Electronic Communications of the EASST
Volume 21 (2009)

Proceedings of the 3rd International Workshop on Multi-Paradigm Modeling (MPM 2009)

DSL Composition for model-based test generation

Bruno Barroca, Levi Lucio, Didier Buchs, Vasco Amaral, Luis Pedro

10 pages

Guest Editors: T. Levendovszky, L. Lengyel, G. Karsai, C. Hardebolle
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

DSL Composition for model-based test generation

Bruno Barroca[1], Levi Lucio[2], Didier Buchs[3], Vasco Amaral[4], Luis Pedro[5]

[1] mailbrunob@gmail.com, [4] Vasco.Amaral@di.fct.unl.pt, [5] luismiguelpedro@gmail.com
http://citi.di.fct.unl.pt/
Departamento de Informática, Faculdade de Ciencias e Tecnologia
Universidade Nova de Lisboa, Portugal

[2] Levi.Lucio@unige.ch, [3] Didier.Buchs@unige.ch
http://smv.unige.ch
Université de Genève, Centre Universitaire d'Informatique
CUI - Battelle bat. A Route de Drize, 7 CH-1227 Carouge, Switzerland

Abstract: Domain specific languages (DSL) which describe reactive systems generally have a need for systematic generation of tests for their models. During the design of a DSL there is a lack of support for its integration with existing model based test generation tools. In this paper, we show how this integration can be conceptualized and systematized. We introduce a framework for composing DSLs for reactive systems, with a particular DSL for Model Based Testing called SATEL (Semi-Automatic Testing Language). This DSL composition is achieved by composing both the syntaxes of the two DSLs and their semantics. The result of this composition is also a language where it is possible to express models in the target DSL and test specifications for those models. The semantics of the composed language corresponds to the generation of test cases for models expressed in the target DSL. We finish the paper by analyzing the compositional framework we obtained in terms of its applicability to other target DSLs.

Keywords: Model based Testing, DSL composition, Inference Rules, Language Engineer, Test Generation.

1 Introduction

The present work lies within the research topic of integrating Model Driven Development (MDD) in the process of building domain specific languages (DSL). We address in particular languages which describe the behaviour of reactive systems. It is common to engineer a DSL by describing its syntax using appropriate metamodeling tools and its semantics using model transformation tools. Modern metamodeling tools, can help the software language engineer to easily prototype usable editors that can restrict and guide the domain experts in expressing valid models.

On the other hand, Model-Based Testing (MBT) is currently an established discipline within MDD. Normally MBT implies specifying some sort of test specification for a model expressed in a specification language (e.g. UML, state machines, logics or others). From the test specification and the model specification, test cases can be produced for an implementation of that model. An example of an MBT framework is defined in [LÓ9], where a language called SATEL was

developed in order to specify test intentions (also called in the literature test specifications) over specifications of reactive systems expressed in a petri-net based formalism [BG91]. Being a black box testing language, SATEL is adapted to generate test cases for reactive systems which interact with their environments.

In this work we describe how the test selection and generation mechanism of SATEL can be reused and integrated as a part of the process of engineering DSLs for describing reactive systems. Our main objective is to provide a methodology which natively includes a test selection mechanism in the development of any DSL. This will allow the engineer to directly generate tests from models expressed in the developed DSL. We will describe how this integration can be realized at the level of the metamodel of the involved languages by means of a DSL compositional framework called CoPsy[PAB08].

This paper is organized as follows: in section 2 we describe the problem of composing a test language with another DSL. In section 3 we identify which are the main issues in that composition. Section 4 lays down the formal foundations of the CoPsy composition framework we use in this paper. In section 5 we instantiate the CoPsy framework in order to explain the details of the composition of SATEL with an example DSL — in particular we use the HALL language [BA07] as the composition target. Section 6 provides a discussion on the Prolog implementation of our composition method as well on some issues raised by our approach. Finally section 7 concludes.

2 Motivation

In this paper, we consider the testing language SATEL as a DSL that we will compose with the DSL that describes models for which we wish to produce test cases — which hereon we call the *target* DSL. The compositional approach consists on merging both languages, at the syntactic and semantics levels. With this composed language, the engineer will be able to express the model of the system (s)he wishes to implement, as well as the test intentions for that model.

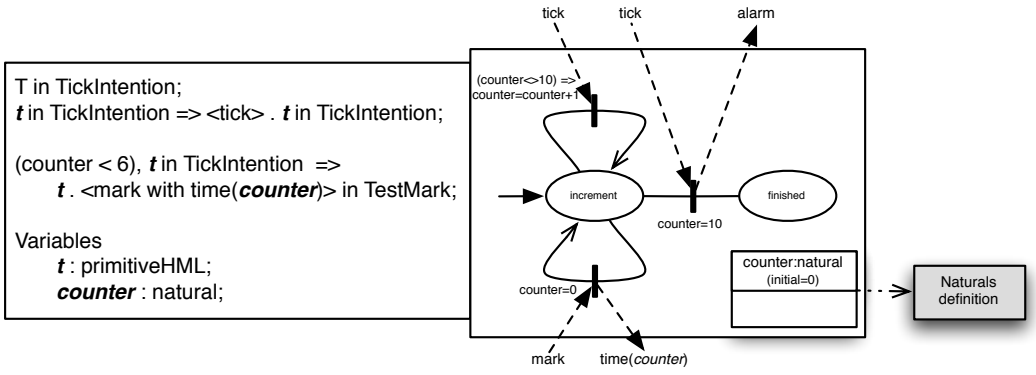

Figure 1: An instance model of the composed SATEL(HALL)

Test intentions in SATEL are written as formulas[1] representing test case templates with constrained variables. The domains of those variables correspond to the three abstract dimensions of a test case, namely: the shape of the sequences of input/output pairs; the kind of input/output pairs inside those sequences; and the parameters of the inputs and outputs. The constraints over the variables allow shaping the resulting *test cases*.

Figure 1 provides an example of the definition of a model, which is an instance of SATEL composed with a second DSL called HALL. A model of the composed language consists of two parts: a HALL model (on the right of figure 1) and a SATEL model (on the left of figure 1). Note that the composed language consists of the HALL DSL merged with a version of SATEL which, in this case, is parameterized by the syntax and semantics of HALL.

Although we do not provide in this paper a thorough description of the syntax and the semantics of HALL or SATEL, let us explain briefly the *timer system* model and it's associated test intentions shown in figure 1. This system specification defines two inputs (*tick* and *mark*) which respectively: advance a time counter by one unit; reset the counter while outputting, using the *time* output, the time reached before the *mark* event. If the timer reaches 10, an alarm is set. Notice that the *timer system* defined in HALL makes use of the type definition of *Naturals*.

In Figure 1 (on the left), two test intentions are declared, the *TickIntention* and the *TestInterruptions*. Both test intentions are meant to generate tests for the timer system. The keyworks *tick*, *mark* and *time* are in fact inputs and outputs taken from the HALL model, as defined by the composition of the two DSLs. The test intention *TickIntention* includes two axioms which recursively build sequences of *tick* inputs, declared in the HALL part of the specification. Note that the axiom '*T in TickIntention*' declares the empty test case, while the axiom '*t in TickIntention* => <*tick*> . *t in TickIntention;*' recursively concatenates — by using the '.' operator — *tick* inputs to already formed test cases.

On the other hand, the *TestInterruptions* test intention concatenates sequences of *tick* inputs obtained with a final input/output pair '<*mark with time(***counter***)*>'[2], where counter is a variable. The axiom for the *TestMark* test intention includes the condition *(counter<6)* — which we can express in a test intention because the composed language can use the *Naturals* definition from HALL. This means that from this test intention five valid test cases will be generated, according to the semantics of the timer system:

```
<tick> <mark with time(1)>
<tick> <tick> <mark with time(2)>
...
<tick> <tick> <tick> <tick> <tick> <mark with time(5)>
```

3 Concepts to be Composed

Having in mind the example in section 2, let us now identify which parts of SATEL are parameterized by the target DSL. We start by the syntactic aspects of the composition:

- SATEL is a language for black-box testing, and as such requires knowledge of the signatures of the types inputs and outputs in the models that are going to be tested;

[1] In HML which is a very simple temporal logic — including the *next*, *and* and *not* operators — adapted to representing the observation of a System Under Test.

[2] The **with** keyword separates *inputs* from *outputs* in SATEL syntax. Notice also that an input may exist without output as in the case of the *tick* input in test intention *TickIntention*.

- The data types coming from the target language provide the types for the parameters of the inputs and outputs in test intentions. They also provide the syntax for expressing conditions that reduce the domains of the variables that are part of the test intention.

In semantic terms the following concepts are part of the composition:

- To produce test cases it is necessary to understand whether those tests describe behaviours that should be accepted or not by the implementation — called the *oracle* problem. This satisfaction is checked for each test case by verifying if the sequence of input/output pairs it describes corresponds to the semantics of the model. It is thus necessary to merge with the semantics of SATEL the rules that define the semantics of models in the target DSL;

- In order to narrow the number of produced test cases by a test intention, SATEL allows the expression of conditions over the domains of the variables included in test intentions. In particular, the conditions over the parameters of inputs and outputs are solved using the semantics of the types in the target language. In other words, the descriptions of the semantics of the types used in the target language are composed with SATEL's semantics in order to solve the constraints in certain variables that may be expressed in the test intentions.

We will now introduce the methodology and the tool we use to formally compose SATEL with any given DSL, according to the composition concepts presented above.

4 Methodology and Foundations

The composition framework we use in this paper was introduced by Pedro in [PAB08]. The framework consists of defining a new language c as the composition of two other languages a and b. On the one hand, the syntax of language c is given by the composition of the metamodels of a and b. On the other hand, in CoPsy, semantics are given by transformation rules. Transformation rules take instances of a language without direct semantics and produce new models in another language for which the semantics is known. This said, the semantics of language c is given by the composition of the transformation rules of a and b. Figure 2 presents an overview of the approach. The figure encompasses three main concepts:

Metamodels: we can distinguish three types of metamodels: *parametric*, *effective* and *target*. The *parametric* metamodel corresponds to the abstract syntax of a language with 'holes'. These 'holes' are called formal parameters (denoted *fp* in the picture). During the composition, the *effective* parameter metamodel (denoted *ep* in the picture) will fill the 'holes' defined in the *parametric* metamodel. The *target* metamodel defines the abstract syntax of the language for which the semantics are known;

Models: models are instances of the metamodels, as is denoted by the *instanceOf* relations in figure 2;

Transformations: the transformations are rules that match elements of a source model and transform them into elements of a target model. Intuitively, the target model is the one for which the semantics are known. Some transformations also have 'holes', more precisely formal parameters. For example in figure 2, the transformation tr_{fp} includes a formal parameter. By replacing

the formal parameters in transformations with effective ones defined in other transformations we are able to compose the semantics of the languages involved in the composition.

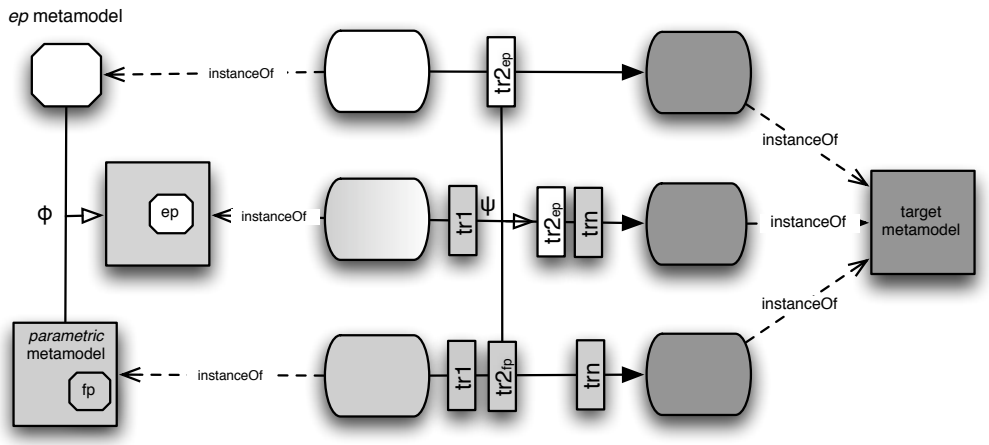

Figure 2: CoPsy Overview

Figure 2 thus represents the whole process of composing two languages. The left row of the image depicts the syntax and semantics of the first language in the composition. This language is parameterized by the second language depicted in the right row of the image. This parameterization is achieved by mapping the effective parameters into the formal ones by the ϕ and the ψ functions. On the one hand, ϕ works at the level of the metamodels of the two languages by specifying how the elements of the *parametric* metamodel which are formal parameters are replaced by the elements of the *ep* metamodel. On the other hand, ψ defines how transformations of the *parametric* metamodel having formal parameters are changed using the transformations of the *ep* metamodel which are effective parameters.

5 CoPsy framework instantiation

The previously identified concepts are now composed by using the CoPsy framework. We take HALL as an example of a target DSL. HALL is a suitable language for the composition with SATEL, since it describes reactive systems, and uses algebraic type definitions. Moreover, its syntax was defined by using a metamodel, and its operational semantics was defined by a set of inference rules. The DSL composition is done at both the languages' metamodel (abstract syntax) and transformation (semantic) levels.

5.1 Composition at the syntax level

Both the SATEL's and HALL's syntaxes are expressed using EMF/ECore metamodels. Due to the space constraints we only present here a subset of the real metamodels of both languages. We start the syntax composition of these languages, by identifying in the SATEL's metamodel

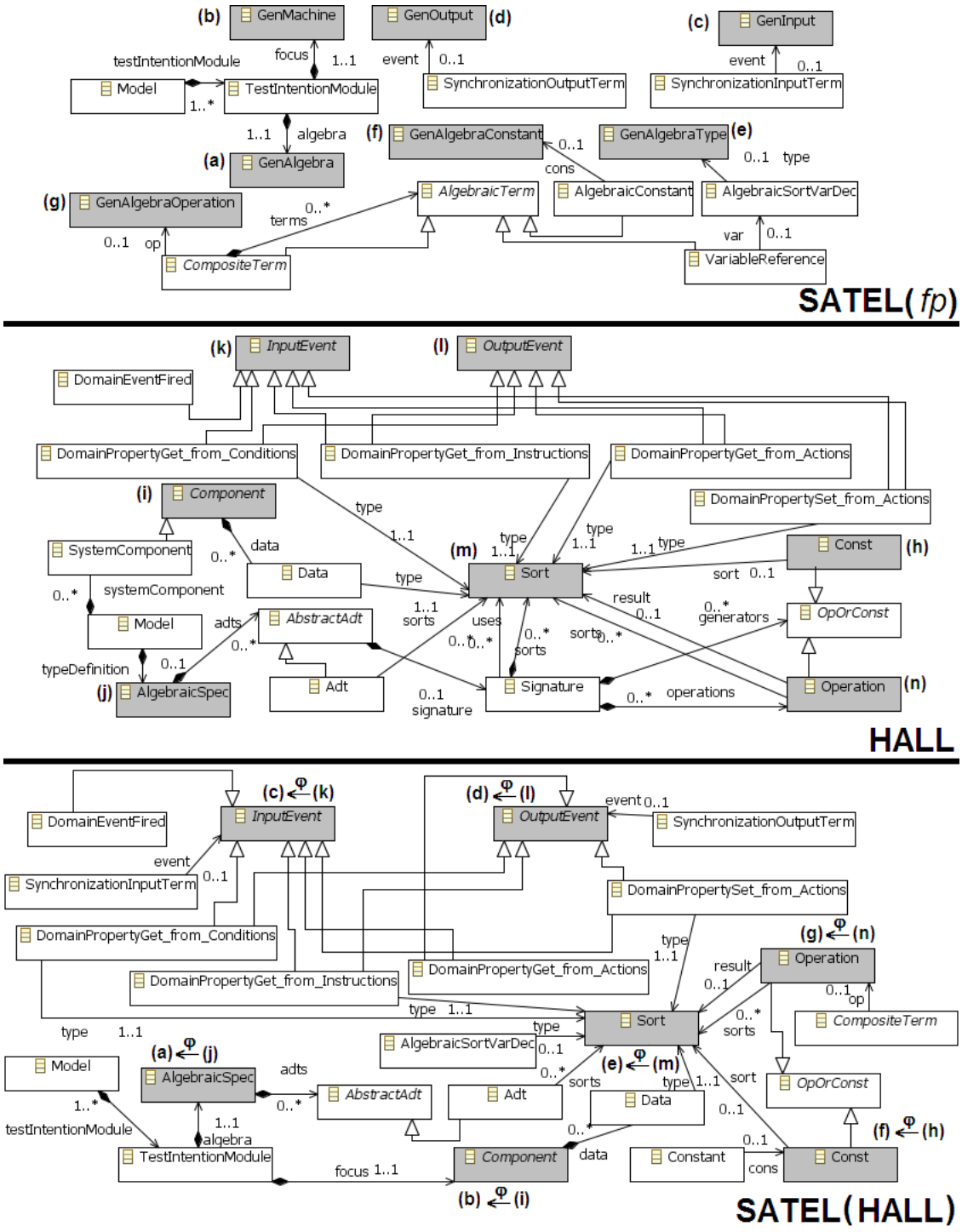

Figure 3: Composition of the SATEL and HALL at the metamodel level

the elements that will belong to the formal parameter *fp* of the composition. As shown in Figure 3 (in the top layer), the meta-classes identified in light gray belong to the formal parameter *fp* of the composition. In our concrete example, see Figure 3 - in the middle layer, we chose terms from the HALL language to form the effective parameter of the composition.

Figure 3 - in the bottom layer, shows the resulting metamodel obtained by the application of the mapping function φ to the metamodel parameterization. The meta-class *GenMachine* (b) was substituted by the meta-class *Component* (i) - which means that the models of the composed language can express HALL components and state machines. The meta-class *GenAlgebra* (a) was substituted by the meta-class *AlgebraicSpec* (j) - which means that we can now produce models in the composed language that defines concrete algebraic specifications that will be used both on HALL components and SATEL intentions specifications. In the composed language, when we want to declare variables of an algebraic type, we must now point to its respective *Sort* (m) declaration. Also, if we want to use some algebraic operation, or use some algebraic constant in our test intentions, we must now point to their respective declarations by referring declaration elements of type *Operation* (n) or *Const* (h). Finally, whenever we want to reference in our test intentions some input or output events from our HALL specification, we will now have to point to their respective declarations by referring elements of type *InputEvent* (l) or *OutputEvent* (k).

5.2 Composition at the semantics level

Firstly, the semantics of each of these DSL's (both SATEL and HALL) is described by means of transformations into a common target language of inference rules. In order to use inference rules as a target language of model transformation tools, we also defined an EMF-based metamodel for the language of Inference Rules. For instance, a SATEL transformation rule will match model elements which are conformant with the SATEL's metamodel, and will produce model elements which are conformant with the Inference Rules Metamodel. We chose the language of Inference Rules because it consists in an elegant mean for specifying the operational semantics of a language, and it has a well defined semantics which is based on formal logics. After all the transformation rules are executed, the inference rules that they produce can be also computed. While in operation, the whole set of inference rules generates different kinds of relations that defines the language's semantics.

The most important semantic relation in this work is generated by the inference rule *ruleTestGen*, shown in Figure 4. This particular inference rule was produced by a transformation rule which does not depend on any kind of model instance of SATEL. The transformation rule that produced *ruleTestGen* had formal parameters which were replaced during the composition with HALL according to the ψ mapping function [3].

The rule *ruleTestGen* states that we can deduce that a given grounded [4] SATEL *HMLformula* belongs to the *Tests* relation if: there exists a tupple $\langle condition, HMLformula \rangle$ from the Intentions relation (premise (1)); the grounded conditions over the *HMLformula* are satisfied given the semantics of *HALLTypes* (premise (2)); and this grounded *HMLformula* is satisfied by the semantics of *HALLSemantics*, given an initial state *HALLinit* (premise (4)).

[3] Due to space constraints, in this paper, we will not further detail the ψ mapping function.

[4] Ground expressions have no free variables: this means that there exists an *isGround* inference rule, which operationally generates values to replace the free variables inside these expressions.

$$ruleTestGen\frac{\begin{array}{l}(1)\ \langle conditions, HMLformula\rangle \in Intentions,\\ (2)\ HALLTypes \models_{alg} conditions,\\ (3)\ HALLSemantics, HALLinit \models_{sat} HMLformula\end{array}}{HMLformula \in Tests}$$

Figure 4: Inference Rule for test case generation in SATEL(HALL) semantics

Premise (1) is satisfied by the Intentions relation which is generated by inference rules produced by transformation rules which match SATEL elements, e.g SATEL conditions and HML formulas.

Premise (2) $\models_{alg}$ satisfaction depends on the *HALLTypes* relation which is generated by inference rules produced by transformation rules which match elements of HALL algebraic type definitions, e.g. sorts, constants, operations, equations.

Premise (3) $\models_{sat}$ satisfaction depends on the *HALLSemantics* relation, which constitutes the oracle for the test cases. This relation is generated by inference rules produced by a set of transformation rules which do not depend on any kind of HALL model (this constitutes the step semantics of the language). It also depends on the *HALLinit* relation, which is generated by inference rules produced by a set of transformation rules matching HALL elements, e.g. state machines, variable definitions, HALL specific inputs, HALL specific outputs.

Notice that all the transformation rules referring to HALL elements involved in Premise (2) and (3) substituted the tr_{fp} transformation rules from the parameterized SATEL that existed before the composition. Also, this substitution occurred according to the ψ mapping function.

6 Discussion

We have implemented a composition of SATEL and HALL using EMF and Prolog. At the syntactic level, we manually composed the EMF metamodels of both languages, by merging their abstract syntaxes as we described in Figure 3. Semantically, the composition of both DSLs was made possible by leveraging both the language's syntax and semantics into a common ground: the Prolog clauses. On the one hand, both DSL's semantics were expressed by inference rules encoded into Prolog clauses. On the other hand, we developed a generic mechanism which automatically translates any model expressed on a given EMF-based DSL into a Prolog fact. This prolog fact contains both the structure and the initial state of the system. By merging all these clauses, we are able to explore the semantics of both DSLs: in HALL this means to explore the transition system of the model, and in SATEL this means to generate test cases using that transition system as an oracle. More precisely, test cases are generated by: expanding test intentions into intermediate formulas that solve recursion and test intention composition; finding substitutions for remaining variables in those formulas by solving conditions associated to the test intention; "playing" the ground formulas (now test cases) against the operational semantics of the HALL model to check if that model satisfies the formulas — thus binding the oracle to each test case.

In our implementation, we formalized the metamodels for each of the languages involved in the composition, as well as the metamodel of the composed language. However, we did not ex-

plicitly formalise the transformation rules described in chapter 5, although we can see the EMF to Prolog transformation mechanism as a transformation rule. Also, we did not explicitly formalise any composition rules - in the DSL's metamodels or transformations. In order to generalize the composition mechanism to other DSLs, we found the need to introduce the composition framework CoPsy. To do so, we first had to explicitly formalise the transformation rules that provide inference rules semantics to both SATEL and HALL. Then, we formalized the formal parameters of the composition, both in SATEL's metamodel and in the transformation rules. Finally, we identified what are the effective parameters from HALL's metamodel and transformation rules, and established their mappings with the formal parameters in SATEL.

However, there are several issues that can be challenging while applying the current compositional framework to provide a test selection and generation mechanism to other DSLs. Firstly, the integration of a target DSL in this solution involves that that DSL has a defined step/operational semantics expressed in terms of inference rules which are able to perform the described $\models_{alg}$ and $\models_{sat}$ deductions in a finite time.

Secondly, since SATEL needs to generate substitutions for the variables in test intentions and solve conditions over these variables, we also need to have access to the semantics of the types that the target DSL uses. In our experiment, to solve this particular problem, we have used algebraic type specifications.

As we've mentioned before, the new composed language corresponds to the union of HALL — which remains unchanged in the composition — with a parameterized version of SATEL. The scope of SATEL is adapted to HALL so that inputs and outputs of this language can be used in SATEL. Also, the semantics of SATEL's test generation (oracle computation) is adapted to HALL's semantics. An issue in the presented example is that the conditions over SATEL variables can be expressed and computed because the syntax and semantics of types used in HALL is fully described by algebraic specifications. Target languages that do not use algebraic specifications must provide some other fashion of describing the syntax and semantics of those types. Depending on this description, the expressivity/usability of the parameterized SATEL can be affected.

This work was inspired by another example [LÓ9] of an implicit composition of SATEL with another language which describes reactive systems, called COOPN. Another example of implicit composition of model based testing languages with another language is presented in SpecExplorer [CGN+05]. SpecExplorer includes a set of tools that enables test generation from specifications expressed in Spec#. In the above examples, the composition of model based testing languages with another language is made implicitly and no reflection is provided on the composition itself.

There are some other composition frameworks that like CoPsy that are also able to perform DSL composition both at the syntatic and semantic levels. For example, in [IP04] a technique and a tool are presented that allow composing a system as a set of properties of subsystems defined as transformations. A technique called semantic anchoring is presented in [CSAJ05]. Semantic anchoring consists of attaching semantics to a metamodel in terms of a computational model, e.g. Abstract State Machines. However, none of these composition frameworks focus the problem of DSL reuse, nor provide a clear definition of the composition based on mapping functions at the metamodel and transformation levels. To our knowledge, there is no existing compositional framework that enables the reuse of test selection and generation mechanisms

within the development of a DSL.

7 Conclusion

In this paper, we proposed a methodology for the composition of a model based testing language called SATEL with other DSLs. Using CoPsy as a composition framework, we extracted a composition interface from the SATEL language, both at the syntactic and semantic levels. We have also presented an example where we composed SATEL with HALL. Currently, there is an implementation of the CoPsy framework, which is able to automate DSL composition. As future work, we plan to use this implementation in order to automate as far as possible the integration of SATEL test selection and generation mechanism in the process of DSL engineering.

7.1 Acknowledgements

The presented work has been developed in the context of project BATIC3S partially funded by the Portuguese foundation FCT/MCTES ref. PTDC/EIA/65798/2006, and the doctoral grant ref. SFRH/BD/38123/2007.

Bibliography

[BA07] B. Barroca, V. Amaral. (H)ALL: a DSVL for designing user interfaces for Control Systems. In *Proceedings of the 5th Nordic Workshop on Model Driven Engineering NW-MoDE*. Blekinge Institute of Technology, Ronneby, Sweden, 2007.

[BG91] D. Buchs, N. Guelfi. CO-OPN: A Concurrent Object-Oriented Petri Nets Approach for system specification. In Silva (ed.), *12th International Conference on Application and Theory of Petri Nets*. Pp. 432–454. Aahrus, Denmark, June 1991.

[CGN+05] C. Campbell, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, M. Veanes. Testing Concurrent Object-Oriented Systems with Spec Explorer. In *FM*. Pp. 542–547. 2005.

[CSAJ05] K. Chen, J. Sztipanovits, S. Abdelwalhed., E. Jackson. Semantic anchoring with model transformations. *Model driven architecture* 3748, 2005).

[IP04] M. C. P. Inverardi, P. Pelliccione. Compositional Verification of Middleware Based Software Architecture descriptions. In *Proceedings of the International Conference on Software Engineering, Scotland, UK*. 2004.

[LÓ9] L. Lúcio. *SATEL — A Test Intention Language for Object-Oriented Specifications of Reactive Systems*. PhD thesis, Université de Genève - Switzerland, 2009. Available as http://smv.unige.ch/members/levi-lucio/files/coopn_testing.pdf.

[PAB08] L. Pedro, V. Amaral, D. Buchs. Foundations for a Domain Specic Modeling Language Prototyping Environment: A compositional approach. In *Proceedings of the 8th OOPSLA ACM-SIGPLAN Workshop on Domain-Specic Modeling (DSM)*. 2008.