Electronic Communications of the EASST
Volume 42 (2011)


# Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)

## Towards a Methodology for Semantics Specification of Domain-specific Models through Properties

Ragnhild Van Der Straeten

12 pages

Guest Editors: Vasco Amaral, Hans Vangheluwe, Cécile Hardebolle, Lazlo Lengyel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

# Towards a Methodology for Semantics Specification of Domain-specific Models through Properties

**Ragnhild Van Der Straeten***

Vrije Universiteit Brussel, Brussels, Belgium

**Abstract:** Domain-specific languages are designed for a specific domain and their use in the development of a software system enables domain experts to understand and develop models. The full description of a complex software system uses various domain-specific languages, each language having its own meaning. The definition of the precise semantics of domain-specific models is an important problem because semantics are a necessity to develop tools allowing formal analysis and verification. Current approaches focus on specific formalisms or languages which are the most appropriate to reason about certain properties. In this work we propose an approach that allows domain experts to specify the semantics of domain-specific models through properties expressed in a domain-specific language. The advantage of the approach is that domain experts can write down and understand the properties of the models.

**Keywords:** domain-specific language, semantics, domain-specific models, workflows

## 1 Introduction

Domain-specific modelling languages are modelling languages designed for a specific domain. The use of these languages in the development of software systems provides many documented advantages [FR05]. These languages are succinct and enable domain experts to understand and develop models. The full description of a complex system uses various domain-specific languages. Each language has its own meaning. A precise semantics is very important in industrial applications. Moreover, semantics are a necessity to develop tools that allow for formal analysis and verification of, for example, quality attributes [HR04]. Based on the current approaches defining semantics for domain-specific languages, we make the following observations.

A common way to specify the semantics of domain-specific languages is to define a translation from that language to a target language, called a semantic domain. These approaches are known as translational approaches. Translational approaches are mainly used in model-driven engineering (MDE) where model-to-model transformations are used for translating the source language into the semantic domain. Based on the properties that need to be verified or analyzed over the domain-specific models, different semantic domains are proposed in the literature [GLMD09, WP10, LV04, RGLV09, NAD03, CSN05, PS07]. For example, when focus is on synchronisation Petri nets are used as target language, when focus is on state, statecharts are

* Funded by the Belgian State – Belgian Science Policy through the Interuniversity Attraction Poles programme.

used, when focus is on processes, process algebras are used, etc. Consequently, the most appropriate semantic domain needs to be chosen to represent and reason about certain properties and to conduct certain analysis kinds.

The translational approach has disadvantages especially in the context of domain-specific languages. The transformations expressing the translation between the source and the target language are expressed in terms of these languages. The domain expert is expected to know and understand the source language, but we cannot expect the domain expert to understand the target language. In this paper, we use the term semantic domain to denote this target language. Another approach is to describe the operational semantics of a language in terms of its structure and to use an interpreter to execute the descriptions. In the remainder of the paper, this approach is called the *interpretational approach*. Because the domain expert knows these structures, (s)he is able to understand the effects of execution. However, as recognised in [Wac08], translational approaches are needed in addition because they allow for matching a particular target platform or for providing higher efficiency.

Another observation is that there is no systematic approach to define semantics of domain-specific models. All approaches are ad-hoc and not defined within a general framework or using a methodology. In this paper, we propose to adopt a methodology to define the semantics of domain-specific models through the definition of properties. These properties describe certain characteristics or qualities of the models and need to be verified over the given set of models. Using the proposed approach, domain experts are not only able to capture their knowledge into models but also to specify the meaning of their models by defining properties over the models in a domain-specific language. As argued in [PS07] the description of the semantics of DSMLs depends on the concepts in the application domain, the choices of the language designer, the requirements of a particular application area, and the fact that semantics can be used for various purposes in design or analysis. This requires flexibility in describing semantics. Our proposed approach addresses this flexibility requirement because it does not demand to specify the exact meaning of each language or model element.

## 2 Methodology

We propose a general methodology in which the domain expert is able to express domain-specific models conforming to their domain-specific language and to express properties over these models in a domain-specific language. Figure 1 shows the general idea. Domain-specific models (DSms) are specified and conform to domain-specific languages (DSLs). Properties are specified describing certain characteristics or qualities of the models. We call such properties *domain-specific properties (DSps)*. These DSps are defined over DSms and also conform to certain DSLs. In a translational approach, transformations are specified from the domain-specific languages into one or several semantic domains (SDs). The domain-specific models are translated into models conforming to the semantic domain and domain-specific properties are translated into semantic domain properties. These properties can be checked and feedback can be given to the domain expert. This implies the existence of back-annotations from the semantic-domain models and the semantic-domain properties towards the DSms and the DSps. In an interpretational approach only the left part of Figure 1 is considered, i.e., without the transformations into semantic do-

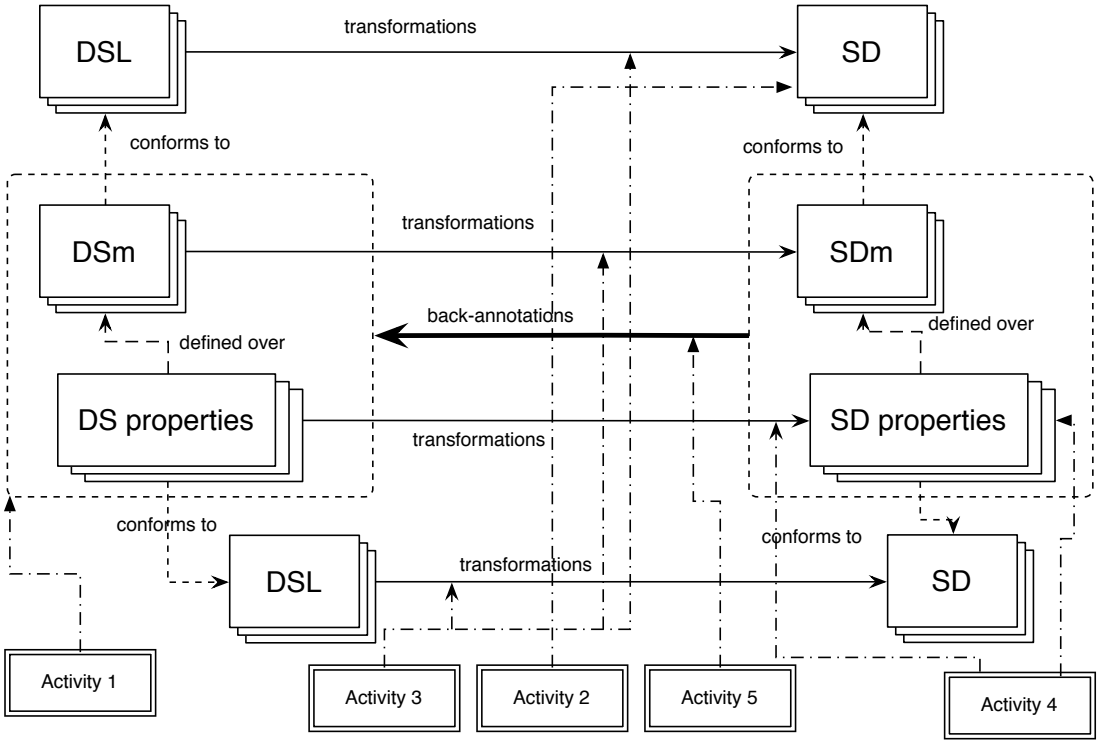


Figure 1: Framework for DSm semantics through properties.

mains.

Our methodology consists of a number of different activities. These activities describe what the important steps are in defining semantics through properties and which actors and products are involved. The actors correspond to the different roles that exist within the context of a software team, e.g., project manager, developer, architect, tester. Certain skills are needed to play a certain role in the context of the domain-specific modeling process. We define the following roles: the application domain expert and the solution domain expert. The application domain expert has knowledge about the application while the solution domain expert has knowledge about modelling languages and semantic domains.

The doubly-outlined boxes at the bottom of Figure 1 indicate the activities of the proposed methodology and the arrows leaving these boxes indicate to which artifacts the activities are related. Before briefly describing these activities in the next section, we make the following remark. It is possible that the semantic domains and the associated properties are translated again into other semantic domains and associated properties. As such a cascade of transformations is executed before the properties can be verified over the models.

The performance of the activities on a certain domain-specific software modelling process defines a concrete process of defining semantics. The starting situation for the application of the

methodology is as follows: a set of models is given defined in certain domain-specific modelling languages. These models and languages can be thought of as parameters of our methodology together with the software development process.

## 2.1 Activity 1: Identification of the Property Configuration

The aim of this activity is to identify *property configurations*. A property configuration consists of two elements. It contains a domain-specific property and a set of models or a set of parts of models, i.e., submodels of which the property needs to be a characteristic. The property configurations are obtained by discovering which models or submodels model certain aspects that affect the given property.

The output of this activity is a repository of identified property configurations. The activity is performed by the application domain expert.

## 2.2 Activity 2: Identification of Semantic Domains

This activity aims at choosing appropriate semantic domains for each identified property configuration. The property contained in the property configuration is considered as a requirement for the semantic domain chosen for defining semantics. Another criterion for choosing the semantic domain is the considered phase of the software development process because this phase will determine the level of abstraction of the models and property. The (sub)models of the property configuration need to be expressible in this semantic domain. Furthermore, tool support should be available for the semantic domains in order to facilitate the checking of the property. Also the considered property needs to be expressible in the chosen semantic domain.

The output of this activity is one or more semantic domains in which the DSms and DSp will be expressed. This activity is performed by the solution domain expert. Remark that this activity is only necessary in a translational approach.

## 2.3 Activity 3: Specification of Model Transformations

Once appropriate semantic domains have been chosen (cf. Activity 2), the DSms of the property configuration need to be mapped into the semantic domain preferably in an automated way. Model transformations will be introduced to define the mappings between the (sub)models and the semantic domains.

The definition of these model transformations is crucial for the correctness of the verification of the formal property. On the one hand, no aspects of the (sub)models should be left out that influence the property. On the other hand, only those aspects of the (sub)models should be mapped into the semantic domains that are important for the considered property otherwise verification or analysis of the property may get too complex.

The output of this activity is a set of model transformations. The activity is performed by the problem domain expert, the solution domain expert and it requires an expert in model transformations as well. This activity is only necessary in translational approaches.

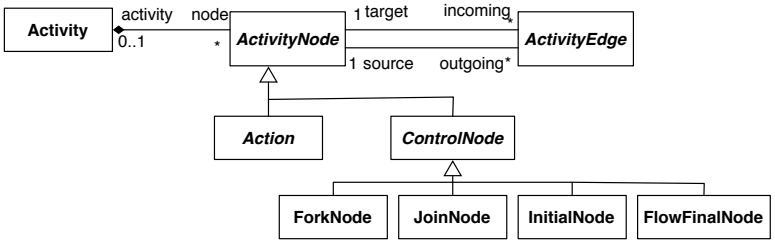


Figure 2: Small part of the UML metamodel for activity diagrams.

### 2.4 Activity 4: Specification of Formal Property

The considered property of a property configuration needs to be formalized in terms of the semantic domain. The resulting formal property is expressed in the language of the semantic domain. If possible this activity is automated through the usage of model transformations.

Using the analysis and verification techniques of the semantic domain the validity of the formal property can be checked w.r.t. the semantic domain models.

The output of this activity is the specification of the formal property of a property configuration. The activity is performed by a solution domain expert.

### 2.5 Activity 5: Specification of Back-Annotations

The results of checking the given property need to be communicated to the domain expert. This requires the translation of the verification result expressed in the semantic domain into a model of the source domain-specific languages. This involves the technique of back-annotation. A known problem related to back-annotation is that we must be able to express the information of the verification result into the source language. For example, it could be the case that the model transformations defined by Activity 3 leave out details from the domain-specific models which are necessary to reconstruct the result into the domain-specific models.

In this activity the information necessary for back-annotation needs to be determined resulting in a repository. The activity is performed by an application and solution domain expert.

In the remainder of this paper, this generic methodology will be instantiated for a specific domain-specific workflow language. First we introduce the general concept of domain-specific workflow languages.

## 3 Case

In this section, we show how the presented methodology can be applied. We start by defining a domain-specific workflow language for the purpose of registering for a conference. Next we introduce properties that express the meaning of the modelled registration scenario and show how the methodology can be applied onto this scenario.

## 3.1 ConferenceRegistration Workflow Language

*Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal.* [Hol95]. Similar to a domain-specific language, a domain-specific workflow language is a small workflow language focused on a particular problem domain. Web-workflow [HVV08] for example, is an object-oriented workflow modeling language for web applications. This language is embedded in WebDSL, a domain-specific language for web application development.

The *ConferenceRegistration Workflow Language* (CWL) we present, is based on the ConferenceRegistrationApps [MV10], i.e., conference registration applications on mobile devices. We defined our language as an extension of the UML 2.0 Activity Diagram language [Obj05]. To be able to understand the ConferenceRegistration Workflow Language we first introduce the most important concepts of UML 2.0 Activity Diagrams. These concepts are shown in Figure 2. *An activity represents a behaviour that is composed of individual elements that are actions. The flow of execution is modelled as activity nodes connected by activity edges. An action is an activity node. Activity nodes also include flow-of-control constructs, such as synchronization, decision, and concurrency control*[Obj05]. An action represents a single step within an activity, that is, one that is not further decomposed within the activity.

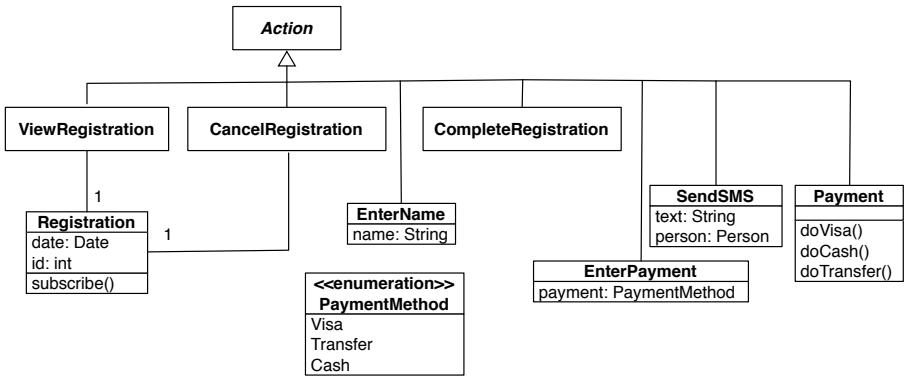


Figure 3: ConferenceRegistration metamodel.

Figure 3 shows the metamodel of our language. The CWL is an embedded language extending the UML 2.0 Activity Diagram language with conference registration abstractions. In particular, the UML metamodel element *Action* is extended by defining seven different conference registration actions: *ViewRegistration*, *CancelRegistration*, *EnterName*, *CompleteRegistration*, *SendSMS*, *EnterPayment* and *Payment*.

## 3.2 Registration Scenario

Figure 4 shows the *Conference Registration* activity consisting of several actions and representing a particular conference registration process. The first two actions that need to be executed

are `EnterFirstName` and `EnterLastName`. These actions are instantiations of the *EnterName* action element of our CWL language. Next, the payment is selected and executed. The counter `i` indicates that if payment fails it is possible to retry three times. If payment fails after three retries, an SMS is sent notifying the user that the payment has failed. If payment succeeds within three retries, the registration is completed and an SMS is sent notifying the user that the registration succeeded. If registration succeeded, the registration process is ended or the user can opt to cancel the registration.

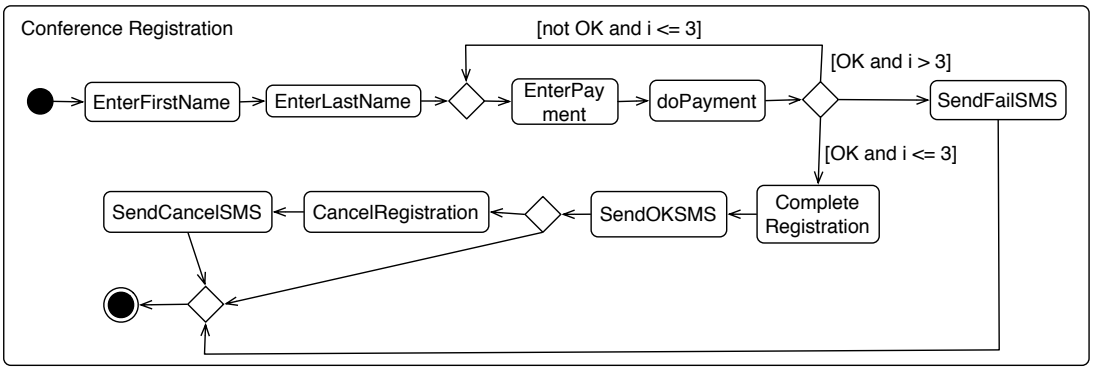


Figure 4: A Conference registration activity.

In a next step, the domain expert wants to express some properties of the activity such as: (1) whenever there is a `CompleteRegistration` action, it needs to be preceded by a `doPayment` action; (2) each `CompleteRegistration` action needs to be immediately followed by the `SendOKSMS` action. We will use the Process Pattern Specification Language (PPSL) presented in [FESS06, FESS07][1] for expressing properties on the Conference Registration activity. The PPSL is a visual language extending the UML 2.0 Activity Diagrams metamodel with stereotypes and new metamodel elements enabling the expression of so-called *process patterns*. These process patterns represent constraints of the models. We introduce the most important concepts of the PPSL in this paper and refer to [FESS06] for the complete set of language elements and an in-depth discussion of these elements.

The PPSL introduces an AllNode element enabling to refer to all occurrences of a certain Action in the pattern. The rationale for this element is the following. Actions can occur multiple times in an activity. If all occurrences of an Action shall be referred to in the pattern at the same time, this can be expressed writing an ≪all≫ stereotype on an Action. Recall the constraint that *each `CompleteRegistration` action needs to be immediately followed by the `SendOKSMS` action*. By applying the ≪all≫ stereotype to the `CompleteRegistration` action of the pattern, this action refers to all actions `CompleteRegistration` at the activity level. Consequently, all execution instances of `CompleteRegistration` in the activity have to be followed immediately by the action `SendOKSMS`. The pattern expressing this constraint is shown in the right middle of Figure 5. The multi-node used in that figure is a visualisation option

[1] The author of the current paper contributed to both publications.

of the AllNode.

A normal ActivityEdge in a pattern activity diagram means that Actions are tightly connected and there may no other Actions or control flows be executed between them. The PPSL introduces a more flexible ActivityEdge between Actions. In PPSL it is possible to put the stereotype ≪after≫ on an ActivityEdge expressing that there may be several Actions in between the ones connected by the specified ActivityEdge. Recall the constraint that *whenever there is a* `CompleteRegistration` *action, it needs to be preceded by a* `doPayment` *action*. The corresponding pattern is shown in the left middle of Figure 5. It expresses that the action `CompleteRegistration` must occur after the action `doPayment` but it does not enforce that the action `doPayment` must be immediately followed by an action `CompleteRegistration`. Several other actions may occur in between these actions. Remark that our activity shown in Figure 4 fulfills both constraints.

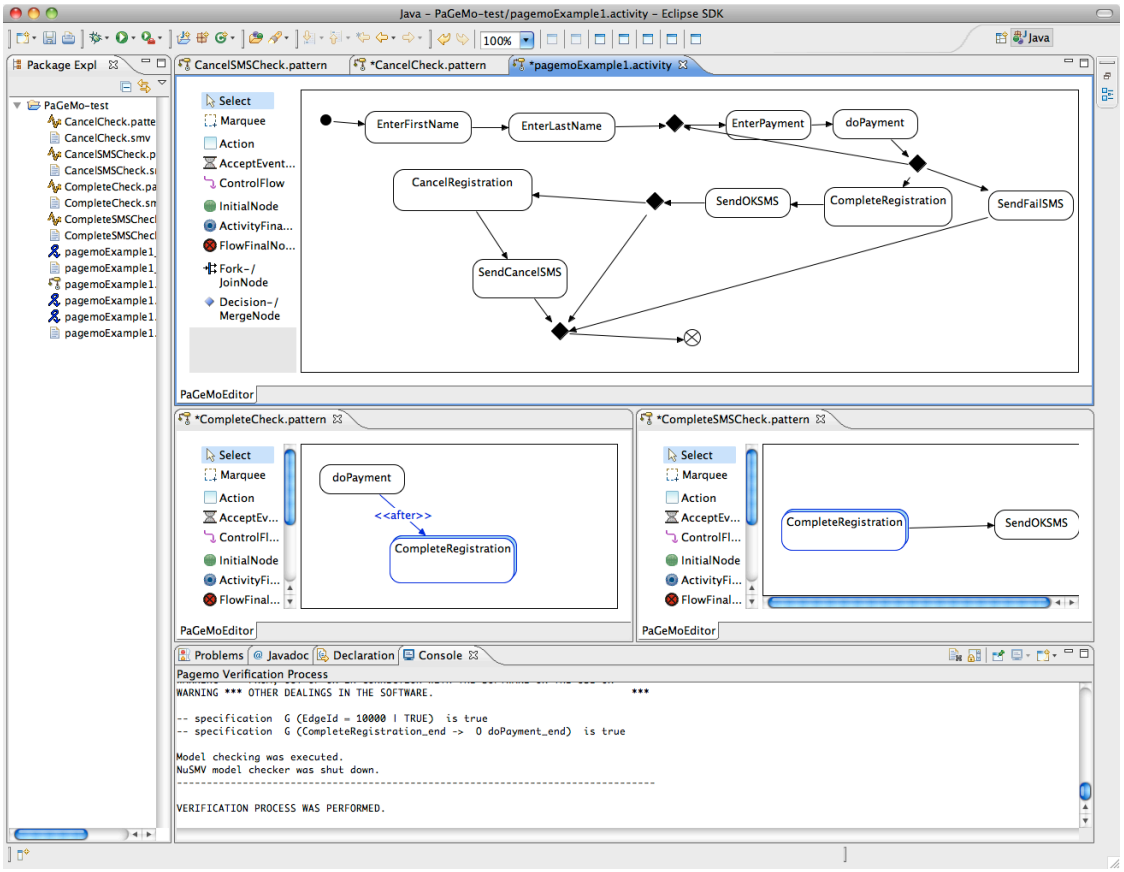


Figure 5: Screenshot of PaGeMo modelling the conference registration activity.

### 3.3 Application of the Methodology

In order to verify the above specified properties over the conference registration activity, we apply our methodology. In a first step, the domain expert needs to determine the property configurations. In this case there are two properties giving rise to two separate property configurations. Both properties need to be checked over the given activity. This results in a property configuration consisting of the conference registration activity and the first property and a second property configuration consisting of the conference registration activity and the second property.

In a second step the semantic domain needs to be determined for the property configurations. The properties are requirements for the choice of the semantic domain. The PPSL patterns are logical and temporal constraints. Consequently in the PPSL approach the semantic domain for the processes is the Labeled Transition System (LTS). The approach uses the Dynamic Meta-Modelling (DMM) [EHHS00] framework. DMM enables the specification of a set of graph transformation rules capturing the behaviour of the semantical concepts of the language. Given the set of DMM rules for a particular language, in this case the UML Activity Diagrams language, and a user-defined model expressed in this language, a LTS is generated by a DMM interpreter reflecting all possible behaviours to the model. In the PPSL approach, given a user-defined UML activity diagram, a LTS is generated that specifies the exact execution paths of the activity diagram. Two remarks need to be made. First, constraints on activity edges are not supported by this approach. Our scenario expressed in Figure 4 uses constraints to check the success of the payment actions. In the PPSL approach we use a simplified version of this scenario, one without these constraints as shown in the upper part of Figure 5. This results in a scenario with possibly infinite execution paths. Remark also that the Actions defined by our CWL (e.g., *EnterName*, *CompleteRegistration*, ...) are treated as a regular UML *Action*. If these actions require different or additional behaviour wrt the UML *Action* meta-class, new DMM rules need to be specified. The third activity of our methodology is in this particular case not treated explicitly because the DMM approach provides the transformations to translate the Activity Diagrams into labeled transition systems.

The semantics of the patterns expressed in PPSL is provided by temporal logic formulae because the patterns are logical and temporal constraints. We now introduce the semantics of the constraints defined above. For a complete overview of the formalization of the patterns we refer to [FESS07]. The temporal connectives used are: F to denote *some Future state*, G to denote *all future states*, X to denote *the neXt state*, O to denote *previously* and Y to denote *the previous state*. The first constraint results in the following LTL formula: $G(\texttt{CompleteRegistration} \rightarrow X\,\texttt{SendOKSMS})$. This formula expresses whenever an Action `CompleteRegistration` is executed it is followed immediately by the execution of an Action `SendOKSMS`. The second constraint results in the following LTL formula: $G(\texttt{CompleteRegistration} \rightarrow O\,\texttt{doPayment})$. This formula expresses whenever an Action `CompleteRegistration` is executed an action `doPayment` was executed previously.

The PPSL approach comes with an integrated workbench PaGeMo implemented as an Eclipse plugin for modelling and verifying PPSL patterns. A screenshot of this workbench showing our (simplified) activity and the two constraints is presented in Figure 5. The workbench allows the domain expert to model activity diagrams and to express several properties in PPSL. The conformance of the modelled activity diagram with selected properties is checked automatically,

i.e., an LTS is generated, this system is automatically translated into the NuSMV model checker [CCGR99], the properties are automatically translated into LTL formulas and the model checker is started. The result of the model checker is shown at the bottom of the workbench as shown in Figure 5. If the property is violated by the activity diagram the cause of the violation is indicated on the property by using different colors. Consequently a rough form of back-annotation is provided by the tool.

## 4 Discussion

Several approaches that propose a way to define semantics for domain-specific modelling languages exist. As mentioned in the introduction, most of these approaches are translational approaches [LV04, WP10, NAD03, RGLV09, CSN05, PS07]. In [Wac08] an interpretational approach relying on MDE techniques is proposed for specifying structural operational semantics of domain-specific modelling languages. Our proposed methodology is related to the approach presented in [GLMD09] which focusses on back-annotation but describes a more general approach on developing domain-specific languages. We intend to apply our methodology on the approach presented in [GLMD09]. Furthermore, we would like to refine and validate our approach on the existing approaches defining semantics of domain-specific languages and on larger cases and cases involving different DSLs. Special attention will go to the validation of our methodology in case properties need to be combined over models expressed using different DSMLs. To guide the domain expert through the process of defining semantics, tool support for the methodology needs to be developed.

Currently correctness of the methodology (*Does our methodology reflect what software engineers actually do?*) and correctness of the methodology execution (*Do software engineers follow our methodology?*) remain invalidated. In the scope of this workshop we are first of all asking for feedback on our proposed methodology, next we are searching for cases to validate our methodology.

Our proposed methodology is similar to the methodology for specifying and verifying consistencies of object-oriented behavioural models proposed by Engels et al. [EKHG01, EHK01]. Their methodology applies to the specification of consistency constraints. These constraints can be seen as properties of the models in our approach.

## 5 Conclusion

This paper proposes the first ideas for a methodology for expressing semantics of domain-specific models through properties. We motivate this methodology by the observation that in general different semantic domains are used depending on the properties domain experts want to verify over their models. In most cases only the model elements of interest or the model elements of interest are given a precise meaning and this because domain experts want to define and check certain characteristics of the domain-specific models. Allowing such local and partial semantics specifications has the advantage of being able to specify semantics of different elements of the models depending on the considered activity of the software development lifecycle.

**Acknowledgements:** We would like to thank Prof. Hans Vangheluwe and Bart Meyers of the University of Antwerp for the nice discussions on this topic.

## Bibliography

[CCGR99] A. Cimatti, E. M. Clarke, F. Giunchiglia, M. Roveri. NUSMV: A New Symbolic Model Verifier. In Halbwachs and Peled (eds.), *Computer Aided Verification, CAV '99*. Lecture Notes in Computer Science 1633, pp. 495–499. Springer, 1999.

[CSN05] K. Chen, J. Sztipanovits, S. Neema. Toward a semantic anchoring infrastructure for domain-specific modeling languages. In Wolf (ed.), *5th ACM International Conference On Embedded Software, Proceedings*. Pp. 35–43. ACM, 2005.

[EHHS00] G. Engels, J. H. Hausmann, R. Heckel, S. Sauer. Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML. In Evans et al. (eds.), *UML 2000*. Lecture Notes in Computer Science 1939, pp. 323–337. Springer, 2000.

[EHK01] G. Engels, R. Heckel, J. M. Küster. Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model. In Gogolla and Kobryn (eds.), *UML 2001*. Lecture Notes in Computer Science 2185, pp. 272–286. Springer, 2001.

[EKHG01] G. Engels, J. M. Küster, R. Heckel, L. Groenewegen. A methodology for specifying and analyzing consistency of object-oriented behavioral models. In *ESEC / SIGSOFT FSE*. Pp. 186–195. 2001.

[FESS06] A. Förster, G. Engels, T. Schattkowsky, R. V. D. Straeten. A Pattern-driven Development Process for Quality Standard-conforming Business Process Models. In *VL/HCC 2006*. Pp. 135–142. IEEE Computer Society, 2006.

[FESS07] A. Förster, G. Engels, T. Schattkowsky, R. V. D. Straeten. Verification of Business Process Quality Constraints Based on Visual Process Patterns. In *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007*. Pp. 197–208. IEEE Computer Society, 2007.

[FR05] R. France, B. Rumpe. Domain specific modeling, Editorial. *Journal on Software and System Modeling* 4(1):1 – 3, 2005.

[GLMD09] E. Guerra, J. de Lara, A. Malizia, P. Díaz. Supporting user-oriented analysis for multi-view domain-specific visual languages. *Inf. Softw. Technol.* 51(4):769–784, 2009.

[Hol95] D. Hollingsworth. The Workflow Reference Model. http://www.wfmc.org/standards/docs/tc003v11.pdf, 1995.

[HR04] D. Harel, B. Rumpe. Meaningful Modeling: Whats the Semantics of Semantics? *IEEE Computer* 37(10):64 – 72, 2004.

[HVV08] Z. Hemel, R. Verhaaf, E. Visser. WebWorkFlow: An Object-Oriented Workflow Modeling Language for Web Applications. In Czarnecki et al. (eds.), *11th International Conference, MoDELS 2008, Toulouse, France. Proceedings*. Lecture Notes in Computer Science 5301, pp. 113–127. Springer, 2008.

[LV04] J. de Lara, H. Vangheluwe. Defining visual notations and their manipulation through meta-modelling and graph transformation. *J. Vis. Lang. Comput.* 15(3-4):309–330, 2004.

[MV10] R. Mannadiar, H. Vangheluwe. Modular Synthesis of Mobile Device Applications from Domain-Specific Models. Technical report 2.5, McGill University, Quebec, Canada, 2010.

[NAD03] J. Niu, J. M. Atlee, N. A. Day. Template Semantics for Model-Based Notations. *IEEE Trans. Softw. Eng.* 29(10):866–882, 2003.

[Obj05] Object Management Group. Unified Modeling Language 2.0 Superstructure Specification. http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf, February 2005. www.omg.org

[PS07] J. Porter, J. Sztipanovits. Towards a Modeling Environment for Composing Domain-Specific Modeling Languages: A Case Study on Controlling Traffic Lights. In Prinz (ed.), *14th International ASM Workshop*. 2007.

[RGLV09] J. E. Rivera, E. Guerra, J. Lara, A. Vallecillo. Analyzing Rule-Based Behavioral Semantics of Visual Modeling Languages with Maude. In *SLE 2008*. Pp. 54–73. Springer-Verlag, Berlin, Heidelberg, 2009.

[Wac08] G. Wachsmuth. Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2007. In Lämmel et al. (eds.), *Modelling the Operational Semantics of Domain-Specific Modelling Languages*. Lecture Notes in Computer Science 5235, pp. 506–520. Springer, 2008.

[WP10] J. R. Williams, F. A. C. Polack. Automated Formalisation for Verification of Diagrammatic Models. *Electron. Notes Theor. Comput. Sci.* 263:211–226, 2010.