified and the counter-example is one of the traces satisfying the temporal property. We only give here the first existential property.

$$\begin{aligned} &always\ (not\ wd.time\_state = ok) \\ \equiv\ &always\ (wd.time\_state = tooEarly \\ &\quad || wd.time\_state = tooLate) \end{aligned}$$

We have given the textual concrete syntax and the associated semantics of our extension of TOCL. In order to integrate it into a metamodelling approach (i.e. defining properties at the metamodel level), it is necessary to define, at the MOF level, the OCL abstract syntax and its temporal extension. To give the ability for any DSL to use TOCL, we start from the OCL metamodel defined in (Richters and Gogolla, 1999) and promote it at the MOF level (omg, 2006) (fig.

Let w be the considered Work Definition.

$$\begin{array}{lll}
\forall ws = w.predecessor, (ws.linkType = startToStart \&\& ws.linkToPredecessor.state = started) & & \\
\quad ||(ws.linkType = finishedToStart \&\& ws.linkToPredecessor.state = finished) & & \\
notStarted, notFinished, clock & \rightarrow & started, notFinished, 0 \\
\forall ws = w.predecessor, (ws.linkType = startToFinished \&\& ws.linkToPredecessor.state = started) & & \\
\quad ||(ws.linkType = finishedToFinished \&\& ws.linkToPredecessor.state = finished) & & \\
started, notFinished, clock < min_time & \rightarrow & finished, tooEarly, clock \\
started, notFinished, clock \in [min_time, max_time] & \rightarrow & finished, ok, clock \\
started, notFinished, clock > max_time & \rightarrow & finished, tooLate, clock
\end{array}$$

Figure 2: Event-based Transition Relation for WorkDefinitions

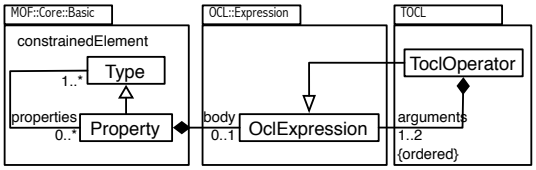

Figure 3: Temporal OCL integration to MOF

3). We also add the set of temporal operators defined in (Ziemann and Gogolla, 2002) and in the aforementioned extension (fig. 3).

We have now introduced the concrete and abstract syntax and semantics of our temporal OCL extension. With these temporal constraints we are now able to express complex properties on the behaviour of the model to be checked. One immediate application of these constraints is the transformation of every invariant as defined in OCL as the first kind of properties. We now consider executing models and each invariant has to be checked in every process state of all possible traces. Therefore, we rename invariant expressions *e* to *always e*.

The next part introduces how these model states can be built using OCL over model attributes.

3.5 Denotational Semantics to Petri Net and LTL

In this study, we choose to use the technical space of Petri nets as the target representation for formally expressing our process models. We also choose to express our temporal formulae as LTL formaluae (*Linear Temporal Logic*) over the Petri net associated to a process model. Then we manipulate Petri nets and LTL formulae within the *Tina*[9] toolkit.

TINA (*TIme Petri Net Analyser*) is a software environment to edit and analyse Petri nets and timed nets (Berthomieu et al., 2004). The different tools constituting the environment can be used alone or together. Some of these tools will be used in this study:

- *nd* (*NetDraw*) : *nd* is an editing tool for automatas and timed networks, under a textual or graphical form. It integrates a "step by step" simulator (graphical or textual) for the timed networks and allows to call other tools without leaving the editor.
- *Tina* : this tool builds the state space of a Petri net, timed or not. *Tina* can perform classical constructs (marking graphs, covering trees) and also allows abstract state space construction, based on partial order techniques. *Tina* proposes, for timed networks, all quotient graph constructions discussed in (Berthomieu and Vernadat, 2006).
- *selt*: usually, it is necessary to check more specific properties than the ones dedicated to general accessibility alone, such as boundedness, deadlocks, pseudo liveness and liveness already checked by *tina*. The *selt* tool is a *model-checker* for formulae of an extension of temporal logic *seltl* (State/Event *LTL*) of (Chaki et al., 2004). In case of non satisfiability, *selt* is able to build a readable counter-example sequence or in a more compressed form usable by the *TINA* simulator, in order to execute it step by step.

3.5.1 Denotational Semantics of SIMPLEPDL

PetriNet In this case study, we use timed Petri nets as a paradigm to express the semantics of our processes, models of SIMPLEPDL. The semantics is now a denotational one defined as a mapping from SIMPLEPDL to Petri nets. The Petri nets metamodel is given in Figure 4. A Petri net (*PetriNet*) is composed of nodes (*Node*) that denote places (*Place*) or transitions (*Transition*). Nodes are linked together by arcs (*Arc*). Arcs can be normal ones or read-arcs (*ArcKind*). The attribute *tokensNb* specifies the number of tokens consumed in the source place or produced

[9] http://www.laas.fr/tina/

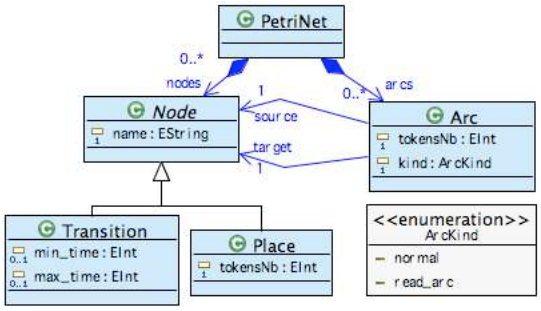

Figure 4: Petri Nets Metamodel

in the target one (in case of a read-arc, it is only used to check whether the source place contains at least the specified number of tokens). Petri nets markings are defined by the *tokensNb* attributes of places. Finally, a time interval can be expressed on transitions.

Mapping The translation schema that transforms a process model into a Petri nets model (SIMPLEPDL2PETRINET) is given in Figure 5. Each workdefinition is translated into four places characterising its state (*NotStarted*, *Started*, *InProgress* or *Finished*). A *WorkSequence* becomes a *read-arc* from one place of the source workdefinition to a transition of the target workdefinition. The state *Started* records that the workdefinition has been started.

We also add five places that will define a local clock. The clock will be in state *TooEarly* when the workdefinition ends before *min_time* and in the state *TooLate* when the workdefinition ends after *max_time*.

Our transformation has been written in ATL, ATLAS Transformation Language (Jouault and Kurtev, 2005). A first rule expresses one workdefinition in terms of places and transitions. A second one translates a work sequence into a read-arc between the adequate place of the source workdefinition and the appropriate transition of the target workdefinition. Finally a third rule considers the whole process and builds the associated Petri net.

In order to manipulate the obtained Petri net inside a dedicated tool such as *Tina*, we have composed the preceding transformation with a transformation PETRINET2TINA that translates a *PetriNet* model into the textual syntax of the *Tina* tool.

Traceability The set of translation choices (i.e. the mapping) defined in the SIMPLEPDL2PETRINET transformation is captured in the ATL source code. The benefit of this language is that it is itself defined as a metamodel. It allows to obtain a model (conform to the ATL metamodel) corresponding to the transformation. This transformation model can be reused as an entry model for another transformation (*Higher Order Transformation*). We can remark that it is possible to enrich traceability information as proposed by (Jouault, 2005).

3.5.2 Denotational Semantics of TOCL

The transformation model defined during the translation to Petri nets is used to instantiate a generic transformation that defines LTL properties from the initial metamodel properties, instantiated relatively to the initial process model.

Our experiments show a lack in current MDE technology that does not allow to parameterise a model transformation. The use of a programming language such as Java, as well as a specific library such as EMF, is necessary to handle such a transformation.

3.6 Models Validation and Feedback

Model checking results have to be interpreted at the SIMPLEPDL model level in order to provide a complete front-end to the end-user. Properties verified in the Petri net correspond to a double instantiation of the properties expressed at the metamodel level. The interpretation of the results must be the conjunction of the results obtained for the different instantiations of a metamodel property.

The feedback of properties results (catching in a first time the truth value of the property) in the model, can be automatically computed using the transformation model defined during the translation SIMPLEPDL2PETRINET. This translation captures the set of choices that have been done during the transformation (i.e. the mapping table). This technique uses a Higher Order Transformation that takes a transformation model and allows to trace back the model checker interpretation into the DSL model.

In a first time, we only handle the boolean value returned by the *Tina* analyser. When the LTL properties associated to one SIMPLEPDL properties are satisfied, the property is satisfied. In the other case, the transformation model allows to identify in the model the faulty element and to update its dynamic information in order to visualise the state in which the property failed. We have to take care of the kind (universal or existential) of temporal properties expressed. In case of an existential one, the negation of the result has to be returned.

The next step consists in handling counter-examples. Such counter-examples generated by the model checker could be expressed on the model and be then injected in the model animator (in our case,

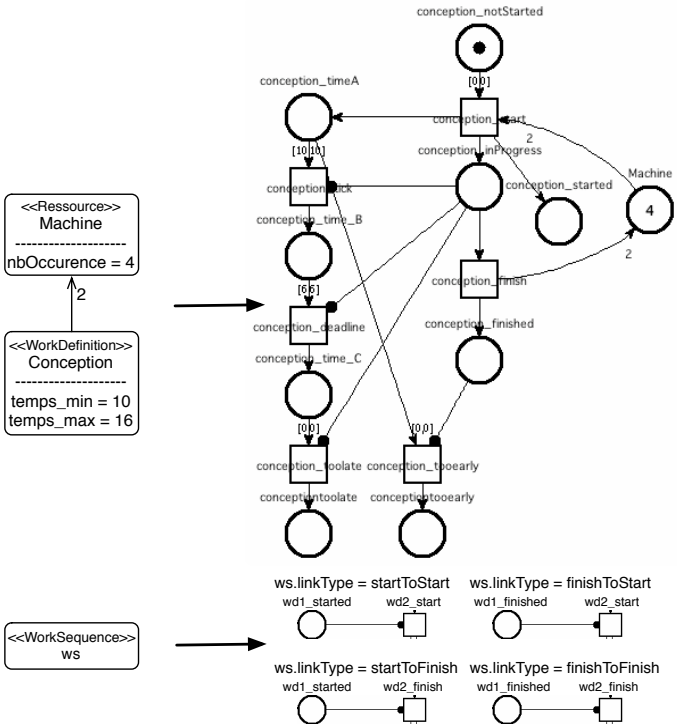

Figure 5: Translation schema from SIMPLEPDL to PetriNet

the one defined on SIMPLEPDL) defined in the Topcased project.

4 Related Works

4.1 Models Semantics

The formal semantics definition of modelling languages is an active research field in the MDE community. Beside our previous works presented in (Combemale et al., 2006b) and (Combemale et al., 2007), we have identified other projects that consider this important subject.

The ISIS laboratory of the Vanderbilt University focuses on MDE for many years. They proposed the MIC approach (Model-Integrated Computing), in which models are at the heart of the integrated software development. Recently, they propose, in (Chen et al., 2005), a semantics anchored to a model of formal semantics built upon ASM (Abstract State Machine) (Gurevich, 2001), using the transformation language GReAT (Graph Rewriting And Transformation language) (Agrawal et al., 2005).

Xactium[10] is a company created in 2003. Its objectives are to provide practical solutions for the development of systems based on MDE principles. It developed the XMF-Mosaic tool (Clark et al., 2004) that allows to define a DSL, to simulate and to validate models using a extension of the OCL language named xOCL (eXecutable OCL). XMF-Mosaic also provides means to transform models and to define translations to other technical spaces.

These works are very near to the objectives of the TOPCASED environment, i.e. to propose a modular modelling environment based on a modular generative approach (like GME, XMF), as well as a formal semantics that provides simulation and model validation tools. Our works based on Kermeta follow an approach similar to the ones of xOCL inside the XMF-Mosaic tool.

The semantics anchoring proposed by the ISIS laboratory is similar to the denotational semantics such as the mapping to Petri nets we propose. The main difference is that we want to give more flexibility in the choice of the semantics model and to allow easier feedbacks from simulations or verifications inside a particular model. However, they do not propose

[10] http://www.xactium.com

the use of models rewriting rules to define the operational semantics.

4.2 Models verification

Verification of UML models In order to specify structural properties on UML models, OCL was introduced. It is therefore accepted as the standard language to express structural properties on UML models. There also exists a bunch of tools to check OCL properties for any model.

As for temporal properties, some recent works intend to extend the usual OCL syntax and semantics to give the capability to express temporal constraints. All these works address OCL extensions in an UML context. They do not address how the transition system is derived from the model.

The aforementioned work of (Ziemann and Gogolla, 2002) proposed to extend OCL with usual temporal operators and defined their semantics on the trace semantics of the UML model. This work is a first step towards the simulation of temporal properties over traces using the USE tool.

Some works, such as (Flake, 2003) and (Flake and Mueller, 2003), are focused on the expression of real time constraints while keeping the original OCL syntax. They relied on StateChart states to express the dynamic constraints of the system. Then, they mapped their constraints into Clocked-CTL.

(Cengarle and Knapp, 2002) proposed to express real time constraints using two new classes Time and Events. A new OCL template is introduced and the usual ones (pre-post, inv and action) are translated in it. The semantics is also defined as a trace semantics.

In (Distefano et al., 2000), the authors expressed non temporal OCL constraints into their object-oriented version of CTL. They defined formally what is a state of the UML model. They are able to check whether a property expressed in OCL can be checked in every reachable state.

The work of (Bradfield et al., 2002) introduced new OCL templates. They mapped them into $O\mu(OCL)$-calculus, an observational μ-calculus, which expressions are OCL expressions. The semantics of their $O\mu(OCL)$-calculus is defined over the states of (Distefano et al., 2000). Using model checking tools, the author intends to check the property on a CCS term modelling their UML system.

All the previous works only specify the way OCL must be extended to deal with temporal formulae in order to verify or simulate them later but do not reach this last step, at least not in an automatic manner. For instance, the point of generating the transition system from the initial UML is not solved.

Verification of DSL models OCL was initially defined on UML but was quickly defined for every metamodel. It is the main tool to express structural properties in DSL. Existing OCL checkers are also model independent.

5 Conclusion & Future Works

The context of this article was to integrate formal methods for refining DSL semantics. DSL semantics is usually restricted to structural properties and dynamic aspects are often only informally described or are even implicit. As our aim is to express and validate behavioural and operational properties within a metamodelling framework, the first step was to introduce and handle an operational semantics, instantiated in this article to our process metamodel SIMPLEPDL. This semantics is introduced with respect to properties of interest, given by an expert of the domain. First, a notion of state is introduced, followed by the definition of transitions and executions. Temporal operators, forming temporal properties, are also introduced. In order to check these properties, first a denotational semantics is provided as a mapping from SIMPLEPDL processes to Petri nets, second a front end to the Tina model-checker is defined.

Few things still remain to be done. In particular, the current presentation is focused on SIMPLEPDL, it still needs to be abstracted away to get a more general approach. The formal connection between the observational operational semantics and the denotational semantics induced by the ATL transformation have to be validated.

Currently, we are implementing a prototype allowing us to define metaproperties through an Ecore modelling language extension given by the Eclipse EMF plugin. The expression of temporal properties uses an extension of OCL metamodel provided by the EMFT plugin on which we add the set of temporal operators described above, in the article. An interface associated to the TOCL textual concrete syntax will be integrated using generators such as Sintaks (Muller et al., 2006) or TCS (Jouault et al., 2006). Our prototype must also integrate the set of ATL transformations and provide a front end to Petri nets using the *T*ina toolkit, through the SIMPLEPDL language. We still have, in case of a negative answer from the model checker, for a given property, to retrieve the generated counter-example. It can then be injected within both the model animator currently developed with the TOPCASED project and the SIMPLEPDL model graphical editor defined with TOPCASED.

REFERENCES

(2003). *UML Object Constraint Language (OCL) 2.0 Specification*. Object Management Group, Inc. Final Adopted Specification.

(2005). *Software Process Engineering Metamodel (SPEM) 1.1 Specification*. Object Management Group, Inc. formal/05-01-06.

(2006). *Meta Object Facility (MOF) 2.0 Core Specification*. Object Management Group, Inc. Final Adopted Specification.

Agrawal, A., Karsai, G., Kalmar, Z., Neema, S., Shi, F., and Vizhanyo, A. (2005). The design of a language for model transformations. Technical report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37235, USA.

Berthomieu, B., Ribet, P.-O., and Vernadat, F. (2004). The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756.

Berthomieu, B. and Vernadat, F. (2006). *Réseaux de Petri temporels : méthodes d'analyse et vérification avec TINA*. Traité IC2.

Bradfield, J. C., Filipe, J. K., and Stevens, P. (2002). Enriching OCL using observational mu-calculus. In *Fundamental Approaches to Software Engineering*, pages 203–217.

Cengarle, M. V. and Knapp, A. (2002). Towards OCL/RT. In *International Symposium of Formal Methods Europe on Formal Methods (FME) - Getting IT Right*, pages 390–409, London, UK. Springer-Verlag.

Chaki, S., E, M., Clarke, Ouaknine, J., Sharygina, N., and Sinha, N. (2004). State/event-based software model checking. In *4th International Conference on Integrated Formal Methods (IFM)*, volume 2999 of *LNCS*, pages 128–147.

Chen, K., Sztipanovits, J., Abdelwalhed, S., and Jackson, E. (2005). Semantic anchoring with model transformations. In *Model Driven Architecture - Foundations and Applications, First European Conference (ECMDA-FA)*, volume 3748 of *LNCS*, pages 115–129.

Clark, T., Evans, A., Sammut, P., and Willans, J. (2004). Applied metamodelling - a foundation for language driven development. version 0.1.

Combemale, B., Crégut, X., Berthomieu, B., and Vernadat, F. (2007). SimplePDL2Tina : Mise en oeuvre d'une Validation de Modèles de Processus. In *3ieme journées sur l'Ingénierie Dirigée par les Modèles (IDM)*, Toulouse, France.

Combemale, B., Crégut, X., Ober, I., and Percebois, C. (2006a). Évaluation du standard SPEM de représentation des processus. *Génie Logiciel, Modèles et Processus de développement*, 77:25–30.

Combemale, B., Rougemaille, S., Crégut, X., Migeon, F., Pantel, M., Maurel, C., and Coulette, B. (2006b). Towards a rigorous metamodeling. In *2nd International Workshop on Model-Driven Enterprise Information Systems (MDEIS)*, Paphos, Cyprus. INSTICC.

Cousot, P. (1990). Methods and logics for proving programs. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 841–994.

Distefano, D., Katoen, J.-P., and Rensink, A. (2000). Towards model checking OCL. In *ECOOP Workshop on Dening a Precise Semantics for UML*.

Flake, S. (2003). Temporal OCL extensions for specification of real-time constraints. In *Workshop Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS) at UML'03*, San Francisco, CA, USA.

Flake, S. and Mueller, W. (2003). Formal semantics of static and temporal state-oriented OCL constraints. *Journal on Software and System Modeling (SoSyM)*, 2(3).

Gurevich, Y. (2001). The abstract state machine paradigm: What is in and what is out. In *Ershov Memorial Conference*.

Jouault, F. (2005). Loosely Coupled Traceability for ATL. In *Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, Nuremberg, Germany.

Jouault, F., Bézivin, J., and Kurtev, I. (2006). TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In *Proceedings of the fifth international conference on Generative programming and Component Engineering (GPCE)*, Portland, Oregon.

Jouault, F. and Kurtev, I. (2005). Transforming models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS*, Montego Bay, Jamaica.

Muller, P.-A., Fleurey, F., Fondement, F., michel Hassenforder, Schneckenburger, R., Gérard, S., and Jézéquel, J.-M. (2006). Model-driven analysis and synthesis of concrete syntax. In *9th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, volume 4199 of *LNCS*, Genova, Italy.

Richters, M. and Gogolla, M. (1999). A metamodel for OCL. In France, R. and Rumpe, B., editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins.*, volume 1723 of *LNCS*, pages 156–171, USA.

Richters, M. and Gogolla, M. (2000). Validating UML models and OCL constraints. In *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference*, volume 1939 of *LNCS*, pages 265–277, York, UK.

Warmer, J. and Kleppe, A. (2003). *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Longman Publishing Co., Inc.

Ziemann, P. and Gogolla, M. (2002). An extension of OCL with temporal logic. In *Critical Systems Development with UML – Proceedings of the UML'02 workshop*, volume TUM-I0208, pages 53–62.