HAL
open science

Teaching MDE through the Formal Verification of Process Models

Benoit Combemale, Xavier Crégut, Arnaud Dieumegard, Marc Pantel, Faiez Zalila

To cite this version:

Benoit Combemale, Xavier Crégut, Arnaud Dieumegard, Marc Pantel, Faiez Zalila. Teaching MDE through the Formal Verification of Process Models. 7th Educators' Symposium @ MODELS 2011: Software Modeling in Education (EduSymp2011), Marion Brandsteidl; Andreas Winter, Oct 2011, Wellington, New Zealand. hal-00646426

HAL Id: hal-00646426

https://inria.hal.science/hal-00646426v1

Submitted on 29 Nov 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Electronic Communications of the EASST
Volume X (2011)

7th Educators' Symposium @ MODELS 2011: Software Modeling in Education (EduSymp2011)

Teaching MDE through the Formal Verification of Process Models

Benoit Combemale, Xavier Crégut, Arnaud Dieumegard, Marc Pantel and Faiez Zalila

10 pages

Guest Editors: Marion Brandsteidl, Andreas Winter
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Teaching MDE through the Formal Verification of Process Models

Benoit Combemale[2], Xavier Crégut[1], Arnaud Dieumegard[1], Marc Pantel[1] and Faiez Zalila[1]

[1] Firstname.Lastname@enseeiht.fr
Université de Toulouse, IRIT – France

[2] Firstname.Lastname@irisa.fr
Université de Rennes 1, IRISA – France

Abstract: Model Driven Engineering (MDE) and formal methods (FM) play a key role in the development of Safety Critical Systems (SCS). They promote user oriented abstraction and formal specification using Domain Specific Modeling Languages (DSML), early Validation and formal Verification (V&V) using efficient dedicated technologies and Automatic Code and Documentation Generation. Their combined use allow to improve system qualities and reduce development costs. However, in most computer science curriculae, both domains are usually taught independently. MDE is associated to practical software engineering and FM to theoretical computer science. This contribution relates a course about MDE for SCS development that bridges the gap between these domains. It describes the content of the course and provides the lessons learned from its teaching. It focuses on early formal verification using model checking of a DSML for development process modeling. MDE technologies are illustrated both on language engineering for CASE tool development and on development process modeling. The case study also highlights the unification power of MDE as it does not target traditional executable software.

Keywords: Modeling language engineering, Formal verification, Metamodeling, Concrete syntax specification, M2M and M2T Model transformations

1 Introduction

The course presented in this contribution was designed in 2007 for M2 students in System and Software Engineering in Toulouse where Aeronautics, Automotive and Space transportations, and especially the development of Safety Critical Systems, are the key industries. These industries have been using model based design for the last 20 years and are experimented the use of formal specification and verification since 10 years. However, in the academic curriculae, these technologies are taught independently in very different manners as Model Driven Engineering (MDE) is associated to practical software engineering and Formal Methods (FM) to theoretical computer science. The growing complexity of critical systems can only be mastered by using both MDE techniques with user level languages and formal methods through tool level languages. This course gathers these elements to bridge the gap between domain-specific languages promoted by MDE and verification-specific languages promoted by formal methods. The usual verification done in MDE is made of structural OCL constraints (as in [BKSW09],[GP10]). This

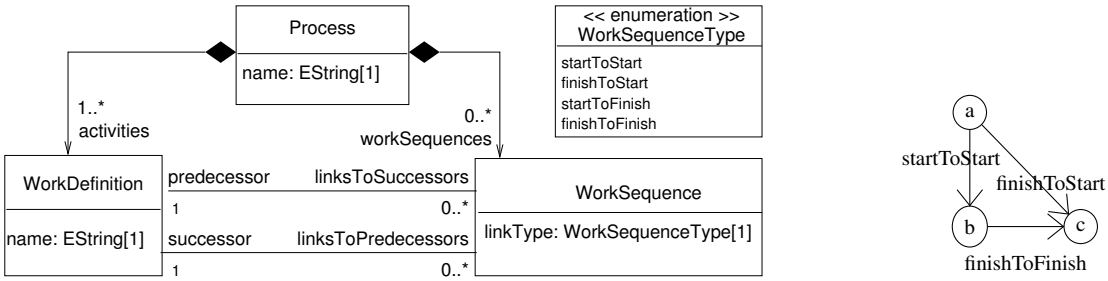

Figure 1: First metamodel of SimplePDL (left) and one conforming model (right)

course relies on model checking[1] based on Temporal Petri nets to verify behavioral correctness of development process models. Furthermore, the unification granted by the MDE is emphasized in this case study through the translation from end-user domain-specific languages to formal verification-specific languages. This course also explains how software modeling and formal verification can coexist during a system development and gives a practical overview of technologies easing the introduction of formal verification technologies such as model-checking. The key principle was to apply the various MDE technologies allowing to implement a Domain Specific CASE tool such as TOPCASED[2] to a verification driven case study.

This contribution is structured as follows. Section 2 presents the simple process modeling language use case that structures the whole course. Section 3 lists the main topics addressed by the course and further described in the next sections: metamodeling and static semantics constraints (section 4), concrete syntaxes (section 5) and transformations (section 6). Section 7 presents some extensions to the initial use case that allows student to develop and validate the acquired knowledge. Finally, section 8 explains how the course is conducted in the different contexts and gives some insights on future evolutions.

2 Formal Verification of Processes: *SimplePDL to Tina* Case Study

The main target of this course is the development of safety critical systems using Domain Specific Modeling Languages (DSML) and formal verification technologies. A simple yet realistic concrete case study is used all over the course in order to illustrate the different concepts and associated tools of MDE that are used to create DSMLs and to connect them to existing tools. The starting point is a very simple SPEM-based process modeling language called SimplePDL and the verification that all activities terminates.

The SIMPLEPDL metamodel (Figure 1) defines the process concept (*Process*) composed of a set of activities (*WorkDefinition*) representing the activities to be performed during the development. The concept of *WorkSequence* illustrates temporal dependencies between work definitions: the target activity can only be started or finished if the source activity is already started or finished. The kind of constraint (*linkType*) is expressed using the enumeration *WorkSequenceType*. For example, the *c* activity (right of figure 1) can only be started when *a* is finished and finished when *b* is finished. This first metamodel is obviously oversimplified but some extensions are proposed in section 7 to gain in expressiveness and validate the acquired knowledge.

[1] The authors wish to thank F. Vernadat that gives that part of the course and initiated the whole activity.
[2] http://www.topcased.org

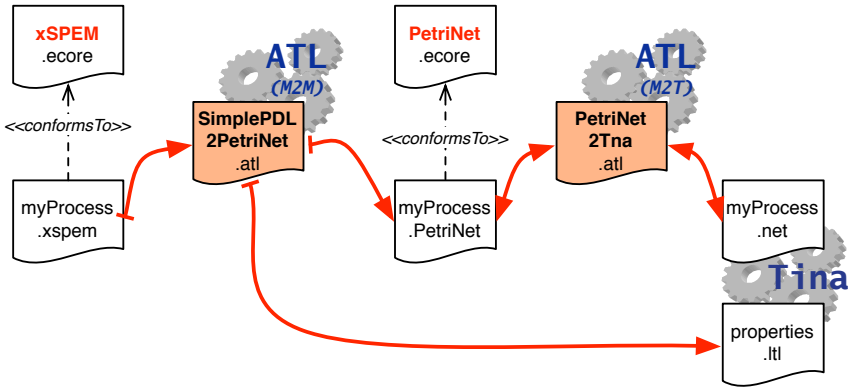

Figure 2: Approach to evaluate behavioral properties on a process model

The end user purpose is to check properties on a SimplePDL model. For example, he may asks whether the modeled process can finish (all activities from the process have been finished while respecting the sequencing constraints expressed by the work sequences).

To answer those questions, the students must reuse model-checking tools that they have studied in previous courses. We rely currently on the Tina toolkit[3] for the verification activities using Temporal Petri nets as modeling language and SE-LTL (*State Event Linear Temporal Logic*) as property language. The principle of the verification is thus to translate a process model into a behaviourally equivalent Petri net and then to express the common end users questions as LTL formulae that will be checked by the Tina model-checker. A PetriNet metamodel is used to avoid to be directly wired to the Tina input syntax. Thus, as depicted in Figure 2, the overall approach is composed of a model to model (M2M) transformation (SimplePDL to PetriNet) and two model to text (M2T) transformations (PetriNet model to Tina concrete syntax for the first one and SimplePDL model to LTL concrete syntax for the second).

3 Content and Schedule of the Course

The course presents the main concepts and tools of MDE as a set of 7 topics[4] grouped into 3 themes (metamodeling, concrete syntaxes and model transformations) and ends with a project as listed in the following table and developed in the next sections.

Metamodeling (section 4)	1. Metamodeling using Eclipse/EMF
	2. Static semantics using OCL
Concrete Syntaxes (section 5)	3. Textual concrete syntaxes using Xtext
	4. Graphical concrete syntaxes using GMF generators
Model transformations (section 6)	5. Model to Model transformation using ATL
	6. Model to Text transformation using xPand
	7. Model to Text transformation using ATL
Project (section 7)	Implementing extensions of the case study

One lecture (≈2h) and a practical work session (≈2h) is allocated to each topic. Lectures

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

[4] An optional topic called "The Tina toolkit" is a 20 minutes tutorial that explains to students who have never used the Tina toolkit how to perform model-checking. It is not described here as it is not strictly related to MDE technologies.

present the main concepts and associated standards. For each topic[5], except the two last one, the practical session has the same three-part structure. The first part is a tutorial that presents the concepts and tools illustrated on the SimplePDL language. The second part provides exercises to ensure that the students got a good understanding. Exercises consists in completing the work done in the first part. The third part is an open exercise that asks the student to do a similar work on the Petri net language, without any guidance. The first two parts are usually finished in about two hours of supervised work but the last one is often only started in the supervised time slot and students have to finish it on their own.

Finally, a project is done by students: the implementation of several extensions to the main case study. Section 7 describes the extensions. First, a 6 hours assignment is used to assess that students master the MDE concepts and tools. It is based on the E_1 and E_2 extensions of section 7. An oral presentation and demonstration is organized. Then, the other extensions are implemented by the students (18 hours of personal work). These extensions can be done independently in order to avoid unnecessary difficulties in regard to the evaluation of their understanding. Students pass an oral examination where they show the result of their work and explain how they conducted this work, what were their main choices and why they made them.

All tools used in practical work are based on the open-source Eclipse platform and are part of (or can be added in) the Eclipse Modeling Tools[6]. This choice allows students to go on with their work using any computer, and have a unified environment for all the tasks they have to do.

4 Metamodeling

The first topic focuses on metamodeling using the Ecore language from the Eclipse Modeling Framework[7]. Students use the Ecore graphical editor to load the initial metamodel of SimplePDL (Figure 1). They use EMF to generate Java classes to load and store models, and a structured editor that is then used to edit small process models. The structured editor provides a good understanding of the *containment* attribute of an *EReference* (and a concrete example of the composition relationship already seen in UML classes that took place in the previous years). Furthermore, the *eOpposite* attribute also helps in the understanding of the UML association (when one reference is updated, the opposite reference is also updated).

The second part of this topic consists in modifying the metamodel by adding a *ProcessElement* metaclass to generalize WorkDefinition and WorkSequence elements and a Guidance element that allows to associate a natural language description to any process element. Aside manipulating the Ecore editor, students discusses advantages and drawbacks of their metamodels.

Finally, students have to define a metamodel for Petri nets from scratch. The starting point is a textual description of Petri nets (taken from wikipedia) with some model examples. Examples are easier to understand by the students but they lead them to define a partial metamodel that has to be completed using the information provided in the textual description. It is a very interesting exercise because several metamodels are usually provided by the various students and many exchanges take place in order to define the best one according to various criteria.

[5] Teaching materials are available at http://cregut.perso.enseeiht.fr/2010/ but, unfortunately, currently only in French.
[6] cf. http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/indigor
[7] See the Eclipse Modeling Project: http://www.eclipse.org/modeling

```
process dvp {
    wd a
    wd b
    wd c
    ws s2s from a to b
    ws f2f from b to c
    ws f2s from a to c
}
```

```
process dvp {
    wd a
    wd b starts if a started
    wd c finishes if b finished
      starts if a started
}
```

Figure 3: Two possible textual concrete syntaxes for SimplePDL

Furthermore, the Petri net metamodel does not capture all the constraints of Petri net models. It thus demonstrates the need to define the static semantics. We use OCL for that purpose.

OCL and static semantics is presented in the next topic. The TOPCASED OCL checker is used for practical activities. When an invariant does not hold, the context element is red marked. First, students must write some OCL constraints in order to understand the basics of this language. The use of the tools allows to stress that it is important to define an invariant at the right place. For example, expressing that activity names are unique may be expressed as an invariant on *Process* but marking a *Process* element as wrong is of little help to find the bad activities. Thus, this invariant is better expressed on *WorkDefinition* elements so that bad activities are highlighted.

Thanks to the Petri net metamodel, students can notice that there is a balance to be found between having a simpler metamodel with less concepts and relationships but more OCL constraints, or a more complex metamodel with less OCL constraints.

5 Concrete syntax

A metamodel only describes an abstract syntax. Models are stored as XML or XMI files that could be considered as concrete syntaxes but these are not designed as editing tools for the end user. Thus, it is mandatory to provide concrete syntaxes, like the Ecore diagram defines a graphical concrete syntax for Ecore models. In this course, textual concrete syntaxes are presented using Xtext and graphical ones using GMF Tooling.

5.1 Textual Concrete Syntaxes

Xtext[8] is used as a tool to define concrete syntaxes. Figure 3 proposes two examples of concrete syntaxes for the SIMPLEPDL model presented on the right of figure 1. The first one is used to explain the concepts of Xtext: both the concrete syntax and the Xtext files are provided. Then, the Xtext file corresponding to the second syntax is given and students have to find the concrete syntax it corresponds to. They first have to write it on a paper to ensure they understood the grammar as defined in Xtext, then they can test it using the generated Eclipse editor. Finally, they have to define their own textual concrete syntax for Petri nets.

Xtext can generate the metamodel that is closely related to the structure of the grammar provided in Xtext for the concrete syntax. The metamodel corresponding to the first syntax is close to the one of Figure 1 whereas the second one is very different. Students are thus shown that a compromise must be found between the structure of the grammar and the structure of the generated metamodel. Using an existing metamodel is also possible but is a bit tricky.

[8] http://www.eclipse.org/Xtext

The main lesson is that the "natural" metamodel defined for a domain is generally not well-suited for defining the concrete syntax. It is thus better to let Xtext generate its own metamodel and then use a M2M transformation to translate this one into the other. Xtext allows to automatically use an Xtend transformation in that purpose but we do not teach that feature to students.

5.2 Graphical Concrete Syntaxes

Graphical concrete syntaxes allows to built a graphical editor for a given metamodel. It is very attractive for the students to be able to do so in a very short time.

From a pedagogical point of view, graphical editors are a good illustration of MDE principles, and generative approaches. The user only has to describe in a declarative manner the features of the expected editors, and the different generators get the things done. Moreover, while most of the generative approaches are defined for a given DSML to automate tasks from the conforming models, the use of such generative approaches can highlight the usefulness of a metametamodel.

We initially used the simple graphical editor generator provided by TOPCASED. All aspects required to describe the editor were blurred in the same configuration model (Views, Controler and mappings to the Model in the MVC pattern) with very few possible customizations.

For three years now, we have switched to GMF Tooling[9] that is part of the Eclipse Modeling bundle. GMF Tooling puts a better stress on separation of concerns as it is based on different models that describe each parts of a graphical editor: tools, graphical representation of elements, palette and mappings among these models and the Ecore metamodel. It also uses OCL to define some behavior, for example to prevent a reflexive transition (illustrated on the *WorkSequence* element). Unfortunately GMF tooling has suffered from nasty bugs for years that generate files with obvious mistakes that have to be hand-corrected by students, which is very confusing.

The main drawback of these tools is that they are quite complex and, often students only perform the required actions without reading the explanations and understanding the underlying motivations. Thus, at first sight, students consider those tools as tedious. Furthermore, they lack many features and does not help when they want to build their own editor for PetriNet models without writing Java code. It is why we plan to switch to OBEO Designer[10] to have a more robust and sophisticated tool.

6 Model Transformation

The last topics concern model transformation. Model to model transformations (M2M) are presented using ATL. ATL and xPand are used for model to text transformations (M2T).

6.1 Model to Model Transformations

M2M transformations are a key concept of the MDE course. A presentation of QVT [OMG08] specification puts the focus on the operational and declarative parts. Many languages are available. We could have used operational languages such as Kermeta, xTend, SmartQVT, EMP

[9] http://wiki.eclipse.org/GMF
[10] http://www.obeodesigner.com

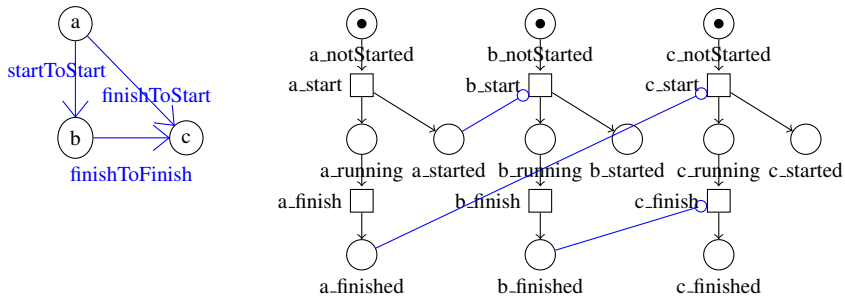

Figure 4: Translation of a process composed of work definitions *a*, *b* and *c* into a Petri net

Operational QVT, Epsilon Transformation Language, etc. But we feared that students would only consider them as classical programming languages. Indeed, they have to manage explicitly the control flow, track links to already created target elements, etc. Their major benefit over a language like Java is their powerful query language and collections operators derived from OCL.

We have chosen ATL and we mainly use its declarative part so that students can concentrate on how to map source elements to target elements. In practice, they write a rule for each source element and describe the elements that have to be created in the target model.

Before handling technical details, we first ask students to find how to translate a SimplePDL model into a Petri net in order to verify that the modeled process terminates. We take a very simple process like the one on the left of figure 4. The corresponding Petri net is on the right of this figure. The purpose is that they gain a good understanding of the mapping between process models and Petri nets. Each *WorkDefinition* is translated into three places characterizing its state (*notStarted*, *running* and *finished*) linked by two transitions. These transitions model the actions that we want to observe on an activity: one can *start* an activity and then *finish* it. An activity is considered as *started* if it is running or finished. This is recorded by the place called *started*. A *WorkSequence* becomes a *read-arc*[11] from one place of the source activity (either *started* or *finished*) to a transition of the target activity (either *start* or *finished*) according to the kind of *WorkSequence*. This allows to illustrate a property driven approach to formal model design.

To show the principles of ATL, we give the rule that translates a Process into a PetriNet and the beginning of the rule that translates a WorkDefinition into nodes, places and arcs. Process to PetriNet is a 1 to 1 rule. Students have to complete the WorkDefintion2PetriNet rule. This rule creates 4 places, 2 transitions and 5 arcs for each work definition (1 to n rule). The only difficulty is that a process element does not have direct access to its process container. An ATL helper is provided to access the missing reference.

Finally, they add a new rule to translate one work sequence into a PetriNet read arc (1 to 1 rule). This last rule illustrates the use of `resolveTemp` ATL operator (`resolveIn` in QVT) which allows to select one element of the target model among those built from a given source model element (in a 1 to n rule).

Operational constructs or more advanced concepts like rule inheritance, called rules and so on are mentioned but not used. One important aspect of the use case is that it allows this kind of simple declarative transformation. It would be useful to illustrate also an example that is really ill fitted and requires a more imperative transformation, however we lack time to have the students do the experiment and can only give insights on this problem.

[1] 1 A read-arc checks that enough tokens are in the input place. Tokens are not withdrawn when the transition fires.

6.2 Model to Text Transformations

A first example shows how xPand may be used to generate the graphical concrete syntax of SimplePDL proposed in Fig. 4. Then, students have to generate a DOT file for textual graph representation. The final exercise consists in generating the Tina text from a Petri net model.

Students are quite familiar with template languages due to the use of JSP in previous courses and have no problem to use xPand despite its verbose syntax and the use of non common characters (even for french students).

ATL also provides M2T transformations. We provide the students with the ATL query that translates a Petri net model into the Tina syntax. All aspects of Petri net are handled except for time constraints. So, once students have noticed that ATL allows to define helpers on metamodel elements and uses a query language very close to OCL, they complete the query. They are then able to write from scratch the second M2T transformation that generates the LTL formulae corresponding to the questions asked on the process (they obviously depends on the process model). The ATL main drawback is that mistakes are generally only detected at runtime.

7 Case Study Extensions

The initial case study allows to introduce MDE concepts and tools and to check that students got a basic understanding of the technologies. Several extensions are proposed to students that they have to develop on their own in order to improve their practice. These extensions are listed hereafter with a short description of their strong points. For each extension, students have to extend the core course realizations: extend the SimplePDL metamodel, add new OCL constraints, update concrete syntaxes, M2M transformation to handle extensions and, eventually the generation of LTL formulae. This work is done in autonomy and is eventually assessed. The stress is also put on how they can validate their work.

E_1: *Resources.* Every work definition requires a set of resources to be executed.

E_1 is quite easy to implement. SimplePDL metamodel is completed with two new concepts *Resource* and *Allocation* to describe how many occurrences of a resource are needed by an activity. OCL constraints specifies restrictions on the amount of allocated resources with respect to the total amount of resources. Extending concrete syntaxes is easy. The M2M transformation is extended with two rules. The first creates one place for each *Resource*, the number of token is the amount of available resources. The second adds two arcs so that an activity takes the resources when it starts and releases them when it finishes.

E_2: *Time constrained.* Are activities able to finish in a defined time interval?

The idea is to add an observer on the activities to record if they finish too early, in time or too late. The observer allows to answer both the initial question and the new one: may the process finish? May it finish while respecting time constraints?

E_3: *Hierarchical work definitions.* A work definition can be split into sub work definitions.

It is mainly a modeling problem. Students have to find how to model hierarchical activities. It also allows to illustrate the composite design pattern. On the ATL side, rule inheritance is useful to avoid code redundancy.

E_4: *Suspending a work definition.* A work definition may be suspended and its resources released. It can then be resumed if the required resources are available at that time.

This extension is not difficult if we consider that time is not stopped when an activity is suspended. The main interest is that the SimplePDL metamodel is unchanged. Only its semantics (encoded by the translation from process models to Petri nets) changes as an activity may be suspended that leads to new possible scenarios to ensure the process finishes. Only the M2M transformation is concerned by this extension.

E_5: *Different possible sets of resources.* A work definition can work with different sets of resources (alternative resources).

The difficulty is to find how to represent the "or" between resource sets in Petri nets.

8 Discussions

Modularity of the course Even if the purpose of the course was to present most of the MDE concepts and tools, it is very modular and parts of it can be dropped of. In an alternate version, only one lecture presents the main MDE concepts, then each of the 7 topic (section 3) is done using 2 hours sessions. The first part of each topic is used to explain the concepts and see how tools work. Students are asked to finish topics on their own. Finally, students have a 6 hours session to implement the two first extensions and have to write a short report. Another one is taught in only 8 hours. It consists in an overview of MDE at the end of the M2 level. There is only a 2 hours lecture to present the main concepts of MDE and 6 hours of practical sessions, 2 hours for metamodeling with SimplePDL and PetriNet (OCL is left as homework) and 4 hours for model transformation (focusing on M2M transformation). Concrete syntaxes have already been studied during M1.

Formal verification use case is not a difficulty Even if the course is based on a verification case study. We have experienced that the lack of knowledge in formal verification and model-checking is not an issue. Petri net are indeed an easy to understand formalism and students are able to define its metamodel and propose a translation of SimplePDL model into Petri net. Furthermore, it allows to present model-checking to students and put the focus on the importance of early verification in the development process.

Technical overload The practical work causes some troubles to students. In particular, when making the metamodel evolve, students often do not entail the consequences of their choices and need some help to validate their proposals. Furthermore, all the steps required to verify a model are run manually: running the M2M transformation, running the two M2T transformations to generate the Petri net file and the LTL formulae, running the Tina model-checker. Presenting tools to automate these steps (like Ant or a workflow engine) could favor the adoption. We are also surprised that students are not asking for such tools.

Studying verification does not imply the verification of the study We observed that students are not really concerned about the validity of their development, despite the fact they had to develop verification tools and thus are aware of that. A major problem is that students do not rigorously verify their work (OCL constraints, transformation...). Students only use the example

provided in the subject to validate their work whereas they should define many wrong models to ensure that all inconsistencies are caught by the OCL invariants, define several models to test the different aspects of their transformations, etc. In fact students are not really experienced at defining good test cases that provide a significant coverage.

Students' difficulties The students' difficulties we have seen in this course are twofold: one is the necessary ability to find the right level of abstraction, especially making the right choices in the design of language is essential to reduce the accidental complexity generated each time the metamodel changes. The other difficulty comes from the technical environment that may appear complicated to learn, especially the lack of maturity and performance of certain tools, and the proliferation of tools which continues to evolve. We noticed that students encounter in particular one of these difficulties according to their curriculums. While some students are comfortable with a complex technical environment but have difficulty with abstraction, other students easily manipulate abstractions but are struggling to implement them using the MDE tools.

9 Conclusion and Perspectives

In this paper, we have presented how a case study may help in presenting MDE concepts and tools in an attractive way that furthermore make students aware of formal verification technologies that are getting more and more interest in the safety critical transportation systems industry. Despite its verification focus that can be considered as difficult for most students, we have noticed that it can also been taught to student with no formal background because Petri net are an easy to understand language in its concepts and semantics.

For the future, we strongly believe that MDE course should be taught at M1 level because it is becoming a very important domain, not only in academy but also in industry. Our main fear about this is to know whether students will have enough background and experience on modeling to be able to handle metamodeling and generative tools.

We also think that MDE is quite close to Software Language Engineering (abstract syntax, concrete syntax, transformation of abstract syntax, etc) and thus it could be interesting to merge these modules to gain on efficiency and stress the important points and favor separation of concerns: abstract syntax, concrete syntax, transformation. We now plan to rely on MDE technologies in our compiler course instead of classical attribute grammar technologies.

Bibliography

[BKSW09] P. Brosch, G. Kappel, M. Seidl, M. Wimmer. Teaching Model Engineering in the Large. 2009. In: Educators' Symposium @ Models 2009.

[GP10] T. Gjosaeter, A. Prinz. Teaching Model Driven Language Handling. *ECEASST*, 2010. In: Educators' Symposium @ Models 2010.

[OMG08] Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Query/View/-Transformation (QVT) Specification, version 1.0. Apr. 2008.