Verification and Validation of Meta-Model Based Transformation from SysML to VHDL-AMS

Jean-Marie Gauthier, Fabrice Bouquet, Ahmed Hammad and Fabien Peureux
FEMTO-ST Institute / DISC department - UMR CNRS 6174
16, route de Gray, 25030 Besançon, France.
{*jmgauthi, fbouquet, ahammad, fpeureux*}*@femto-st.fr*

Keywords: SysML, VHDL-AMS, Verification, Validation, Unit Testing, Meta-Model Transformation, Code Generation

Abstract: This paper proposes an approach to verify SysML models consistency and to validate the transformation of SysML models to VHDL-AMS code. This approach is based on two main solutions: the use of model-to-model transformation to verify SysML models consistency and writing unit tests to validate model transformations. The translation of SysML models into VHDL-AMS simulable code uses MMT (Model to Model Transformation) ATL Atlas Transformation Language and M2T (Model To Text) Acceleo tooling. The test validation of the model transformations is performed using EUNIT framework.

1 CONTEXT AND MOTIVATION

Critical heterogeneous systems design is a complex activity since such systems are often composed of mixed hardware and software components. This complexity makes it difficult to validate the conception of the entire system in regards of its requirements. Moreover, strengthened by the growing complexity and intensive use of such systems, the validation activity still defines a key challenge for engineering practices. It is then crucial to set up a process in order to verify and simulate the system model at the earliest stage of the design process. In this context, the Object Management Group (OMG) promotes the SysML language (Friedenthal et al., 2009) to model heterogeneous systems, while the IEEE association proposes the standard VHDL-AMS (Christen and Bakalar, 1999) to describe and simulate hardware components. In this paper, we propose an approach based on Model-Driven Engineering (MDE) techniques to verify and transform SysML models into VHDL-AMS code.

The first goal of this approach was to define a subset of the SysML meta-model and to design the VHDL-AMS meta-model addressed by the SysML subset. The transformation between these meta-models, called *SysML2VHDLAMS*, was introduced in (Gauthier et al., 2012). The second goal is to verify the SysML model consistency in regards of design rules. The third goal is to validate the transformation process using unit testing techniques. These last two goals define the contributions of this paper.

The paper is decomposed into five main parts. Section 2 introduces some backgrounds about languages and MDE techniques. Section 3 describes the approach we used to verify the SysML model consistency in regards of design rules. In Section 4, we present white-box testing method to validate the transformation. Finally, Section 5 and 6 give some related works, conclude and outline our future work.

2 PRELIMINARIES

This section introduces languages, tools and concepts of model-driven engineering in order to generate VHDL-AMS code from SysML models.

2.1 SysML Language

SysML (Friedenthal et al., 2009) is a profile of UML 2.0 (Rumbaugh et al., 2004) standardized by the Object Management Group (OMG) and the International Council on Systems Engineering (INCOSE). SysML consists of several diagrams, which are requirement, use case, sequence, activity, block, internal block, constraint blocks, parametric, state machine and allocation. It supports the specification, design, verification and the validation of a broad range of systems.

We use a subset of SysML expressiveness to represent requirements and to model components of a system with their interactions. Therefore, we use elements that are part of the Requirements, Block Definition, Internal Block and Parametric Diagrams.

2.2 VHDL-AMS Language

VHDL-AMS (Christen and Bakalar, 1999) is the IEEE standard modeling language created to provide compatibility and capability in an open language for modern analog, mixed-signal and multi-domain designs. It provides both continuous-time and event-driven modeling semantics. VHDL-AMS also facilitates modeling of multi-domain systems that can include (among others) a combination of electrical, mechanical, thermal, hydraulics, and magnetic models. This allows the entire system to be modeled, simulated, analyzed, verified and optimized.

2.3 The ATL Language

ATL (ATL,) is a model transformation language and toolkit initiated by the AtlanMod team (previously called ATLAS Group). In the field of MDE, ATL provides ways to produce a set of target models from a set of source models. An ATL transformation program is composed of rules that define how source model elements are matched and navigated to create and initialize the elements of the target models. An ATL transformation can be decomposed into three parts: a header, helpers and rules. Rules are the heart of ATL transformations because they describe how output elements (based on the output meta-model) are produced from input elements (based on the input meta-model). They are made up of bindings, each one expressing a mapping between input and corresponding output element.

2.4 Acceleo Language

Acceleo (Acceleo,) is a language specialized on code generation based on Eclipse Modeling Framework (EMF) models. The Acceleo language defines templates to control the output generation. Most of the time, the output is a programming language. Acceleo requires the definition of EMF meta-models and one or more templates that translate the model into text.

2.5 EUnit Framework

EUnit (Garcia-Domnguez et al., 2011) is a unit testing framework based on the Epsilon platform. It is designed for testing model management tasks, such as model-to-model transformations, model-to-text transformations or model validations. It notably provides convenient tools for testing ATL transformation. Moreover, EUnit unit tests can be executed automatically in a continuous integration process.

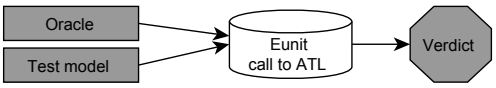

Figure 1: EUnit workflow.

Figure 1 shows the workflow of the EUnit testing framework. Before writing a rule, the engineer creates a SysML model (test case) and a VHDL-AMS model (expected result of the transformation). These two models are next used by EUnit (by calling ATL). Finally, EUnit compares the result of the ATL transformation with the expected result: if they are equal, the verdict is true, false otherwise.

3 MODEL VERIFICATION

This section concerns the verification of the model consistency before generating VHDL-AMS code. The issue of the verification raises two questions:

- How to ensure that the generated VHDL-AMS code does not contain syntactic and semantic errors? For instance, the naming of a component with reserved words of the VHDL-AMS language (syntactic error) or the connection of two ports with different types (semantic error).
- How to provide to the final user, *i.e.* system engineer, a SysML modeling guideline to generate correct VHDL-AMS code?

To answer these questions, we provide a verification approach based on the ATL technology by generating a list of problems written in a XMI file. This file is then parsed in order to show problems in an user interface. As ATL provides a model-to-model transformation based on meta-model, we define the meta-model that allows to represent a problem.

3.1 The Problem meta-model

The problem meta-model of the Figure 2 is the target of a transformation, which is called *SysML2Problem*. The meta-class *Problem* represents a problem with its location in the SysML file, its description and its severity. The severity of the problem is represented by the enumeration *Severity*, which could be *warning*, *error* or *critic*.

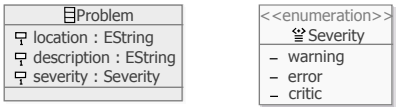

Figure 2: Problem meta-model.

3.2 SysML to Problem Transformation

To illustrate this transformation, we propose to study the case of reserved words. We wish to forbid the naming of SysML elements with reserved words of VHDL-AMS. A rule named *uncorrectNameReserved-Word* is defined in the ATL file. This rule, see Figure 3, generates a problem only if the precondition *umlElement.isReservedWord()* is evaluated to true. This precondition uses an ATL helper named *isReserved-Word()*, whose aim is to compare the name of *umlElement* with a set of reserved words.

```
rule uncorrectNameReservedWord {
 from
  element: MMuml!NamedElement(
   element.isReservedWord()
  )
  to
   problem: MMproblem!Problem(
    severity <- #error,
    description <- 'Invalid identifier',
    location <- 'root::'+element.getQualifiedName()
   )
}
```

Figure 3: ATL rule for reserved words.

3.3 Results

The *SysML2Problem* transformation is implemented as an Eclipse plug-in for the Topcased modeler (TOPCASED,). It includes 31 rules with 24 helpers, and provides a way to check that, for instance, identifiers are well formed, properties are typed or constants are always initialized. This kind of approach is well suited to check SysML models consistency in order to generate correct code. Indeed, it provides a way to write rules that automatically browse and check the SysML elements that can potentially bring errors in the generated VHDL-AMS code. Moreover, different levels of severity help the system engineer to adopt a specific methodology for modeling.

4 VALIDATION USING EUNIT

This section is about the validation of the *SysML2VHDLAMS* and *SysML2Problem* transformation with the framework EUnit. It also introduces a tool chain, which provides a transformation process applying a Test-Driven Development (TDD) strategy. The TDD approach advocates writing unit tests of a function before writing its code. TDD, and more generally unit testing, ensures that what is developed remains functional, and allows a very early detection of bugs in the development cycle.

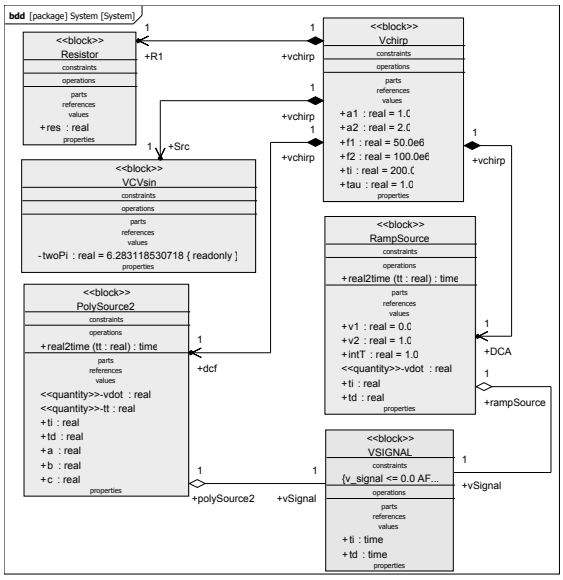

Figure 4: BDD of the chirp signal generator.

4.1 SysML Model to VHDL-AMS Code

This subsection explains the transformation of SysML model into VHDL-AMS code using MDE technologies. The SysML model typically uses the Block Definition Diagram (BDD), in the Internal Block Diagram (IBD) and in the parametric diagram. To illustrate this transformation, we introduce the model of a chirp signal generator, i.e. a system which generates a chirp signal. A chirp signal has frequency and/or amplitude that changes over time. The purpose of such a generator is to enable the activation of systems such as Inter Digital Transducer (IDT).

Figure 4 shows the BDD of this system. The block *Vchirp* represents the generator. It is linked by composition to the *Resistor*, *VCVsin*, *PolySource2* and *RampSource* blocks. The block *Resistor* receives the chirp signal in order to display it during the simulation. The block *VCVsin* generates a sinusoidal signal. The component *RampSource* enables the amplitude of the signal to change over time, whereas the component *PolySource2* enables the frequency to change over time. Finally, the block *VSignal* contains VHDL-AMS statements to initialize the simulation time.

The proposed MDE approach is based on two consecutive model transformations: the first one, called MMT (Model to Model Transformation), transforms SysML models into VHDL-AMS model representation using ATL, while the second one, called M2T (Model To Text), transforms the obtained VHDL-AMS models into VHDL-AMS source code using Acceleo. All the ATL and Acceleo transformation rules are available in (Gauthier et al., 2012).

As instance, Figure 5 shows the ATL transformation of SysML block into VHDL-AMS entity and architecture. A SysML block is transformed only if the precondition in the *from* section is true.

```
rule Block2Entity { (i)
 from
  block: MMsysml!Block(
   block.generalization->size() = 0 and
   block.notParentGeneralization() and
   not block.oclIsTypeOf(MMsysml!ConstraintBlock)
  )
 to
  entity: MMvhdlams!Entity(
   name <- block.name,
   owner <- block.owner
  ),
  architecture: MMvhdlams!Architecture(
   name <- 'behav',
   owner <- block.owner,
   entity <- entity
  )
}
```

Figure 5: ATL rule Block2Entity.

The ATL transformation result takes thus the form of a VHDL-AMS model, which is then translated into VHDL-AMS code by the Acceleo engine. Figure 6 shows the main concept of Acceleo with the dedicated keyword *template*.

```
[template generateEntity(du : Sequence(DesignUnit))]
 ENTITY [entity.name/] IS

 END ENTITY [entity.name/];
 [generateArchitecture(entity.architecture)/]
[/template]

[template generateArchitecture(archi : Architecture)]
 ARCHITECTURE [archi.name/] OF [archi.entity.name/] IS

 BEGIN
 END ARCHITECTURE [archi.name/];
[/template]
```

Figure 6: Acceleo templates.

4.2 Validation of Transformations

We now illustrate the validation of three kinds of ATL rules. The first kind of rules (Figure 5) only contains a helper: *notParentGeneralization()*, which verifies that the block is not a generalization of another block. If the precondition of the *from* section is true, the section *to* of the rule is automatically executed by the ATL engine. In other words, if the precondition is true, the section *to* is covered. To test this rule, it is necessary to cover its associated control flow graph given in Figure 7. Note that squared nodes of this graph represent boolean OCL expressions.

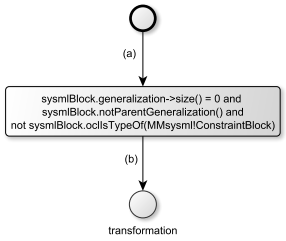

Figure 7: Control flow graph of the rule (i).

To create a test case, we create a SysML model (test input) and its expected transformation result (expected test result). This test case, which contains a block without generalization link, is therefore sufficient to cover the *Block2Entity* rule: path *ab* of the associated control flow graph.

The second category of rules contains *if* statements. For instance, the Figure 8 shows a rule with two *if* statements and the Figure 9 depicts its associated control flow graph. Note that diamonded nodes of this graph represent decision nodes.

```
rule Constraint2ConcurrentialStatement{ (ii)
 from sysmlConstraint: MMuml!Constraint(
   if (d) then ...
    if (g) then true
    else false
    endif
   else false
   endif
  )
 to
  vhdlamsCS: MMvhdlams!ConcurrentialStatement(
   [transformation]
  )
}
```

Figure 8: ATL rule with an "if" statement.

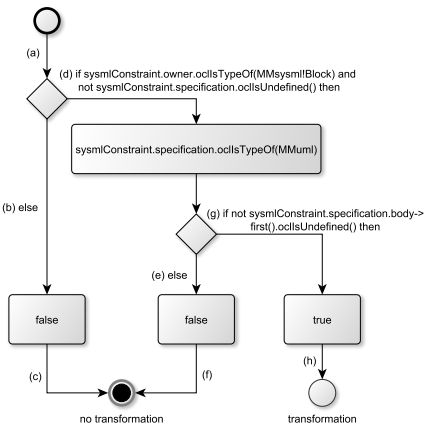

Figure 9: Control flow graph of the rule (ii).

According to (Myers et al., 2004), the decision coverage usually can satisfy statement coverage. In our case, this criterion is sufficient (even if it is rather weak) because no variable is modified during the execution of ATL rules. Thus, we have to write enough test cases to ensure that each condition in decision has a *true* and a *false* outcome, and that each statement is executed at least once. From this point of view, we assess that decision coverage can be met by three SysML models covering paths *abc*, *adef* and *adgh*.

The last rule, given in Figure 10, has an *or* statement. It comes from the *SysML2Problem* transformation and prohibits the use of *out* and *inout* direction. Its control flow graph is shown in Figure 11. In this case, the decision coverage can be met by two SysML models covering paths *abcd* and *abef*.

```
rule outInoutDirection{ (iii)
 from
  parameter: MMuml!Parameter(
   not parameter.uncorrectIdentifier() and
   (parameter.direction=#inout or
    parameter.direction=#out)
  )
 to
  problem: MMproblem!Problem(
   severity<-#error,
   description<-'Direction not supported',
   location<-'root::'+parameter.getQualifiedName()
  )
}
```

Figure 10: ATL rule with an "or" statement

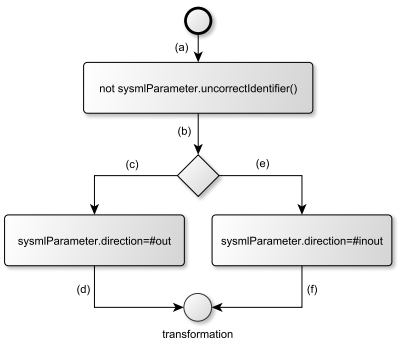

Figure 11: Control flow graph of the rule (iii).

4.3 Results

This section has presented a white-box testing method to validate ATL transformation from SysML to VHDL-AMS. We have created 41 unit tests (SysML models) to validate 34 ATL rules and 23 helpers of the *SysML2VHDLAMS* transformation. Concerning the *SysML2Problem* transformation, we wrote 41 unit tests which validate 31 rules and 24 helpers.

The use of decision coverage enables the complete coverage of our ATL rules both for *SysML2Problem* and *SysML2VDHLAMS* transformations. Moreover, unit tests are executed by an ANT file, which allows to validate transformations by running test cases in a continuous integration manner.

5 RELATED WORK

Further works have been studied in the model transformation testing domain, as introduced in (Jochen and Mohamed, 2006), by using white box approach for deriving test cases. Some of these techniques concern the meta-model coverage testing. For example, the tests generation tool of (Baudry et al., 2006) is based on the idea that unit tests should cover the source meta-model. However, this approach does not take care of preconditions that define constraints of transformations. But, our approach aims at performing structural testing on ATL rules, which implies that the test cases have to be essentially based on the preconditions of ATL rules.

Fleurey et al. (Fleurey et al., 2004) present an initial exploration of techniques for model transformation validation. They discuss two approaches to generate test data using systematic and bacteriologic algorithm. The systematic approach provides a way to build a model that covers as many items as possible of the effective meta-model. They compare this approach to the bacteriologic approach, which is an adaptation of genetic algorithms.

In (Erwan et al., 2006), the authors propose an algorithm and a prototype that build test models from the input meta-model and a set of model fragments. The set of model fragment is automatically deducted from the meta-model with a test criterion provided by the user. Constraints of transformation are not taken into account, which implies that test cases do not cover the transformation code with a white-box testing approach.

Sen et al. (Sagar et al., 2008) and (Sagar et al., 2009) have developed a tool called Cartier that generates hundreds of models conforming to the input domain and guided by different strategies. However, this tool is based on a black-box testing method that does not take the internal model transformation design requirements into account.

Finally, (Mottu et al., 2008) identify and discuss important issues that are relevant to define transformation testing oracles. Among them, we can underline the model comparison techniques with an oracle using an expected output model. This technique is very close to our approach since EUnit provides an engine that compares (using EMFCompare framework) expected data with the obtained transformation result.

6 CONCLUSION AND FUTURE WORKS

Verification of SysML consistency and validation of model transformation allow to improve the quality of critical systems design process. In this paper, we have presented an approach to verify the model consistency in regards of design rules. We have used an ATL transformation to generate problems from SysML models. Indeed, ATL provides an engine that automatically browses the source meta-model to execute the transformation process. Consequently, this technique is relevant to verify the modeling process in order to generate correct code, i.e. which does not contain any syntactic or semantic errors.

An approach to validate model-to-model transformation with EUnit has been presented. This approach is part of a larger work and provides a way to validate our transformation toolchain by using white-box testing method. The decision coverage criterion used and presented in this paper allows to ensure the exhaustive coverage of ATL rules. Moreover, unit tests could be automatically executed in a continuous integration manner and Test Driven Development context.

Further challenges have been identified in the domain of model transformation testing. The main limit of this approach is that the user has to define and write test cases and expected outputs. The automatic generation of test cases for model transformation has been discussed in several works and it would be possible to implement such a tool to generate test models with the same coverage criteria for ATL rules. For instance, the automatic generation of SysML models from the interpretation of header and *from* section of ATL rules.

Finally, the Acceleo model-to-text transformation has not been validated by unit tests, even if it has been validated with several experiments. It would be interesting to define a dedicated unit testing approach to validate Acceleo templates.

ACKNOWLEDGEMENTS

We would like to acknowledge the support of the Regional Council of Franche-Comté with the SyVad project (`http://syvad.univ-fcomte.fr/syvad/`) and the ANR-2011-BS-03-005 Smart Blocks project.

REFERENCES

Acceleo. Acceleo Documentation. `http://www.eclipse.org/acceleo/`, last viewed december 2012.

ATL. ATL Documentation. `http://www.eclipse.org/atl/`, last viewed december 2012.

Baudry, B., Dinh-trong, T., Mottu, J.-M., Simmonds, D., France, R., Ghosh, S., Fleurey, F., and Traon, Y. L. (2006). Model Transformation Testing Challenges. In *Proceedings of IMDT workshop*, Bilbao, Spain.

Christen, E. and Bakalar, K. (1999). VHDL-AMS-a hardware description language for analog and mixed-signal applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 46(10):1263–1272.

Erwan, B., Franck, F., Jim, S., Benoit, B., and Traon, Y. L. (2006). Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. In *Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE'06)*, pages 85–94, Washington, DC, USA. IEEE CS.

Fleurey, F., Steel, J., and Baudry, B. (2004). Validation in model-driven engineering: testing model transformations. In *Int. Workshop on Model, Design and Validation (MODEVA'04)*, pages 29–40, Saint-Malo, France.

Friedenthal, S., Moore, A., and Steiner, R. (2009). *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann. ISBN 9780123743794.

Garcia-Domnguez, A., Kolovos, D., Dimitrios, S., Rose, L., Paige, R., and Medina-Bulo, I. (2011). EUnit: a unit testing framework for model management tasks. In *Proceedings of the 14th Int. Conf. on MOdel-Driven Engineering Languages and Systems (MODELS'11)*, pages 395–409, Berlin, Germany. Springer-Verlag.

Gauthier, J.-M., Bouquet, F., Hammad, A., and Peureux, F. (2012). Transformation of SysML structure diagrams to VHDL-AMS. In *IEEE Workshop on design, control and software implementation for distributed MEMS (dMEMS'12)*, Besançon, France. IEEE CPS.

Jochen, K. and Mohamed, A.-E.-R. (2006). Validation of Model Transformations - First Experiences Using a White Box Approach. In *Int. Workshop on Model, Design and Validation (MODEVA'06)*, pages 62–77, Genova, Italy.

Mottu, J.-M., Baudry, B., and Traon, Y. L. (2008). Model transformation testing: oracle issue. In *Int. Workshop on Model, Design and Validation (MODEVA'08)*, Lillehammer, Norway.

Myers, G., Sandler, C., Badgett, T., and Thomas, T. (2004). *The Art of Software Testing*. Wiley, 2nd edition. ISBN 978-0-4714-6912-4.

Rumbaugh, J., Jacobson, I., and Booch, G. (2004). *The Unified Modeling Language Reference Manual*. Addison-Wesley, 2nd edition. ISBN 0321245628.

Sagar, S., Baudry, B., and Mottu, J.-M. (2008). On Combining Multi-formalism Knowledge to Select Models for Model Transformation Testing. In *International Conference on Software Testing, Verification, and Validation (ICST'08)*, Lillehammer, Norway.

Sagar, S., Baudry, B., and Mottu, J.-M. (2009). Automatic Model Generation Strategies for Model Transformation Testing. In *Theory and Practice of Model Transformations*, volume 5563 of *LNCS*, pages 148–164. Springer.

TOPCASED. Topcased: Toolkit in OPen-source for Critical Application and SystEms Development. `http://www.topcased.org`, last viewed december 2012.