



7th Educators' Symposium@MODELS 2011  
— Software Modeling in Education —

Pre-Proceedings



edited by

Marion Brandsteidl

TU Wien, Information & Software Engineering Group

Andreas Winter

Carl von Ossietzky Universität Oldenburg, Software Engineering

OLNSE Number 2/2011  
September 2011

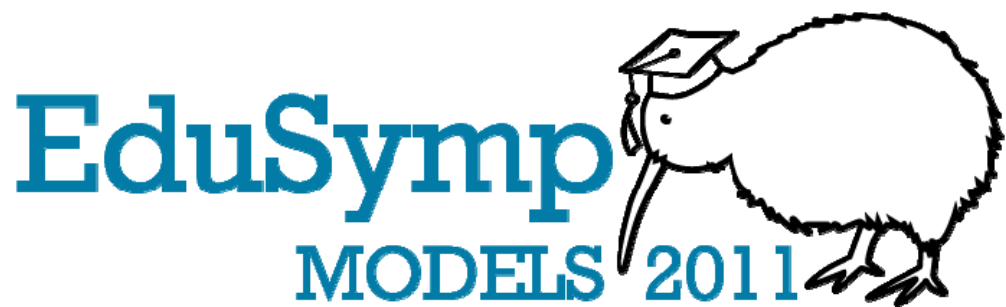
**Oldenburg Lecture Notes  
on Software Engineering (OLNSE)**  
Carl von Ossietzky University Oldenburg  
Department for Computer Science  
Software Engineering  
26111 Oldenburg, Germany

– copyright by authors –



<b>09:00 - 10:30</b>	<b>Session 1: Teaching Certain Topics</b>	<b>page</b>
09:00 - 09:05	Opening Remarks	
09:05 - 09:40	<i>Guest Speaker: Robert B. France: Teaching Student Programmers How to Model: Opportunities &amp; Challenges</i>	5
09:40 - 10:05	Avoiding OCL specification pitfalls Dan Chiorean, Ileana Ober and Vladiela Petrascu.	7
10:05 - 10:30	Teaching MDE through the Formal Verification of Process Models Benoît Combemale, Xavier Crégut, Arnaud Dieumegard, Marc Pantel and Faiez Zalila.	17
10:30 - 11:00	<b>Break</b>	
<b>11:00 - 12:30</b>	<b>Session 2: Practical Approaches in Teaching and Teaching in Practice</b>	
11:00 - 11:30	Mismatches between industry practice and teaching of model-driven software development Jon Whittle and John Hutchinson.	27
11:30 - 12:00	Ready for the Industry: A Practical Approach to Teaching MDE Gordana Milosavljevic, Igor Dejanovic and Branko Perisic.	31
12:00 - 12:30	Models and Clickers for Teaching Computer Science Matthias Hauswirth.	41
12:30 - 14:00	<b>Lunch</b>	
<b>14:00 - 15:30</b>	<b>Session 3: Teaching approaches und Modeling Skills</b>	
14:00 - 14:30	Model Correctness Patterns as an Educational Instrument Azzam Maraee, Mira Balaban, Arnon Strum and Adiel Ashrov.	45
14:30 - 15:00	Threshold Concepts in Object-Oriented Modelling Ven Yu Sien and David Weng Kwai Chong.	55
15:00 - 15:30	Teaching Modeling-An Initial Classification of Related Issues Ludwik Kuzniarz and Jürgen Börstler.	65
15:30 - 16:00	<b>Break</b>	
<b>16:00 - 17:30</b>	<b>Session 4: Discussion: Modeling Curriculum</b>	
16:00 - 16:20	Position Paper: Software Modelling Education Martina Seidl and Peter Clarke.	71
16:20 - 17:20	Discussion: Skills to teach in a Modeling Curriculum	
17:20 - 17:30	Wrap up and plans for the next symposium	





Collocated with the ACM/IEEE International Conference on Model-Driven Engineering Languages and Systems (MODELS), the Educators' Symposium (EduSymp) focuses on the wide topic of software modeling education ranging from experience reports and case studies to novel pedagogical approaches. MODELS 2011 will host the 7<sup>th</sup> Educators' Symposium, whose papers are compiled in these pre-proceedings.

Modeling systems plays an important role in today's software development and evolution. Modeling provides goal-oriented abstractions in all phases of software development, which requires deep knowledge on modeling techniques and broad experiences in applying these techniques. Software Engineering is supported by various modeling techniques, providing modeling languages, modeling language definition technologies, and model transformation technologies. Industry and academia successfully realized expressive modeling and meta-modeling languages and mature tools for the practical application.

The Educators' Symposium at MODELS focuses on discussing educating these technologies to software engineers at universities and software industries. Although most computer science curricula include some education in modeling technologies and therefore provide the basic building blocks for modeling, meta-modeling, and model transformation, the whole spectrum of modeling in software engineering is rarely captured, even a curriculum on modeling is not available to define education standards in modeling.

EduSymp 2011 received 13 papers, from which 8 were accepted for presentation, eventually. The papers have passed through a rigorous review process and will be presented at EduSymp 2011. We are very delighted that Robert France will present his broad insights in teaching modeling in an interesting keynote on "Teaching Student Programmers How to Model: Opportunities & Challenges". This year's symposium will also include an intensive discussion on skills and competencies to be educated in modern modeling education. These discussions will be introduced by a stimulating position paper by Martina Seidl Peter Clarke on Software Modelling Education.

The final proceedings of EduSymp 2011 will be published at the Electronic Communications of the EASST, which will contain reworked and extended versions of the papers presented in these pre-proceedings.

Thanks to all authors who considered EduSymp 2011 for sharing and discussing their thoughts and submitting a paper. Our deepest thanks also go to Robert France, Martina Seidl and Peter Clarke for supporting EduSymp with their additional presentations. We would also like to express our gratitude to the program committee who supported excellent and timely reviews, which will provide significant hints to improve and extend the already much elaborated submitted papers. The list of the International Program Committee is shown below:

- Colin Atkinson, University of Mannheim, Germany
- Jordi Cabot, University of Nantes, France
- Peter J. Clarke, Florida International University, USA
- Ira Diethelm, Carl von Ossietzky University, Germany
- Jean-Marie Favre, OneTree Technologies, Luxembourg
- Robert France, Colorado State University, USA
- Michael Godfrey, University of Waterloo, Canada
- Martin Gogolla, University of Bremen, Germany
- Øhystein Haugen, SINTEF, Norway
- Gerti Kappel, Vienna University of Technology, Austria
- Ludwik Kuzniarz, Blekinge Institute of Technology, Sweden
- Jochen Ludewig, University of Stuttgart, Germany
- Karl Reed, La Trobe University, Australia
- Jean-Paul Rigault, University of Nice, France
- Patricia Roberts, University of Brighton, UK
- Martina Seidl, Vienna University of Technology and Johannes Kepler University of Linz, Austria
- Ven Yu Sien, HELP University College, Malaysia

Our thanks also include the additional reviewers (Christina Dörge , Malte Dünnebieer Elena Planas, Lars Hamann, and Manuel Wimmer). Finally, we like to thank the organizers of MODELS 2011 in Wellington for providing brilliant support for organizing the 7<sup>th</sup> Educators' Symposium@Models.

October 2011

Marion Brandsteidl  
Andreas Winter

# Teaching Student Programmers How to Model: Opportunities & Challenges

Robert B. France<sup>1</sup>

<sup>1</sup> [france@cs.colostate.edu](mailto:france@cs.colostate.edu)  
Dept. of Computer Science  
Colorado State University

**Abstract:** In my experience, students with some programming expertise (or students who believe they are programming experts) tend to view software modeling with great skepticism. They often feel that modeling adds accidental complexity to the software development process as they perceive it.

While we should acknowledge that there may be some elements of truth in their views (new methods, tools and techniques do bring additional baggage that can initially contribute to accidental complexity), we, as educators, should also try to leverage such skepticism in an opportunistic manner. In this talk I'll present some thoughts on how we can leverage such skepticism and also discuss some of the challenges of teaching students how to discover and use "good" abstractions in their models.





# Avoiding OCL specification pitfalls

Dan Chiorean<sup>1</sup>, Ileana Ober<sup>2</sup>, Vladiela Petraşcu<sup>3</sup>

<sup>1</sup>[chiorean@cs.ubbcluj.ro](mailto:chiorean@cs.ubbcluj.ro), Babeş-Bolyai University, Cluj-Napoca, Romania

<sup>2</sup>[ober@irit.fr](mailto:ober@irit.fr), Université Paul Sabatier, Toulouse, France

<sup>3</sup>[vladi@cs.ubbcluj.ro](mailto:vladi@cs.ubbcluj.ro), Babeş-Bolyai University, Cluj-Napoca, Romania

**Abstract:** This paper discusses about teaching software modeling by using OCL specifications, in the context in which the web represents the main source of information. The raise of the interest for models induced a higher need for clear and complete specifications. In case of models specified by means of MOF based languages, adding OCL constraints proved to be an interesting answer to this need. Several OCL examples posted on web include hasty specifications, that are often dissuasive with respect to complementing models with OCL specification. OCL beginners, and not only, need to know how to avoid potential specification traps.

Our proposal is based on a clear, unambiguous and complete description of requirements, that represents the first step towards good OCL specifications. The work highlights several major aspects that need to be understood and complied with to produce meaningful and efficient OCL specifications. This approach was tested while teaching OCL at Babes-Bolyai University of Cluj.

**Keywords:** rigorous modeling, OCL specifications, meaningful specifications, efficient specifications, model understanding

## 1 Introduction

OCL is a language whose spread has not confirmed the optimistic expectations expressed since its inclusion as part of UML 1.1, and then as part of all OMG MOF-based modeling languages. Being much more active in promoting the language compared to its industrial counterpart, the academic community has published several reviews in this respect, identifying among the causes of this state of facts the ambiguities and gaps from the language specification, as well as the immaturity of OCL tools, as opposed to the now classical IDEs (Integrated Development Environments). Although there has been progress in the above mentioned fields, the developers' feedback is far from satisfactory. One possible reason is given by both the lack of illustrative examples for the advantages of using OCL, and the availability of a large number of examples which, at best, cause confusion among readers. An experience of over ten years in teaching OCL to computer science students (at both bachelor and master levels) has allowed us to conclude that, besides providing positive recommendations (articles, books, etc.), it is mandatory to warn potential OCL users (students, in this case) on the pitfalls enclosed by negative examples. As web users, students are exposed to both clear, well-written documents and to documents containing pitfalls, on whose potential occurrence teachers have the duty of raising warnings. However, merely showing that particular models or specifications are inadequate or even incorrect with re-

spect to the purpose they were created for is not enough. Presenting at least one correct solution and arguing on its advantages is a must.

Complementing models with OCL is meant at eliminating specifications ambiguities, increasing rigor, reaching a full and clear definition of query operations, as well as promoting design by contract through the specification of pre and post-conditions.

Development of models and applications takes place as an iterative incremental process, which allows developers to return to earlier stages whenever the case. Enhancing models with OCL specifications facilitates their deeper understanding, through both rigor and extra detail. Whenever the results of evaluating OCL specifications suggest a model change, this change should only be done if the new version is more advantageous compared to the previous ones, as a whole. The use of OCL specifications should contribute to the requirements validation. An application is considered as finished only when there is full compliance among its requirements, its model, and itself.

The remaining of this paper is organized as follows. Section 2 explains the reasons why teaching OCL through examples integrated in models is more advantageous compared to the classical way of teaching OCL. In Section 3, we argue on the necessity of understanding the model's semantics, which is the first prerequisite for reaching a good specification. Section 4 emphasizes the fact that we need to consider several modeling solutions to a problem and choose the most advantageous one with respect to the aspects under consideration. Section 5 shows the role of OCL in specifying the various model uses, while Section 6 justifies through an example the need of using snapshots for validating specifications. The paper ends with conclusions.

## **2 Teaching OCL Through Examples Integrated in Models**

The teaching of OCL can be achieved in various ways. The classical approach emphasizes the main language features: its declarative nature and first order logic roots, the type system, the management of undefined values, the collection types together with their operations and syntax specificities, and so on [CD10], [por]. Many examples used for collections employ expressions with literals, which are context-independent and easy to understand.

OCL is a textual language which complements MOF-based modeling languages. The students' interest in understanding and using the language increases if there are convinced with respect to the advantages earned from enriching models with OCL specifications. To convince students on the usefulness of using OCL, the chosen examples should be suggestive in terms of models and enlightening in terms of earned benefits. That is why we have considered more appropriate taking an "inverted curriculum"-type of approach, by introducing OCL through examples in which the specifications are naturally included in the models. Unfortunately, along with positive OCL specification examples, the existing literature also offers plenty of negative ones, starting with the WFRs (well-formedness rules) that define the static semantics of modeling languages. The negative examples may wrongly influence students' perception. Therefore, we argue that a major issue in teaching OCL to students is explaining them the basic principles that should be obeyed when designing OCL specifications, principles that should help them avoid potential pitfalls.

An example that has been probably meant to argue for the use and usefulness of OCL (taking

into account the title of the paper in question) is the one enclosed by the reference [Tod11]. The examples and solutions proposed by this article provide an excellent framework for highlighting important aspects that should be taken into account within the modeling process. In the second semester of the 2010-2011 academic year, we have used these examples in order to warn students on the pitfalls that should be avoided when enriching models with OCL specifications.

### 3 Understanding the Model's Semantics

A model is an abstract description of a problem from a particular viewpoint, given by its intended usage. The design model represents one of the possible solutions to the requirements of the problem to solve. It is therefore essential for the students to realize the necessity of choosing a suitable solution with respect to the aspects under consideration. The first prerequisite for designing such a model is a full understanding of the problem at hand, reflected in a thorough informal requirements specification. Nygaard's statement "Programming is Understanding" [Ven04] is to be understood as "Modeling is Understanding", since "Object-oriented development promotes the view that programming is modeling" [Nie11]. Understanding is generally acquired through an iterative and incremental process, in which OCL specifications play a major role. That is because, "if you don't understand something, you can't code it, and you gain understanding trying to code it." [Ven04].

The modeling example from [Tod11], mentioned in the previous section, describes parents-children relationships in a community of persons. However, the model requirements description is incomplete with respect to both its intended functionalities and its contained information. In such cases, the model specification, both the graphical and the complementary textual one (through Additional Operations - AOs, invariants, pre and post-conditions), should contribute to enriching the requirements description. The process is iterative and incremental, marked by repeated discussions among clients and developers, until the convergence of views from both parties.

The proposed solution should allow a correct management of information related to persons, even when this information is incomplete. Unknown ancestors of a particular person is such a case (sometimes not even the natural parents are known). For such cases, the model provided in [Tod11] and reproduced in Figure 1 is inadequate, due to the infinite recursion induced by the self-association requiring each person to have valid references towards both parents. Snapshots containing persons with at least one parent reference missing will be thus qualified as invalid.

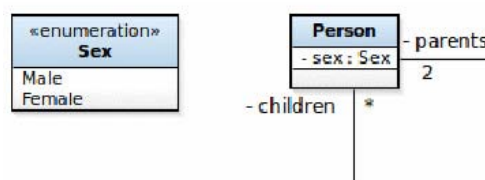


Figure 1: Genealogical tree model [Tod11]

Both this problem and its solution, consisting in relaxing the `parents` multiplicity to `0..2`,

are now “classical” [Cab11]. Partial or total lack of references (1 or 0 multiplicity) indicates that either one or both parents are unknown at that time.

The only constraint imposed in [Tod11] on the above mentioned model requires the parents of a person to be of different sexes. Following, there is its OCL specification, as given in [Tod11].

---

```
self.parents->asSequence()->at(1).sex <> self.parents->asSequence()->at(2).sex
```

---

Although apparently correct, this expression encloses a few pitfalls:

1. In case there are valid references to both parents, but the sex of one of them is not specified, the value of the corresponding subexpression is undefined and the whole expression reduces to either `undefined <> Sex::Male` or `undefined <> Sex::Female`. This later expressions provide tool-dependent evaluation results (`true` in case of USE [use] and `undefined` in case of OCLE [LCI]). The results produced by OCLE comply with the latest OCL 2.3 specification [OMG11]. However, as the topic of evaluating undefined values has not yet reached a common agreement, students should be warned on this.
2. In case at least one parent reference is missing and the multiplicity is 2, the evaluation of WFRs should signal the lack of conformance among the multiplicity of links between instances and the multiplicity of their corresponding association. To be meaningful, the evaluation of model-level constraints should only be performed in case the model satisfies all WFRs. Unfortunately, such model compilability checks are not current practice. In case the `parents` multiplicity is 0..2, the model will comply with the WFRs, but the constraint evaluation will end up in an exception when trying to access the missing item (due to the `at(2)` call);
3. The OCL expression would have been simpler (not needing the `asSequence()` call), in case an ordering relation on `parents` had been imposed at the model level.

Ordering the `parents` collection with respect to sex (such that the first element points to the mother and the second to the father) allows writing a more detailed invariant shape. Following, there is the OCL specification we propose in this respect, in case both parents are known. In case of invariant violation, the debugging information is precise, allowing to easily eliminate the error’s cause.

---

```
context Person
  inv parentsSex:
    self.parents->size = 2 implies
      self.parents->first.sex = Sex::female and self.parents->last.sex = Sex::male
```

---

Yet, a correct understanding of the model in question leads to the conclusion that the mere constraint regarding the parents’ sex is insufficient, despite its explicit specification for each parent. As rightly noticed in [Cab11], a person cannot be its own child. A corresponding OCL constraint should be therefore explicitly specified.

---

```
context Person
  inv notSelfParent:
    self.parents->select(p | p = self)->isEmpty
```

---

However, restricting the age difference among parents and children to be at least the minimum age starting from which human reproduction is possible (we have considered the age of sixteen) leads to a stronger and finer constraint than the previous, that may be stated as follows:

---

```

context Person
  inv parentsAge:
    self.parents->reject(p | p.age - self.age >= 16)->isEmpty

```

---

In the above expression, each `Person` is assumed to own an `age` attribute. The `reject` subexpression evaluates to the collection of parents breaking the constraint in question.

The fulfillment of this constraint could be also required at any point in the construction of the genealogical tree. Assuming any parent to be created prior to any of its children, this restriction could be stated by means of the precondition included in the contract below.

---

```

context Person::addChildren(p:Person)
  pre childrenAge:
    self.children->excludes(p) and self.age - p.age >= 16
  post childrenAge:
    self.children->includes(p)

```

---

The conclusion that emerges so far is that the lack of OCL specifications prohibiting undesired model instances (such as parents having the same sex, self-parentship or the lack of a minimum age difference among parents and children) seriously compromises model's integrity. The first prerequisite for models to reach their purpose is to have a complete and correct specification of the requirements, and to deeply understand them. An incomplete specification reveals its limits when trying to answer questions on various situations that may arise. Specifying and evaluating OCL constraints should enable us to identify and eliminate bugs, by correcting the requirements and the OCL specifications themselves. Another conclusion, as important, is that the model proposed in the analyzed paper does not fully meet the needs of such a problem, and we are therefore invited to seek for a better solution.

## 4 Modeling Alternatives

A model equivalent to that of Figure 1, but which is more adequate to the specification of the required constraints, is the one included in Figure 2. The model in question contains two recursive

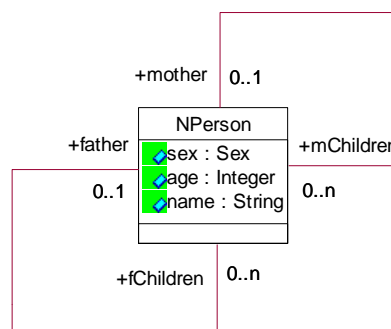


Figure 2: An alternative model for expressing parents-children relationships

associations: one named `MotherChildren`, with roles `mother[0..1]` and `mChildren[*]` and the other named `FatherChildren`, with roles `father[0..1]` and `fChildren[*]`.

Within this model, the constraint regarding the parent's sex can be stated as proposed below.

---

```

context NPerson
inv parentsSex:
  self.mother->size = 1 implies self.mother.sex = Sex::female and
  self.father->size = 1 implies self.father.sex = Sex::male

```

---

Compared to its equivalent constraint stated for the model in Figure 1, the above one is wider, since it also considers the case with a single parent and checks the sex constraint corresponding to the parent in question. The problem with the previous model (the one in Figure 1) is that we cannot count on an ordering when there is a single parent reference available. The parent in question would always be on the first position, irrespective of its sex. As opposed to this, in Figure 2, the parents' roles are explicitly specified, with no extra memory required.

With respect to the second constraint, we propose the following specification in context of the model from Figure 2.

---

```

context NPerson
inv parentsAge:
  self.mChildren->reject(p | p.age - self.age >= 16)->isEmpty and
  self.fChildren->reject(p | p.age - self.age >= 16)->isEmpty

```

---

The corresponding pre and post-conditions are similar to their equivalents from the previous section, therefore their specification could be left to students, as homework.

## 5 Explaining the Intended Model Uses

Any requirements specification should include a detailed description of the intended model uses. In case of the model under consideration, it is important to know what kind of information may be required from it. Is it merely the list of parents and that of all ancestors? Do we want the list of ancestors ordered, with each element containing parents-related information, in case such information is available? Do we only need information regarding the male descendents of a person?

In case of the initial model in which the recursive association is ordered, the list of all ancestors of a person can be easily computed as follows.

---

```

context Person
def allAncestors() : Sequence(Person) =
  self.parents->union(self.parents.allAncestors())

```

---

The evaluation result for the constraint above is correct only if we assume the genealogical tree as loops-free. This latter constraint is implied by the one restricting the minimum age difference between parents and children. In the absence of this assumption, the OCL expression's complexity increases.

A simpler alternative for this case employs the semantic closure operation on collections. This operation, now included in OCL 2.3, has been implemented in OCLE ever since its first release and returns a set.

---

```

context Person
def allAncestors() : Sequence(Person) =
  (Sequence{self}->closure(p | p.parents))->asSequence

```

---

The `asSequence()` operation orders the collection it is applied on with respect to the insertion time of each element. In OCLE, elements appear in the same order they were added to the set.

In case of the model from Figure 2, the use of the Tuple data type allows us to design a specification enclosing more suggestive information. Following, there is the proposed specification.

---

```
context Nperson
def parents:TupleType(mother:Nperson, father:NPerson) =
  Tuple{mother = self.mother, father = self.father}

def allAncestors:Sequence(TupleType(mother:Nperson, father:NPerson))=
  Sequence(self.parents)->closure(i |
    i.mother.parents, i.father.parents)->asSequence->prepend(self.parents)
```

---

## 6 Using snapshots to better understand and improve the requirements and the model

One of the primary roles of constraints is to avoid different interpretations of the same model. Therefore, the specification process must be seen as an invitation for a complete and rigorous description of the problem, including the constraints that are part of the model. The model must conform to the informally described requirements, even before attaching constraints. In case this condition is not fulfilled, the constraints specification process must ask for additional information, meant to support an improved description of requirements, a deeper understanding of the problem, and by consequence, a clear model specification.

Despite its importance, as far as we know, this issue has not been approached in the literature. That is why, in the following, we will try to analyze the second example presented in [Tod11], concerning a library model. This example aims to model the contractual relationships between a library, its users and companies associated with the library. The only informal specification provided is the following: “In this example, we’ll assume that the library offers a subscription to each person employed in an associated company. In this case, the employee does not have a contract with the library but with the society he works for, instead. So we add the following constraint (also shown in Figure 10): ...”.

First of all, we would like to remind the definition of a contract, as taken from [glo]: “A binding agreement between two or more parties for performing, or refraining from performing, some specified act(s) in exchange for lawful consideration.” According to this definition and to the informal description of requirements, we conclude that, in our case, the parts in the contract are: the user on the one hand, and the library or the company, on the other hand. Therefore, the natural context for the constraint is `Contract`. As one of the involved parts is always the user, the other part is either the library (in case the user is not employed in any of the library’s associated companies), or the company (in case the user is an employee of the company in question).

Regarding the conformance among requirements, on the one side, and model, on the other side (the class diagram, the invariant presented in Figure 10 and the snapshots given in Figures 12 and 13), several questions arise. Since a thorough analysis is not allowed by the space constraints of this paper, in the following, we will only approach the major aspects related to the probable

usage of the model. In our opinion, this concerns the information system of a library, that stores information about library users, associated companies, books, book copies and loans. The library may have many users and different associated companies.

Since the `Library` concept is missing from the model, we have no guaranty that, in case the user is unemployed, the second participant to the contract is the library. Moreover, in case the user is employed, the invariant proposed in [Tod11] does not ensure that both the user and the corresponding company are the participants to the contract. In our vision, two invariants are needed - one in the context of `Contract` and the other in the context of `User`.

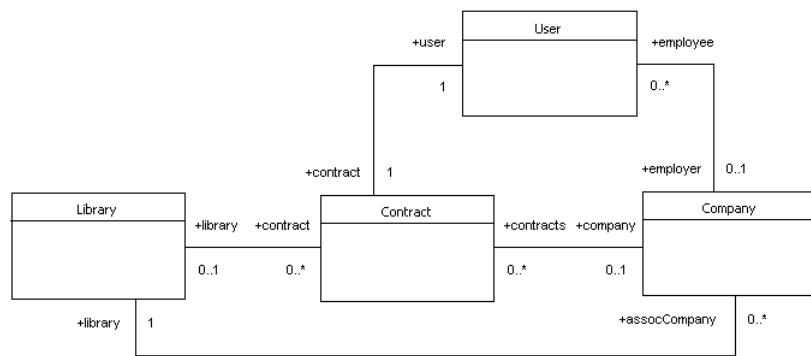


Figure 3: A revised version of an excerpt of the library model from [Tod11]

---

```

context Contract
  inv onlyOneSecondParticipant:
    self.library->isEmpty xor self.company->isEmpty

context User
  inv theContractIsWithTheEmployer:
    if self.employer->isEmpty
    then self.contract.library->notEmpty
    else self.employer = self.contract.company
    endif
  
```

---

The above constraints forbid situations like those from Figure 4 (in which the user `u1` has a contract `c1` both with the library `l1` and the company `comp1`) and Figure 5 (in which the user is employed by `comp3`, but its contract `c2` is with `comp2`). This undesirable model

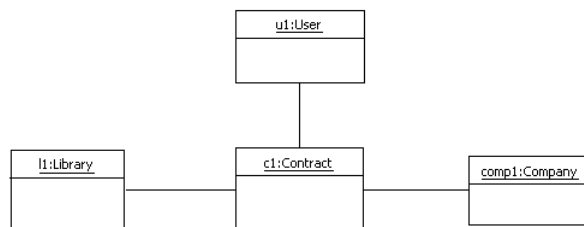


Figure 4: The user has a contract with both the library and the company

instantiations are not ruled out by the invariant proposed in [Tod11] in the `User` context, namely `self.contract->notEmpty xor self.company <> null`.



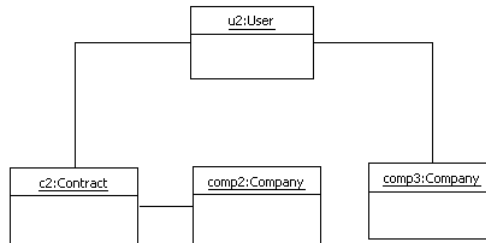


Figure 5: The user is employed by `comp3`, but its contract `c2` is with `comp2`

Even more, in Figure 12 from [Tod11], `contractB65` and `contractR43` have only one participant, `company80Y`, a strange situation in our opinion. Also, in the same figure, if `userT6D` is unemployed by `company80Y`, and, by consequence, `contractQVR` is between `userT6D` and the library, we cannot understand why `company80Y` (which does not include among its employees `userT6D`) has a reference towards `contractQVR` between `userT6D` and the library.

Unfortunately, as stated before, our questions do not stop here. In Figure 10 from [Tod11], a user may have many contracts, but in the requirements a different situation is mentioned. In the class diagram of Figure 10, all role names are implicit, which burdens the intelligibility of the model.

In this example, the snapshots meant to be used for testing have supported us in understanding that the requirements are incomplete and, by consequence, so are the model and the proposed invariant. In such cases, improving the requirements is mandatory.

## 7 Conclusions

The building of rigorous models, which are consistent with the problem requirements and have predictable behavior, relies on the use of constraints. Such constraints are not independent, they refer to the model in question. Consequently, the model's accuracy (in terms of the concepts used, their inter-relationships, as well as conformance to the problem requirements) is a mandatory precondition for the specification of correct and effective constraints. In turn, a full understanding of the model's semantics and usage requires a complete and unambiguous requirements specification. Requirements' validation is therefore mandatory for the specification of useful constraints.

The examples presented in this article illustrate a number of bugs caused by failure to fulfill the above-mentioned requirements. Unfortunately, the literature contains many erroneous OCL specifications, including those concerning the UML static semantics, in all its available releases. Having free access to public resources offered via the web, students should know how to identify and correct errors such as those presented in this article. Our conclusion is that the common denominator for all the analyzed errors is *hastiness*: hastiness in specifying requirements, hastiness in designing the model (OCL specifications included), hastiness in building and interpreting snapshots (test data).

There are, undoubtedly, several ways of teaching OCL. The most popular (which we have

referred as the “classic” one, due to its early use in teaching programming languages), focuses on introducing the language features. OCL being a complementary language, we deemed important to emphasize from the start the gain that can be achieved in terms of model accuracy by an inverted curriculum approach. In this context, we have insisted on the need of a complete and accurate requirements specification, on various possible design approaches for the same problem, as well as on the necessity of testing all specifications by means of snapshots.

However, the teaching and using of OCL has a number of other very important issues that have not been addressed in this article, such as the specifications’ intelligibility, their support for model testing and debugging, code and test data generation, language features, etc. The theme approached by this article only concerns, in our view, a first introduction to the language and its purpose.

**Acknowledgements:** This work was supported by CNCSIS-UEFISCSU, project number PNII-IDEI 2049/2008.

## Bibliography

- [Cab11] J. Cabot. Common UML errors (I): Infinite recursive associations. 2011. <http://modeling-languages.com/common-uml-errors-i-infinite-recursive-associations/>.
- [CD10] J. Chimiak-Opoka, B. Demuth. Teaching OCL Standard Library: First Part of an OCL 2.x Course. *ECEASST* 34, 2010.
- [glo] InvestorWords. <http://www.investorwords.com/1079/contract.html>.
- [LCI] LCI (Laboratorul de Cercetare în Informatică). Object Constraint Language Environment (OCLE). <http://lci.cs.ubbcluj.ro/ocle/>.
- [Nie11] O. Nierstrasz. Synchronizing Models and Code. 2011. Invited Talk at TOOLS 2011 Federated Conference, <http://toolseurope2011.lcc.uma.es/#speakers>.
- [OMG11] OMG (Object Management Group). Object Constraint Language (OCL), Version 2.3 Beta 2. 2011. <http://www.omg.org/spec/OCL/2.3/Beta2/PDF>.
- [por] The OCL portal. [http://st.inf.tu-dresden.de/ocl/index.php?option=com\\_content&view=category&id=5&Itemid=30](http://st.inf.tu-dresden.de/ocl/index.php?option=com_content&view=category&id=5&Itemid=30).
- [Tod11] A. Todorova. Produce more accurate domain models by using OCL constraints. 2011. <https://www.ibm.com/developerworks/rational/library/accurate-domain-models-using-ocl-constraints-rational-software-architect/>.
- [use] A UML-based Specification Environment. <http://www.db.informatik.uni-bremen.de/projects/USE>.
- [Ven04] B. Venners. Abstraction and Efficiency. A Conversation with Bjarne Stroustrup - Part III. 2004. <http://www.artima.com/intv/abstreff2.html>.

# Teaching MDE through the Formal Verification of Process Models

Benoit Combemale<sup>2</sup>, Xavier Crégut<sup>1</sup>, Arnaud Dieumegard<sup>1</sup>, Marc Pantel<sup>1</sup> and Faiez Zalila<sup>1</sup>

<sup>1</sup> [Firstname.Lastname@enseeiht.fr](mailto:Firstname.Lastname@enseeiht.fr)

Université de Toulouse, IRIT – France

<sup>2</sup> [Firstname.Lastname@irisa.fr](mailto:Firstname.Lastname@irisa.fr)

Université de Rennes 1, IRISA – France

**Abstract:** Model Driven Engineering (MDE) plays now a key role in the development of Safety Critical Systems (SCS) relying on Domain Specific Modeling Languages (DSML), early Validation and Verification (V&V) and Automatic Code Generation. It reduces the development cost and improves the system qualities. This contribution describes the content and provides the lessons learned from a course about MDE for SCS development that started five years ago. This course focuses on the use of DSML to allow early V&V based on formal methods. It relies on a process modeling and verification case study that leads students to experiment the various MDE tools that ease the definition and implementation of Domain Specific CASE tool. MDE is introduced as a bridge between the formal methods course that introduces specific formalisms (e.g., Petri net) dedicated to efficient verification tools (e.g., model-checker), and the software modeling course that promotes user oriented abstraction through DSML. The unification power of MDE is also highlighted by the case study that does not target traditional executable software.

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

## 1 Introduction

The Model Driven Engineering (MDE) 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. Nowadays, in these domains, model based early verification and validation activities conducted by engineers is a common practice. It is also widely acknowledged that testing does not allow to reach the required level of safety at affordable costs and that MDE techniques allow to raise the quality of the developed system while opening doors toward formal verification. This course gathers these elements to prepare the students to enter efficiently the system engineering world. Most of the time, when a verification is done in MDE, it relies on OCL constraints (as in [BKS09],[GP10]). This course relies on model-checking for Petri nets in the context of MDE to verify behavioral correctness that are hardly verified with static verification solutions like OCL. The necessity for earlier verifications is highlighted by the non-executable software targeted in this course: a simple process modeling language. This

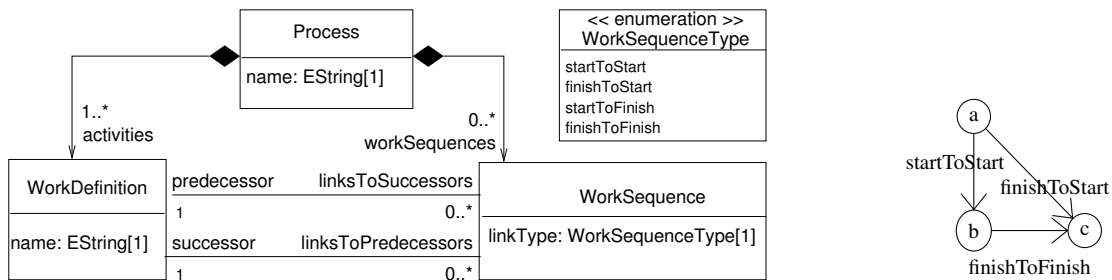


Figure 1: First metamodel of SimplePDL (left) and one conforming model (right)

allows us to explain 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<sup>1</sup>. The key principle was to apply the various technologies allowing to implement a Domain Specific CASE tool such as TOPCASED<sup>2</sup> to a verification driven case study. This case study has also been used to illustrate the TOPCASED approach to system engineering.

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 termination verification activities.

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.

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

<sup>1</sup> The authors wish to thank F. Vernadat that gives that part of the course and initiated the whole activity.

<sup>2</sup> <http://www.topcased.org>

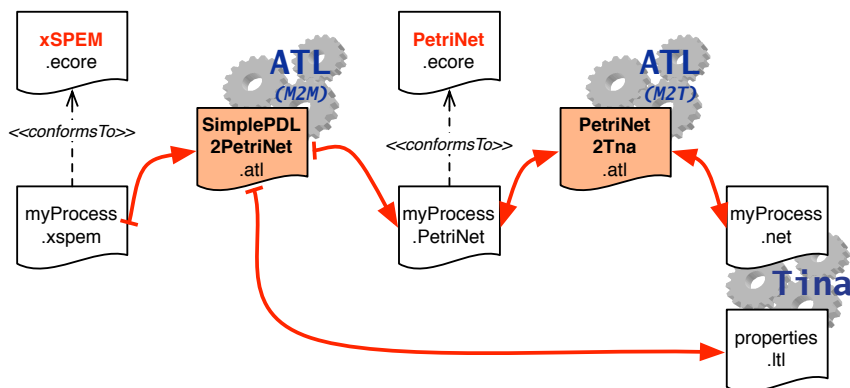


Figure 2: Approach to evaluate behavioral properties on a process model

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<sup>3</sup> 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<sup>4</sup> 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.

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

One lecture ( $\approx 2h$ ) and a practical work session ( $\approx 2h$ ) is allocated to each topic. Lectures present the main concepts and associated standards. For each topic<sup>5</sup>, except the two last one, the

<sup>3</sup> <http://www.laas.fr/tina/>

<sup>4</sup> 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.

<sup>5</sup> Teaching materials are available at <http://cregut.perso.enseeiht.fr/2010/> but, unfortunately, currently only in French.

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<sup>6</sup>. 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<sup>7</sup>. 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.

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.

<sup>6</sup> cf. <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/indigor>

<sup>7</sup> 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

OCLE 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<sup>8</sup> 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.

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

<sup>8</sup> <http://www.eclipse.org/Xtext>

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 metamodel.

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 (the Views, the Controller and their mapping to the Model in the MVC pattern) and it provided only few possible customizations.

For three years now, we have switched to GMF Tooling<sup>9</sup> 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<sup>10</sup> 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 Operational QVT, Epsilon Transformation Language, etc. But we feared that students would

---

<sup>9</sup> <http://wiki.eclipse.org/GMF>

<sup>10</sup> <http://www.obeodesigner.com>



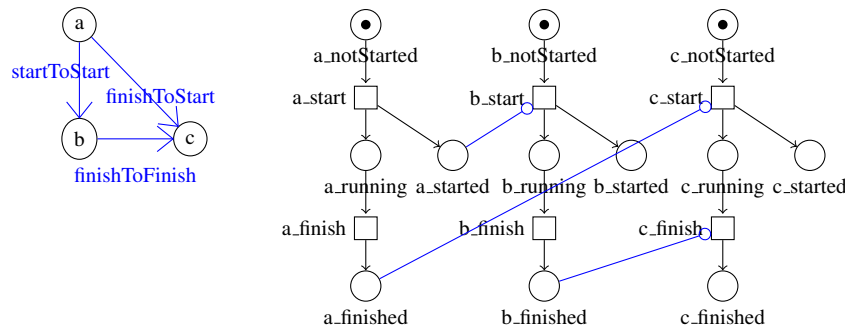


Figure 4: Translation of a process composed of work definitions *a*, *b* and *c* into a Petri net

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*<sup>11</sup> 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 WorkDefinition2PetriNet 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.

<sup>11</sup> 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 as queries. We provide the students with the ATL query that translate a Petri net model into 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 meta-model elements and uses a query language very close to OCL, they complete the query to handle time constraints. 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 main drawback of ATL is that mistakes are generally not highlighted in the editor because they are 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<sub>1</sub>: Resources.** Every work definition requires a set of resources to be executed.

E<sub>1</sub> 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<sub>2</sub>: 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<sub>3</sub>: 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<sub>4</sub>: 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<sub>5</sub>: 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 be 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

- [BKS09] 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.

# Mismatches between industry practice and teaching of model-driven software development

Jon Whittle and John Hutchinson

School of Computing and Communications  
Infolab21, Lancaster University, UK

**Abstract:** EAMDE was a 12 month research project, investigating how industry uses model-driven software development (MDSD). Using quantitative and qualitative research techniques, experiences were collected on the adoption and application of MDSD in 17 companies. The study highlighted examples of good and bad practice that lead to success or failure with MDSD. Some of these practices appear to have ramifications on the way that MDSD, and software modeling more generally, is taught within universities. This paper presents three of the key findings relevant to education: (1) A significant number of successful MDSD companies build their own modeling languages and generators, suggesting a re-orientation of education away from UML notation to fundamental modeling principles; (2) MDSD is generally taught top-down, whereas industry success is more likely when MDSD is applied bottom-up; (3) successful application of MDSD requires skills both in abstract modeling and compilers/optimization; however, these skills tend to be separated in standard CS curricula.

**Keywords:** model-driven software development, education

## 1 Introduction

EAMDE<sup>1</sup> was a twelve month empirical research project, beginning in October 2009, that aimed to investigate how industry applies model-driven software development (MDSD) in practice. The original motivation behind EAMDE was that, whilst there are some clear benefits to MDSD (such as increased productivity through code generation), there are also some potential drawbacks (such as increased training costs or difficulties in integrating legacy code). The project aimed to discover what factors – technical, organizational and social – lead some companies to succeed with MDSD, whereas others fail.

The methodology was to apply quantitative and qualitative research methods to understand when, how and why companies do or do not succeed with MDSD. A three pronged approach was followed: (i) a questionnaire widely disseminated to MDSD practitioners, which received over 400 responses; (ii) in-depth interviews with 22 MDSD professionals from 17 different companies; (iii) on-site studies observing MDSD in practice. In particular, the in-depth interviews were recorded and transcribed, resulting in over 150,000 words describing rich and detailed experiences of the application of MDSD spanning many years and covering a wide range of MDSD knowledge (our interviewees had more than 360 years of software engineering cumulative industrial experience).

The study was intended as an exploratory one, from which key themes would emerge to suggest more formal research hypotheses. Results from the study have previously been

---

<sup>1</sup> Empirical Assessment of the Efficacy of Model Driven Engineering, <http://www.comp.lancs.ac.uk/~eamde>

described [1, 2]. In addition, however, some of the themes that emerged relate to the way that MDS, and software modeling more generally, is taught in universities.

This paper reports briefly on three key findings from the EAMDE study that suggest a reconsideration of the way that modeling is taught. For each, we present the finding, illustrate it using examples from our interviews, and suggest an alternative educational approach.

## 2 Greater Emphasis on Domain-Specific Modeling

A key observation from our study is that MDS may be much more widespread than is generally believed. We have found that *some form of MDS* is practised widely, across a diverse range of industries (including automotive, banking, printing, web applications etc.) and in companies of varied size. The questionnaire respondents, for example, (all of whom were MDS practitioners) were employed in a range of different roles (37% developers, 36% project managers) and represented a good spread of size of company with respect to the number of people involved in software development (e.g. 53% <100 and 20% >1000).

Perhaps surprisingly, a significant number of MDS examples from our study followed domain-specific modeling paradigms. Around 46% of questionnaire respondents used in-house or vendor-provided domain-specific modeling languages (DSLs). Interview data shows that a very successful approach is to develop small DSLs for narrow, well-understood domains. Practical application of domain modeling is pragmatic, where DSLs (and accompanying generators) are developed sometimes in as little as two weeks. Hence, much MDS success is 'hidden' – in the sense that there is very widespread use of mini-DSLs, often textual, and that there may be many such mini-DSLs used within a single project: one interviewee reported on the use of multiple XML-based DSLs to generate 70% of a system, for example.

This evidence of practice has ramifications on the way that modeling is taught. Most modeling courses tend to focus on UML and, furthermore, emphasize the presentation of notation rather than principles. This occurs perhaps because it is straightforward to teach notation – an OMG standard exists that describes the notation explicitly and therefore offers a clear path to follow both for textbook writers and educationalists. In contrast, there are relatively few books that teach modeling *principles*. Our study suggests that, although UML may be an important language to learn, it may be more beneficial to focus on underlying modeling skills. The prevalence of DSLs points to the need for developers with skills in modeling that may be divorced from any specific knowledge of UML notation.

Note that this does not necessarily mean a switch to focus on the notational details of a particular metamodeling framework. Rather, we would advocate an emphasis on getting students to understand the key concepts in a domain and using DSLs to show how such concepts can be structured and organized. Most UML books put notation first and concept structuring is either only secondary or hidden entirely. Although much more limited, books on DSLs (e.g., [3]) seem to do a better job of teaching general principles of (domain) modeling.

The following quote from one of our interviewees is illustrative of the kind of difficulties that a focus on notation can bring.

*What we found is when we taught UML, we'd be teaching about classes and inheritance relationships etc., and there'd be no practical use of this then you'd give an exercise and of course it would be done badly... And what we basically realised is we were spending much much more time discussing the semantics of the language...so we scrapped the course*

*completely... we went out, we bought 4 boxes of the monopoly board game... We gave them this, we said go model the game using these concepts.... We reintroduced the core of UML, class diagrams, in about an hour and then we had people get on with it. We set this up so they would spend 7/8 hours – a whole day – just modelling, playing with the concepts, so they could write down things and then actually work with the sort of physical version of this actually using the monopoly pieces, and actually exploring the concepts within this simple board game – it's quite a challenging board game*

### **3 Teach MDSB Bottom-up rather than Top-down**

Following on from the previous section, our findings lead us to believe that successful MDSB practice tends to be driven from the ground-up. MDSB efforts that are imposed from high-level management typically struggle. As a result, there are fewer examples of the use of MDSB to generate whole systems. Rather than following heavyweight top-down methodologies, successful MDSB practitioners use MDSB as and when it is appropriate and combine it with other methods in a pragmatic fashion.

Those companies that do succeed invariably do so by driving MDSB adoption from the bottom-up: that is, small teams of developers try out aspects of MDSB, which in turn leads them to recognize reusable assets, and eventually MDSB propagates upwards to the organisation as a whole. The following quote from our interviews is typical:

*Yes, yes of course we started just with a few components and we started I think around [the year] 2000 with the first component and now I think 50-60% of all our code is from re-used building blocks but in the beginning it was only 5% or 10%*

This way of working suggests that developers find it easier to get to grips with MDSB when they use it to refactor existing assets from the ground-up rather than trying to abstract from above. In turn, it suggests that modeling should be taught bottom-up rather than top-down.

A typical course in software modeling (and in software engineering, more generally) teaches in a top-down fashion in which requirements models are first developed and are then iteratively refined into architecture, design, code, etc. Students often have a great deal of difficulty proceeding in this manner because it requires formulating abstractions of a system before the concrete details are understood [4].

Given that success in industry seems to be associated with bottom-up introduction of MDSB, we advocate an approach to teaching MDSB that mirrors this practice. Although as yet we have no concrete proposals for such a course, one can imagine a programming-focused module that starts with an existing system and asks students to develop slightly different versions of some features of the system. This process could then be used to discuss the merits of defining reusable assets and abstracting from those assets by defining a modeling language and code generator. The advantage of such a course over most existing modeling courses would be that abstraction skills are introduced and nurtured using very specific, concrete examples, which give students a handle on the difficult topic of abstraction.

### **4 Integrate Abstraction and Compiler Skills within CS Curricula**

As we have seen previously, successful MDSB companies often develop in-house domain-specific languages and code generators, or, in some cases, they extend or modify off-the-shelf tools. In the interviews, we heard of two ways of achieving this task: either use separate

developers, one for modeling and one for writing a generator that could produce optimized code, or use a single developer capable of carrying out both developing a DSL and writing a decent generator for it. The following quote is indicative of the former approach:

*...they couldn't optimize the generated code so the way they had to do it was asking the hardware guys to have more hard disc, more memory, because of the tool. So beforehand we had very small memories and we'd been using C and we were very clear about the memory map and each engineer has a clear view on how much memory space they can use. But this case we cannot do something with the generated code so we simply ask the hardware guys to have more hard disc.*

The interviews tend to suggest that the second way of working is more successful. In other words, companies that are successful with MDSO tend to have MDSO 'gurus' within the organization who possess a combination of skills in both abstraction (modeling, metamodeling, DSLs, etc.) and compiler/optimization. It is interesting to note, however, that these two skill sets – abstraction/modeling and compiler/optimizations – tend to be quite far apart in typical CS curricula. Although it may be common for a compiler course to be included as a core module, taken by all CS students, software engineering is typically taught very separately from this and usually does not make much reference to it, if any. The danger is that students specializing in software engineering receive only very basic training in compiler/optimization skills, which may cause problems when applying MDSO in practice, as the following quote illustrates:

*The tool itself is very inefficient. But I also developed a lot of CASE tools whilst I was at the university as a PhD student, but if somebody asks me how to optimize your code from your CASE tool, then I don't know how to do that!*

Based on our study, we would argue that perhaps abstraction and compilation/optimization techniques ought to be taught together in an integrated fashion. Although further study is needed to validate this hypothesis, such an idea would radically alter the way that software engineering is taught and would skill-up a new generation of developers capable of both abstracting in a problem space and transitioning to a solution space in an efficient manner.

## 5 Conclusion

This paper has presented some insights from a large scale study on industry adoption of MDSO, concentrating on those findings relevant to MDSO education. The paper has suggested three ways to reconsider the way MDSO is taught, to better align with industry practice. At this point, these suggestions are untested so further educational research is required to investigate their potential benefits and understand their drawbacks.

## 6 References

- [1] John Hutchinson, Jon Whittle, Mark Rouncefield, Steinar Kristoffersen: Empirical assessment of MDE in industry. ICSE 2011: 471-480
- [2] John Hutchinson, Mark Rouncefield, Jon Whittle: Model-driven engineering practices in industry. ICSE 2011: 633-642
- [3] Tony Clark, Paul Sammut, James Willans: Applied Metamodelling, A Foundation for Language Development, 2<sup>nd</sup> Edition. Ceteva, 2008.
- [4] Jeff Kramer: Is abstraction the key to computing? Commun. ACM 50(4): 36-42 (2007)



# Ready for the Industry: A Practical Approach to Teaching MDE

Gordana Milosavljević<sup>1</sup>, Igor Dejanović<sup>2</sup> and Branko Perišić<sup>3</sup>

<sup>1</sup> [grist@uns.ac.rs](mailto:grist@uns.ac.rs)

<sup>2</sup> [igord@uns.ac.rs](mailto:igord@uns.ac.rs)

<sup>3</sup> [perisic@uns.ac.rs](mailto:perisic@uns.ac.rs)

Faculty of Technical Sciences  
University of Novi Sad, Serbia

**Abstract:** This paper presents an approach to teaching a course dealing with MDE topics where the focus is on acquiring practical skills. The main goal was to enable the students to apply the knowledge from this course in practice after graduation. This was achieved by directing practical lectures not only to exercising different MDE techniques, but also to their use in a particular field of software development – developing business applications.

**Keywords:** MDE, practical skills, business applications

## 1 Introduction

In order to survive in competitive markets, globalised economy, and rapid technological changes, companies – and the software they use – need to be quickly adaptable to changes in their environment [TBK99]. Rapid and cheap software development and efficient adaptation to changes are requirements that are increasingly often posed to software development teams.

Recognising the needs of IT companies for staff that could satisfy such requirements, the master studies curriculum at the Computing and Control Department, Faculty of Technical Sciences, University of Novi Sad was extended with the new course titled Rapid Systems Development Methodologies three years ago. This course aims to introduce students to methodologies and techniques for efficient software development, mainly Model Driven Engineering (MDE) technologies and agile methodologies. This paper focuses on describing the MDE topics covered by the course.

The rest of the paper is structured as follows. [Section 2](#) presents information on the background of students attending this course. [Section 3](#) describes an overview of theoretical topics. [Section 4](#) presents the organisation of practical topics. [Section 5](#) reviews the statistics of the course held last year. [Section 6](#) concludes the paper.

## 2 Course Background

Starting from the third academic year, most courses in software engineering and information systems are organised in an environment that, as much as possible, resembles the real work environment the students will experience after graduation. Each course comprises theoretical and practical topics, where the practical assignments are carried out by student teams. A team,

comprising 3-5 students, is assigned a task of developing a relatively complex software solution. Development includes modeling, where models are used to specify (pieces of) solutions and fostering the communication among team members as well as the team and the instructor. Team work is supported by version control software (e.g., Subversion or Mercurial) and the software for progress monitoring and project management (e.g., Trac). The practical topics usually include solving a (part of a) real world problem, depending on the course. Software design and implementation in the initial stages are lead by the instructor who participates in specifying the basic elements of the architecture, while in the latter stages the team operates independently while the instructor inspects the results and helps the team as needed. The implementation contains unit tests (e.g., JUnit) and documentation (e.g., javadoc). A short video with presentations of student projects from various courses is available at [\[Stu\]](#).

The most students who attend the Rapid Systems Development Methodologies course also have attended the following courses: Software Requirements Specification and Software Modeling (introduced right after the courses in OO programming), Compilers, Software Engineering Topics, Business Information Systems, Net-Centric Computing, Web Services, etc, so they possess the adequate theoretical and practical prerequisites for this course.

### 3 Course Goals and Organisation

The goal we strive for in the Rapid Systems Development Methodologies course is that students, while working in almost a real-world environment, realise the power of MDE technologies so that they may apply their knowledge later in practice. We have also aimed at providing the students the sufficient theoretical basis so that they can expand their knowledge of the subject in their later assignments and master theses. Since the course spans one semester, comprising 3 hours for theoretical topics and 3 hours for practical exercises per week, time limits had to be considered in order to achieve the ambitious goals of the course.

MDE is a very dynamic field currently consisting of many approaches: Model Driven Architecture (MDA), Model Driven Software Development (MDS), Executable UML (xUML), Domain Specific Modeling (DSM), etc. Each of these approaches could be a main subject for a university course. The dilemma was whether to choose a single approach (which one?) and study it in detail, or to try to present the students with the overview of the whole field with less detail.

Since the students not willing to pursue doctoral studies will not have another chance to study these topics, we opted for the second approach: to provide a broad overview of the field, with analysis of advantages and shortcomings of different approaches, so that the students can choose the most suitable one in a particular situation. The lectures were focused on the core ideas, with less emphasis on formalisms that would require more time to be presented. Each topic was supplied with references for additional reading and links to university courses that deal with the same topic in a different way, in order to provide the students with a basis for further research on a particular topic. The main literature for shaping the theoretical lectures included [\[SVBS06\]](#), [\[KT08\]](#), [\[KWB03\]](#), [\[FP10\]](#), and [\[MB02\]](#), and OMG documents and standards [\[Met06\]](#), [\[Obj06\]](#), and [\[MOF07\]](#).

Theoretical lectures introduce students with the following topics:

- Introduction to MDE: an overview, goals, different approaches, models, metamodels, meta-metamodels (1 lecture)
- Introduction to MDA: CIM, PIM, PSM, transformations, MOF, UML Infrastructure and Superstructure, OCL, XMI (3 lectures)
- MDSD and DSLs: graphical and textual DSLs, DSL construction, implementing frameworks and code generators, techniques for combining manually written and generated code, generation of different artifacts (documentation, tests, installation and configuration scripts), DSL creation tools (5 lectures)
- Executable UML (1 lecture)
- Agile methodologies: Agile Modeling, Extreme Programming, Scrum, Lean Software Development (2 lectures)

The practical lectures were organised so that the students, while developing a real-world project, gain a deeper understanding of the most of the concepts covered in theoretical lectures and practical skills in the MDE field. Details on the organisation of practical lectures are presented in the next Section.

## 4 Organisation of the Practical Lectures

While analysing various university courses with similar topics, we have noted that practical exercises are usually carried out as a series of small examples illustrating certain theoretical aspects of MDE, where the course usually ends without an application of MDE to a certain software development field. However, knowing the possibilities for application of MDE is crucial if we want the students to use this knowledge in practice.

Since we are involved in the application of MDE technologies in developing complex business information systems (for example, see [DMPT10, PMDM11]), the example we have chosen for the practical assignments is a simplified environment for the model-driven development of three-tier business applications, with the usual infrastructure [Fow03]:

- data is persisted in a relational database,
- data management operations are implemented in the middle tier, and
- graphical client application (desktop or web) implements only the user interface and manipulates data by invoking the operations residing in the middle tier.

In their previous courses (Business Information Systems, Net-Centric Computing, Web Services) students have carried out the implementation of this kind of applications through the following steps: (1) specifying the business domain model in terms of UML class diagrams using a modeling tool (Sybase PowerDesigner [Pow] was used), (2) automatic transformation of this model into the database model which is used, with some manual optimisations, to generate SQL scripts, (3) semi-manual implementation of the middle tier (class diagrams are used to generate Java classes which were manually annotated for Hibernate [Hib]), (4) manual implementation of the user interface (Swing- or web-based GUI), and (5) manual implementation of complex business transactions and reports.

We wanted the students to develop an environment for model-driven development of business applications that would facilitate automated implementation of all three application layers (excluding the implementation of complex business transactions and reports) and to use it to develop a small business application (e.g., human resources, stock management) so they can grasp the benefits and shortcomings of both approaches.

The generation of database schema and middle tier code is supported by a number of tools, while the generation of the user interface based on declarative specifications is the subject of intensive research efforts [Sil01]. The description of the user interface in the general case uses different types of models (presentation model, content model, navigation model, interaction model, etc.) but the number of models can be reduced if we target applications with a specific user interface functionality. In [PMDM11] we have presented a kind of user interface based on our HCI (human-computer interaction) standard that specifies functional and visual features of different course-grained client application components. For student projects, we have restricted the component set to the following:

- standard data management form (associated to a single entity class or a database table, providing display, creation, update, removal, and search of the objects of the given class, traversal to forms of related classes, and invoking the reports and complex transactions – see [Figure 2](#) and [Figure 3](#)),
- main form (contains a menu for invoking standard forms), and
- parameter form (for entering parameters for transactions and reports).

This way, it is possible to specify a relatively simple user interface language that can be integrated with languages for the specification of other aspects of the system with transformations, according to the MDA approach, or merging in order to create a comprehensive DSL for describing business applications, according to the MDSD approach (see [Figure 1](#)). During the design of the language(s) for specifying business applications we have discussed different options. The existence of multiple languages for describing different aspects of the system ([Figure 1a](#)) enables the separation of concerns and the independent work of different kind of specialists, but the synchronisation of models may introduce an overhead in project development. Discarding the database model (e.g., using the default object/relational transformation provided by Hibernate) reduces both the overhead and the number of development options ([Figure 1b](#)). If we introduce the rules like: (1) all problem domain classes are persistent, (2) each association end with the multiplicity being \* is lazily loaded, (3) each association end with the multiplicity being 0..1 or 1 is eagerly loaded, etc, the middleware model is not needed any longer, while all relevant information can be extracted from the business domain model ([Figure 1c](#)). Introducing such rules makes the solution simpler and the development process faster, but reduces the model expressiveness (which may be appropriate in some situations). In order to prevent the developers feeling too constrained or too overwhelmed, careful planning is needed. Student teams had to choose their own strategy and explain their choices.

Since the main topic of this course is not MDE itself, but rather the rapid application development based on MDE, the goal was to model and generate only the elements of the application for which that approach is more efficient than the manual implementation. The students also had to decide on the way to reach a balance between automatic code generation and manual coding,

and to define mechanisms for their integration (based on [SVBS06]). The development method (of both the tools and the business application) was planned to be in the agile manner, so iterative and incremental development was assumed where repetitive code generation was not allowed to damage manually implemented segments.

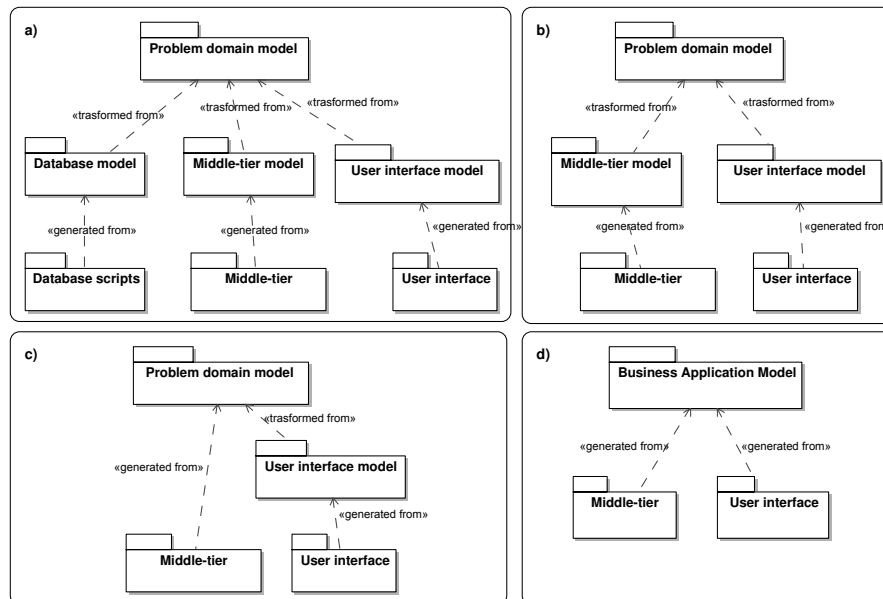


Figure 1: Different approaches to specifying business applications

As a basis for the business application development environment we chose MagicDraw [Mag] since it is fully UML 2.0 compatible, supports the specification of OCL rules, and is extensible with plugins written in Java. Plugin implementation uses MagicDraw Open API, a very powerful library that provides access to the object model representation, changing the model programmatically, developing custom transformations, and customising menus and toolbars. Additional gain is provided by the already implemented transformations between object and relational models that can be programmatically invoked. A possible shortcoming is the fact that a DSL can only be implemented as a UML profile. However, MagicDraw features a DSL Customisation Engine that enables hiding some UML elements (defining custom toolbars for the new language, customising properties dialogs, specifying new symbols, etc). For transformations from the model to the program code we used the FreeMarker [Fre] template engine.

The practical lectures were structured as follows:

- the development of the problem domain model in terms of class diagrams for stock management (1 lecture)
- discussion on business rules for stock management and their specification using OCL (in order to exercise a specific way of thinking imposed by the use of predicate logic) (2 lectures)
- designing the language(s) for business applications design: developing the initial meta-model of the user interface language and the middle-tier language and the discussion on

different approaches of their integration (students were encouraged to choose the approach they want and to change metamodels as they see fit), the mapping of the sample metamodel to the UML profile, specifying chosen OCL constraints, customisation of the UML profile in order to hide unneeded metaclasses and metaattributes (2 lectures)

- designing the client application framework based on our HCI standards components: discussion on different approaches, from minimalistic framework where the focus is on the code generation, to a very complex framework that interprets the model (or data extracted from the model) in the xUML manner, techniques for integration of framework and generated code (students could also choose their approach) (2 lectures)
- implementation of a MagicDraw plugin and a sample transformation using Open API (for teams that chose MDA like approach) (1 lecture)
- designing the code generator as a MagicDraw plugin (the students were given a simplified code generator that connected all of the used technologies in order to change and/or extend it) (2 lectures)
- discussion on alternative sources of meta-information (when the model is not available, for example in an reengineering project): Java reflection, database metadata, etc. (1 lecture)
- instructor-supervised development of the environment (3 lectures).

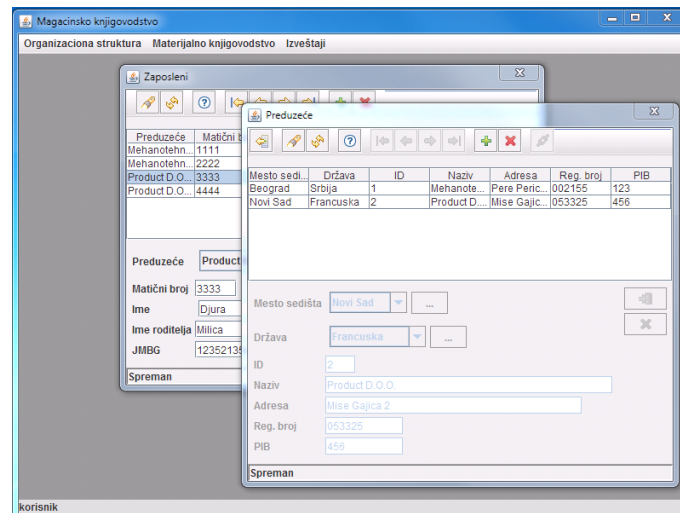


Figure 2: An example of a generated web application (main form and two standard forms)

## 5 Course Statistics

- In the last academic year, 36 students attended the course, grouped into 9 teams. One student has failed to complete the course.
- Two teams have implemented the transformation of the problem domain model into the user interface model (see Figure 1c), while the remaining 7 teams had one model that was used directly to generate middle tier and user interface code (see Figure 1d).

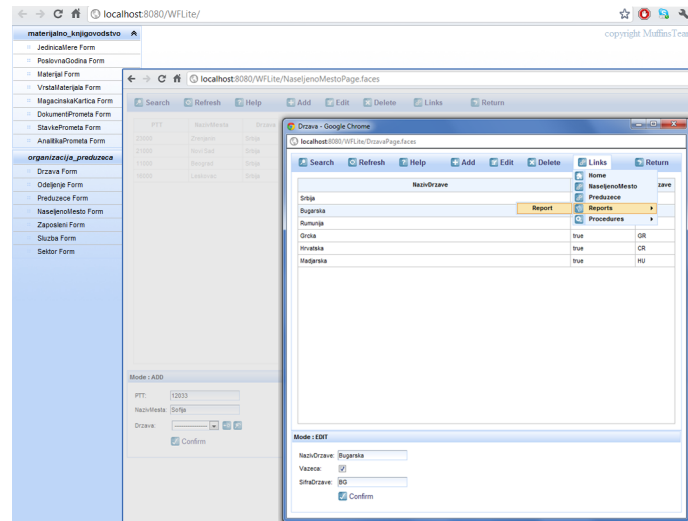


Figure 3: An example of a generated web application (main form and two standard web forms)

- One team has implemented a generator of web-based applications, while 8 teams have chosen to target desktop applications.
- One team was focused on the development of a generic framework while generating code only for EJB entities and form descriptions interpreted by the framework, 6 teams had a minimalistic framework while the focus was on code generation (over 20000 LOC for a relatively small business application), and 2 teams have found – in our opinion – a balance between the use of the framework and the code generator.
- Two teams have kept the initially proposed metamodel, while the remaining 7 teams have improved it.
- One team has implemented a parser for simplified OCL expressions in order to generate derived (calculated, aggregated, etc) form fields.
- No team has managed to fully implement OCL constraints (MagicDraw supports a subset of OCL, so some correct expressions were not accepted).
- The time needed to implement this project ranged from 4 to 6 weeks. The number of lines of code (code generator, framework, FreeMarker templates) ranged from 4500 to 8500. The initial code generator that was used as a starting point contained 1500 LOC, so each student wrote about 1000 LOC, which was a moderate sized task. The project of a generator of web-based applications had the most LOC (8500) because it contained a lot of artifacts that had to be maintained (JSF was used for the user interface).

Figure 4 presents an SVN statistics of an average student team project. The steep rise of LOC at the end of the curve stems from committing the generated code and documentation of the business application into the repository. Although it was emphasized that the generated artifacts need not be versioned, this curve is useful to illustrate the gain made possible with the use of automation.

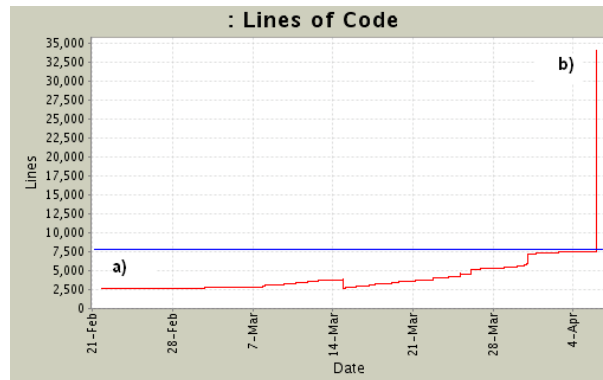


Figure 4: SVN statistics – LOC for a single team project a) Tool LOC b) Generated application LOC

The poll taken upon the completion of projects has resulted with the following students' observations:

- The project was not too hard and it didn't take too much time. Compared to the assignments in other courses, the complexity of this project was rated in the middle.
- The course was evaluated with the average mark of 9.93 (5 being the lowest, 10 being the highest mark).
- The importance of the course topics for their future work was rated as high.

## 6 Conclusions

This paper has presented an approach to conducting a university course dealing with MDE topics where the focus was on acquiring practical skills. The initial goal set forth, making the students grasp the power of MDE technologies by working in a real-life environment, is achieved in our opinion.

Students have showed a satisfactory knowledge of the field, thanks to the instructor-supervised work who gave them enough guidelines but also the opportunities for creative work and making decisions. By implementing the same application in two ways (manually, in previous courses, and using model-driven techniques, in this course) the students were able to compare the two approaches, and realise the benefits of the latter. The course has fostered the students' interest in the field, and ten students have started the work on their master theses based on the topics presented in the course: A generic DSL editor, Development environment for end-user specification of static and dynamic aspects of business applications (DSL, editor, framework, generator, or interpreter) – 3 related theses, Code generator for OCL business rules, Test-cases generator, Adaptive aspect-oriented frameworks for desktop and web applications (2 related theses), DSL design time integration, Multiple DSL generator interoperability. The last two theses are being developed in cooperation with dr Nikola Milanović within the BIZWARE project [Biz].

**Acknowledgements:** This work was partially funded by the grant III-47003 of Ministry of



Science and Technological Development of the Republic of Serbia.

## Bibliography

- [Biz] BIZWARE – Modellgetriebene Softwarekonzeption und -entwicklung.  
<http://www.bizware.org>
- [DMPT10] I. Dejanović, G. Milosavljević, B. Perisić, M. Tumbas. A Domain-Specific Language for Defining Static Structure of Database Applications. *Computer Science and Information Systems (ComSIS)* 7(3):409–440, 2010.
- [Fow03] M. Fowler. *Patterns of enterprise application architecture*. The Addison-Wesley signature series. Addison-Wesley, 2003.
- [FP10] M. Fowler, R. Parsons. *Domain-Specific Languages*. Addison Wesley Signature Series. Addison-Wesley, 2010.
- [Fre] FreeMarker.  
<http://freemarker.sourceforge.net>
- [GP10] T. Gjørøseter, A. Prinz. Teaching Model Driven Language Handling. *Electronic Communications of the EASST* 34, 2010.  
<http://journal.ub.tu-berlin.de/eceasst/article/view/591>
- [Hib] Hibernate.  
<http://www.hibernate.org>
- [KT08] S. Kelly, J.-P. Tolvanen. *Domain-specific modeling: enabling full code generation*. A Wiley-Interscience publication. Wiley-Interscience, 2008.
- [KWB03] A. Kleppe, J. Warmer, W. Bast. *MDA explained: the model driven architecture: practice and promise*. The Addison-Wesley object technology series. Addison-Wesley, 2003.
- [Mag] MagicDraw.  
<http://www.magicdraw.com>
- [MB02] S. J. Mellor, M. J. Balcer. *Executable UML: a foundation for model-driven architecture*. Addison-Wesley object technology series. Addison-Wesley, 2002.
- [Met06] Meta Object Facility (MOF) Core Specification, Version 2.0. 2006.  
<http://www.omg.org/spec/MOF/2.0/>
- [MM03] J. Miller, J. Mukerji (eds.). *MDA Guide v 1.0.1*. Object Management Group, 2003.  
<http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>
- [MOF07] MOF 2.0/XMI Mapping, Version 2.1.1. 2007.  
<http://http://www.omg.org/cgi-bin/doc?formal/2007-12-01>

- [Obj06] Object Constraint Language Version 2.0. 2006.  
<http://www.omg.org/spec/OCL/2.0/PDF>
- [OMG] Object Management Group.  
<http://www.omg.org>
- [PMDM11] B. Perisić, G. Milosavljević, I. Dejanović, B. Milosavljević. UML Profile for Specifying User Interfaces of Business Applications. *Computer Science and Information Systems (ComSIS)* 8(2):405–426, 2011.
- [Pow] Sybase PowerDesigner.  
<http://www.sybase.com/products/modelingdevelopment/powerdesigner>
- [Sil01] P. P. D. Silva. User Interface Declarative Models and Development Environments: A Survey. In Palanque and Paternó (eds.), *Interactive Systems Design, Specification, and Verification*. Lecture Notes in Computer Science 1946, pp. 207–226. Springer Berlin / Heidelberg, 2001.
- [Stu] Student Projects Presentation. University of Novi Sad.  
<http://youtu.be/M7KCiHT3StI>
- [SVBS06] T. Stahl, M. Völter, J. Bettin, B. von Stockfleth. *Model-driven software development: technology, engineering, management*. John Wiley, 2006.
- [TBK99] D. P. Truex, R. Baskerville, H. Klein. Growing systems in emergent organizations. *Communications of the ACM* 42:117–123, August 1999.

# Models and Clickers for Teaching Computer Science

Matthias Hauswirth

[Matthias.Hauswirth@usi.ch](mailto:Matthias.Hauswirth@usi.ch)

<http://www.inf.usi.ch/faculty/hauswirth>

Faculty of Informatics

University of Lugano, Lugano, Switzerland

**Abstract:** Many courses in a computer science curriculum, from computer architecture over programming languages to operating systems, discuss complex and intricate mechanisms and systems. Engineers who *develop* such mechanisms and systems (e.g. an operating system) use models to deal with their complexity. Instructors who *teach* the concepts behind those mechanisms and systems often implicitly use models to make those concepts more approachable: they present simplified abstractions and draw diagrams to focus on the essential.

In this position paper we propose to make this implicit use of models explicit. We propose to use models as a teaching tool in *all* courses where they are helpful, not just in a course on models or model-driven development. Moreover, we present an infrastructure, Informa, that provides support for integrating models into an interactive classroom.

**Keywords:** Models for understanding, technology-enhanced learning, clickers

## 1 Introduction

Maybe the best way to teach models is **not** to *teach* the topic of models. Maybe the best way to teach models is to teach other topics by *using* models as an explanatory device. In this paper we propose to subliminally introduce modeling already in early computer science courses, decoupled from the notion of model-driven software engineering, by creating models when explaining and analyzing newly introduced concepts, and by asking students to create or transform models to demonstrate their understanding of such concepts. In the next section we outline the use of models for teaching one example computer science topic: programming. We believe that the approach applies equally well to other courses, especially courses related to systems topics, such as computer architecture, operating systems, compilers, or databases.

## 2 Example: Using Models for Teaching Programming

“Programming” is one of the central topics in a computer science curriculum. When teaching students how to program, instructors have to explain the semantics of a given programming language. Besides giving students a multitude of examples and informal descriptions of the language’s semantics, textbooks and instructors often also use more formal textual (such as BNF

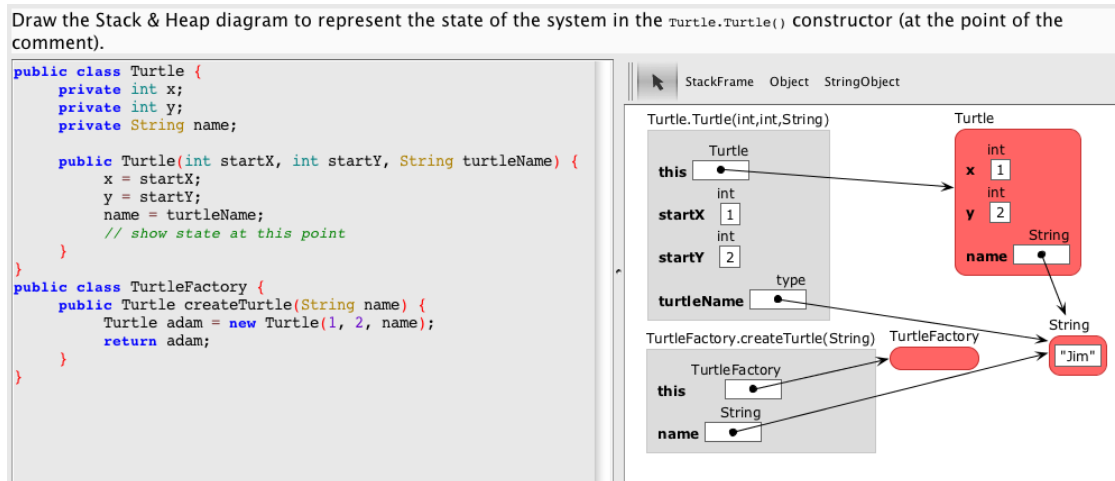


Figure 1: Informa problem requiring a student to model the state of a running program

when discussing syntax) or visual (such as flow charts when discussing control flow) modeling languages.

### 3 Models in the Classroom with Informa

Brandsteidl et al. [BWH11] have shown the benefit of using innovative teaching approaches for teaching modeling. A particularly innovative approach that they have not discussed, the use of classroom response systems (clickers), has been shown [FM06, RPA04] to be beneficial for teaching a broad range of scientific topics. Clickers are a form of remote controls with a small number of buttons that all students bring to a lecture. The instructor then poses a (multiple-choice) question, and the students submit their answers by clicking the corresponding button on their clickers. The instructor’s clicker server collects all responses and provides the instructor (and often also the students, via a classroom projector) with a histogram of the submitted answers. This use of clickers allows an instructor to immediately check whether students understand the topic she just explained, and it allows students to check their understanding already during the lecture.

We propose to use clickers for teaching models, and to do so “subliminally” already in early courses that are not themselves about modeling. To do so, we propose to use Informa [Hau08], a software-based clicker system that is not limited to multiple-choice questions. Informa allows an instructor to post arbitrary types of problems, including problems that require students to write text or to draw graphs. Informa is particularly strong for those kinds of problems that one could call “model transformations”, where students demonstrate their understanding by transforming one representation of a concept (e.g., an XML document) into a different representation (e.g., a node-link diagram representing the document’s DOM tree). Moreover, Informa’s Solve & Evaluate approach [HA09, HA11] enables peer evaluation, where students who already have solved a problem get to evaluate the solutions of their peers.

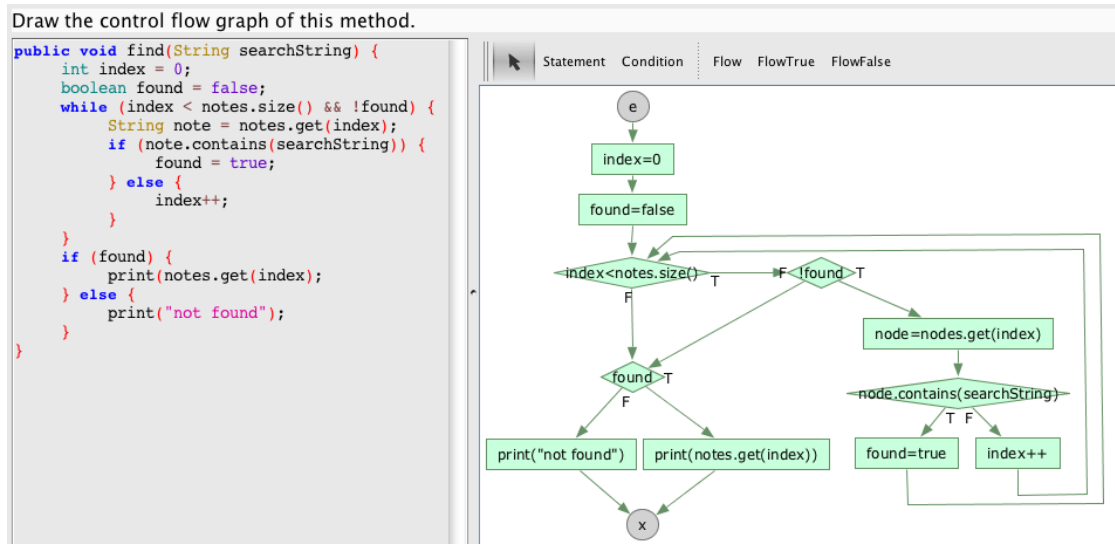


Figure 2: Informa problem requiring a student to model the control flow of a program

Figure 1 shows an example of an Informa modeling problem as it could occur in a programming course. Given the Java program on the left, the students have to model its state (by drawing the diagram on the right), in the form of its call stack frames and its heap objects, at a given point in time in the program’s execution. In this problem type (which we call “Stack & Heap”), we do not use UML object diagrams, but we use a visual language introduced in a textbook [BK06] specifically for modeling the state of a Java program in terms of its stack and heap. The modeling language is limited to the aspects that are essential to the concept being taught, and the user interface of Informa allows students to efficiently<sup>1</sup> draw these diagrams.

Figure 2 shows a different kind of problem type (called “Graph”) supported by Informa. Here the instructor asked the students to model the given method (left) as a control flow graph (right). The Graph problem type is more general, because it allows the instructor to define simple meta-models (visual languages based on node-link diagrams with different types of nodes and edges). This way, instructors can ask students to draw control flow graphs, to draw simple class diagrams (currently nodes do not support structured contents, so only the class names can be shown), to draw call graphs, data-flow graphs, DOM trees, or any other type of model representable by graphs with unstructured nodes and binary links.

Informa supports many other problem types, such as multiple-choice questions, text highlighting, or text editing. Moreover, it is architected as an extensible framework, allowing developers to implement other kinds of problem types, such as editors for the UML and other modeling languages.

<sup>1</sup> Our students found drawing these diagrams with Informa so efficient that they asked us to provide them with the corresponding editor so they could also use it to draw such diagrams when taking notes during lectures.

## 4 Conclusions

Modeling is a complex intellectual activity. The subliminal introduction of models already early in the computer science curriculum may turn the creation, analysis, and transformation of models into a natural activity for our students. Instructors can use models to explain arbitrary phenomena and concepts in the entire spectrum of computer science courses, and they can ask students to demonstrate their understanding by creating or transforming models of the concepts under study. To support this kind of formative assessment using models, we propose to use Informa, an extensible classroom clicker system that is freely available<sup>2</sup>. We hope that the pervasive use of models, together with Informa, will improve the teaching not just of models, but also of any other complex concept in computer science.

## Bibliography

- [BK06] D. J. Barnes, M. Kölling. *Objects First with Java: A Practical Introduction using BlueJ*. Prentice Hall / Pearson Education, 3 edition, 2006.
- [BWH11] M. Brandsteidl, K. Wieland, C. Huemer. Novel Communication Channels in Software Modeling Education. In Dingel and Solberg (eds.), *Models in Software Engineering*. Lecture Notes in Computer Science 6627, pp. 40–54. Springer Berlin / Heidelberg, 2011.  
[http://dx.doi.org/10.1007/978-3-642-21210-9\\_5](http://dx.doi.org/10.1007/978-3-642-21210-9_5)
- [FM06] C. Fies, J. Marshall. Classroom Response Systems: A Review of the Literature. *Journal of Science Education and Technology* 15(1):101–109, March 2006.  
[doi:10.1007/s10956-006-0360-1](http://dx.doi.org/10.1007/s10956-006-0360-1)
- [HA09] M. Hauswirth, A. Adamoli. Solve & Evaluate with Informa: A Java-based Classroom Response System for Teaching Java. In *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*. PPPJ '09, pp. 1–10. ACM, New York, NY, USA, 2009.  
[doi:http://doi.acm.org/10.1145/1596655.1596657](http://doi.acm.org/10.1145/1596655.1596657)  
<http://doi.acm.org/10.1145/1596655.1596657>
- [HA11] M. Hauswirth, A. Adamoli. Teaching Java Programming with the Informa Clicker System. *Science of Computer Programming, Special issue: PPPJ 2009/2010, to appear*, 2011.
- [Hau08] M. Hauswirth. Informa: An Extensible Framework for Group Response Systems. In *Proceedings of the 4th International Conference on Collaborative Computing (CollaborateCom'08)*. November 2008.
- [RPA04] J. Roschelle, W. R. Penuel, L. Abrahamson. Classroom Response and Communication Systems: Research Review and Theory. In *Annual Meeting of the American Educational Research Association*. April 2004.

---

<sup>2</sup> <http://informaclicker.org/>

# Model Correctness Patterns as an Educational Instrument

Azzam Maraee<sup>1</sup>, Mira Balaban<sup>1</sup>, Arnon Strum<sup>2</sup>, Adiel Ashrov<sup>1</sup>

<sup>1</sup> [mari, mira, ashrov@cs.bgu.ac.il](mailto:mari, mira, ashrov@cs.bgu.ac.il)

Computer Science Department

Ben-Gurion University of the Negev, Beer-Sheva 84105, ISRAEL

<sup>2</sup> [sturm@bgu.ac.il](mailto:sturm@bgu.ac.il)

Department of Information Systems Engineering

Ben-Gurion University of the Negev, Beer-Sheva, 84105, ISRAEL

**Abstract:** UML class diagrams play a central role in modeling activities. Given the difficulty in producing high quality models, modelers must be equipped with an awareness of model design problems and the ability to identify and correct such models. In this paper we observe the role of class diagram correctness patterns as an educational instrument for improving class diagram modeling. We describe a catalog of correctness and quality design (anti)-patterns for class diagrams. The patterns characterize problems, analyze their causes and provide repairing advice. Pattern specification requires an enhancement of the class diagram meta-model. The pattern classification has a major role in clarifying design problems. Finally, we describe an actual experiment of using the catalog for teaching modeling.

**Keywords:** Design patterns, correctness and quality patterns, educational instruments, model and Meta-Model levels, abstraction, visual language, pattern catalog.

## 1 Introduction

Models are the backbone of the emerging Model Driven Engineering (MDE) approach, whose major theme is development of software via repeated model transformations. The quality of models used in such a process affects not only the final result, but also the development process itself. In order to achieve high quality, it is important to have educated modelers, who are sensitive to model quality. Indeed, similarly to software construction, model quality can be improved by applying automatic transformations (refactoring), but this approach still cannot replace the need for good modelers.

Design patterns encapsulate expert advice for solving typical problems that might occur in multiple contexts. They fulfill an educational role: Awareness of design patterns yields better solutions. The design patterns trend in software construction gained increasing popularity since the appearance of the design patterns book of the “GoF” [GHJV95]. On the model-level, there is research on formulation of software patterns [LP09, BBC08b], formulation of model-level general and domain specific design anti-patterns [EBL11], proposals for design pattern specification languages [FKGS04, Kim07, BBC08a], and research on the impact of design patterns [TB11]. [EBL10] is a rich catalog of model-level patterns that identify modeling problems.

In this paper we present a catalog of modeling anti-patterns for problems of correctness and quality in class diagram design [BGU10], and discuss its role as an educational instrument, for

improving class diagram modeling. Given the widespread usage of UML class diagrams and the difficulty in producing high quality models, modelers must have deep understanding of model design problems, their identification and repair. Patterns of correctness and quality problems in modeling characterize typical situations in which correctness or quality problems arise, analyze the causes, and suggest possible solutions. Their educational role is to increase the awareness of designers to inter-relationships between modeling elements that create incorrect or low quality models.

To the best of our knowledge, this is the first catalog that provides an in-depth analysis of causes of correctness and quality problems, together with repair advices. The catalog is intended to play an educational role in teaching object modeling. In view of this goal, we discuss the pattern specification language, the problem-oriented organization of the catalog, and its instruction. Finally, we describe an actual experiment in using the catalog in teaching modeling.

Section 2 presents a variety of causes for incorrect class diagram modeling. Section 3 discusses the nature of the pattern specification language and presents an enhancement to UML class diagrams. Section 4 shortly introduces our patterns catalog, and Section 5 describes an experiment that observes the role of the correctness patterns in teaching class diagram modeling. Section 6 concludes the paper.

## 2 Correctness patterns

The two main correctness problems in UML class diagrams are *consistency* [BCG05] and *finite satisfiability* [MMB08]. Consistency deals with necessarily empty classes, and finite satisfiability deals with necessarily empty or infinite classes. Both problems are caused by problematic interaction of constraints.

Figure 1 presents four incorrect class diagrams, in which the kind of incorrectness problem, or the kind of problematic constraint interaction vary. Figure 1d presents an inconsistency problem, caused by the interaction of the disjoint generalization-set constraint, and the multiple inheritance of class  $C3$ : The disjoint constraint forces class  $C3$  to be empty in every legal instance. Figures 1a, 1b, 1c present three cases of the finite satisfiability problem, caused by different kinds of constraint interaction. In Figure 1a each instance of  $C$  has a single successor and at least two predecessors. Therefore, if the number of  $C$ -s in a legal instance is  $c$ , and the number of predecessor-successor links is  $d$ , then  $d$  must satisfy  $d = c \cdot 1$  and also  $d \geq c \cdot 2$ , implying the inequality  $c \geq c \cdot 2$ , that can be satisfied only by an empty or an infinite extension of class  $C$ . The problematic constraint interaction in this case involves the multiplicity constraints on the *predecessor* – *successor* association. In Figure 1b the problematic constraint interaction involves the multiplicity constraints in the cycle of associations  $w, q, r$ , and in Figure 1c the problem is with the class hierarchy between  $C1$  and  $C$ , and the multiplicity constraints on the *parent* – *child* association.

Figure 1 shows that correctness problems are varied and can occur for many reasons. We argue that in real class diagrams it is not easy to understand the various interactions among constraints, and their impact on correctness or quality. Awareness to patterns that single out problematic constraint interactions, analyze the problems they create, and suggest possible repairs, improves the overall design quality.

[BGU10] present a catalog of correctness patterns, that sort out different templates of interac-



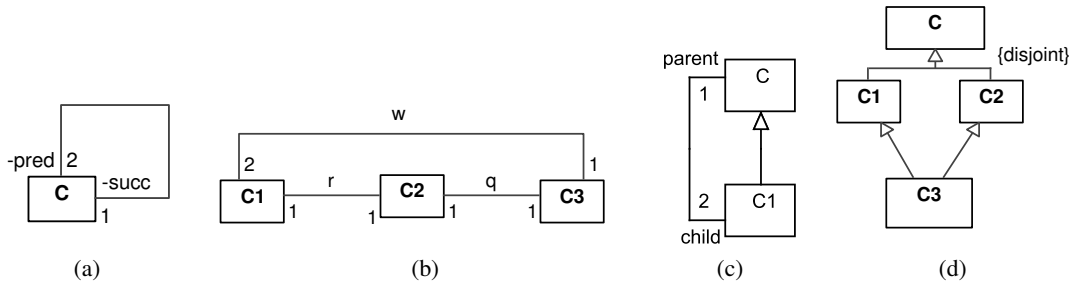


Figure 1: Finite satisfiability problems

tions, characterize the involved problems, and suggest solutions. Each pattern describes a design problem that is identified by characteristic structures within a class diagram [BMS10].

The class diagrams in Figures 1a, 1b are instances of the *Pure Multiplicity Cycle (PMC)* pattern, which characterizes finite satisfiability problems due to interaction of multiplicity constraints on a cycle of associations. Figure 2a informally sketches the identification template of this pattern. The dashed line indicates a sequence of successive binary associations (the binary associations path). The pattern includes an additional verification constraint, and analysis of possible repairs, like relaxing the multiplicity constraint “2” to “1..2”.

The class diagram in Figure 1c is an instance of the *Multiplicity Hierarchy Cycle (MHC)* pattern, which characterizes finite satisfiability problems due to interaction of multiplicity constraints on a cycle of associations and class hierarchy constraints. Figure 2b informally sketches the identification template of this pattern. The two dashed lines indicate an interleaved path of associations and class hierarchy constraints. This pattern also includes an additional verification constraint, and analysis of possible repairs, like switching the direction of a class hierarchy constraint in the cycle.

The class diagram in Figure 1d is an instance of the *diamond* pattern, which characterizes consistency problems due to interaction between multiple class hierarchy (multiple inheritance) and disjoint constraints. Figure 2c informally sketches the identification template of this pattern. A possible repair is to remove the disjoint constraint.

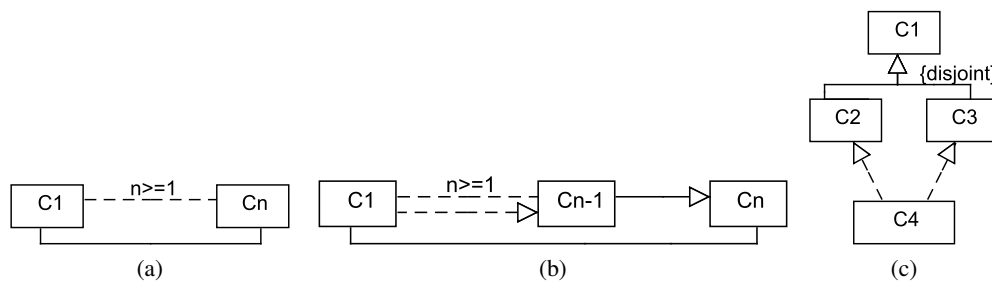


Figure 2: Informal sketches of identification templates of correctness patterns

### Correctness patterns as an educational instrument

Using positive examples (like design patterns) or negative examples (like anti-patterns) is an effective educational tool for developing high modeling skills. [BSV09] show that positive exam-

ples enhance syntactic quality, while negative examples enhance semantics equality, and neither has much effect on pragmatic quality. While there is an educational value in presenting concrete examples, patterns provide an abstraction level that classifies the concrete cases along typical characterizations. [TB11] show the influence of design patterns on improving the inception of models (although their patterns are presented by concrete syntactic examples).

Correctness pattern abstraction characterizes correctness problems in terms of conceptual structures such as *cycle of associations* and *class hierarchy cycle*, rather than in terms of concrete examples. Furthermore, patterns provide an accurate specification of *problem domains*, and solution advices. They can be viewed as *anti-patterns* that point to negative designs and suggest repairs. The benefit is twofold: First, they provide a systematic approach for identifying pattern instances; second, they sharpen the distinction between different kinds of problematic constraint interaction.

### 3 Model-Pattern Specification Language

Pattern specifications include desired and undesired structures of elements from the pattern context. Writing a specification demands a language. Pattern writing approaches can be categorized by two factors: 1) textual notation vs. visual notation (or both); 2) expression at the model-level vs. expression at the Meta-Model-level. The main mode of usage for patterns is as expert advice for educational purposes. Formally specified patterns are also used as automatic transformations (refactorings). It seems that the appropriate mode for pattern expression depends on the intended usage.

**Visual vs. textual specification:** Visual notations are uniquely human-oriented representations, that facilitate human communication, comprehension and problem solving [Moo09]. In contrast, textual representation might provide a better support for rigorous reasoning but is inferior with respect to user comprehension [Kim07]. Indeed, pattern specifications usually employ some kind of visual representations, usually in an informal manner, using concrete examples. Good visual representations enjoy the *cognitive effectiveness* property, i.e., the ability to directly clarify translations between cognitive and visual concepts [FDCB10]. This property sets a criterion for visual language evaluation [Moo09].

**Model-level vs. Meta-model-level specification:** In the model-level approach for pattern specification, patterns are specified using typical examples, that are enhanced with textual specification [GHJV95, LP09]. This is the more popular approach, especially in software design patterns. Model-level specification proved helpful for communicating design experience to developers. Yet, since structures are expressed via examples and text, it can create ambiguities that make it difficult to verify conformance to patterns [KE07]. For example, Figure 1a does not capture the intentioned problem domain, i.e., “finite satisfiability problem due to a cycle of associations with multiplicity constraints” in a way that enables identifying it with the class diagram in Figure 1b (which belongs to the same problem domain). This approach does not lend itself to automation of reasoning about patterns.

Meta-Model based specification enables rigorous textual specification of problem domains that abstract concrete examples [EBL11, EBL10]. The two examples in Figures 1a and 1b can be captured by a single textual specification, demonstrated in Listing 1 in Section 4. Therefore,

meta-model-level specification can support pattern automation as refactorings. However, it is inappropriate for educational purposes, since defining and comprehending the meta-model-level textual patterns demand expert knowledge.

**Conclusion:** We argue that for educational purposes, patterns should have **visually**, hence **model-level** specification, but use a notation that captures the **meta-model-level abstractions**. For that purpose, the model-level visual language should be enhanced with new notation that enables visualization of the meta-level abstractions. Below, we describe few extended notations to the UML class diagrams, that enables visual, model-level specification of the correctness patterns in our catalog. A similar approach is used in [BBC08a].

**Class diagram enhancement for pattern specification:** The necessary abstraction involves constructs for specification of unbounded relationship structures like association paths, hierarchy paths, interleaved association and hierarchy paths, aggregation paths, etc. We extend the UML class diagram meta-model with new classes that capture these abstractions, and provide their concrete syntax as new visual notations in class diagrams.

Figure 3b presents the Meta-Model extension for the *generalization path* abstraction. The enhancement includes a new meta-class *GeneralizationPath*, and a derived meta-association *nextassoc*. Existing meta-model elements appear within dashed rectangles. The new visual notation for relationship paths uses \* for denoting paths of lengths  $\geq 0$ , and + for paths whose lengths  $> 0$ . These symbols are added as labels, on top of the standard relationship symbol. For example, a class hierarchy (generalization) path is represented by a generalization line, labeled by \*. Figure 3a presents the identifying structure for the *Multiplicity-Hierarchy-Cycle* pattern, that uses the new generalization-path construct. The concrete syntax enhancement is demonstrated in Table 1. Figure 4 intuitively sketches a possible instantiation of the identification structure of the *Pure-Multiplicity-cycle* pattern.

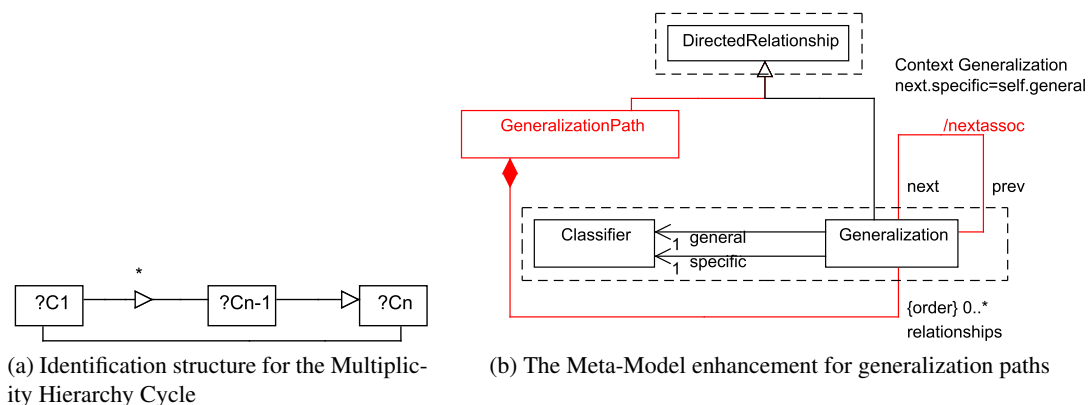


Figure 3

[Moo09] formulates nine evidence-based principles for achieving cognitively effective visual notations. Three main principles are: 1) *Semiotic Clarity principle* – importance of one-to-one correspondence between semantic constructs and graphical symbols; 2) *Semantic Transparency principle* – are symbols and their corresponding concepts are easily associated; 3) *Perceptual discriminability* – symbols that represent different constructs should be clearly distinguishable. We try to follow such principles in adopting new notation. In particular, the use of \*,+ labeled

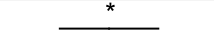

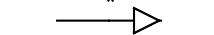

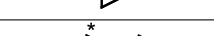
Graphic notation	Meaning
	A path of associations with length is $\geq 0$
	A path of compositions (aggregation) with length $\geq 1$
	A path of generalizations with length $\geq 0$
	An interleaved path of associations and generalizations with length $\geq 0$
	An interleaved path of compositions and generalizations with length $\geq 0$

Table 1: Concrete syntax for relationship paths

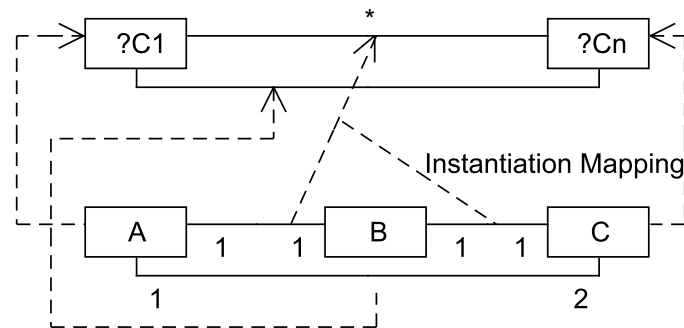


Figure 4: Instantiation of the Pure-Multiplicity-Cycle pattern

relationship lines is associated with the traditional meaning of these operators as denoting  $\geq 0$  and  $> 0$  repetitions, respectively.

## 4 The Correctness-Pattern Catalog

The catalog includes patterns for solving problems of *correctness* or of *quality* of class diagrams. The correctness problems refer to the two formal correctness problems of consistency and finite satisfiability. Quality problems refer to formally correct design problems that do not meet criteria of desirable design. The quality problems are further classified into *incomplete design*, *redundancy problems*, and *comprehension problems*. Within the categories, patterns are classified by the kind of the constraint interaction that causes the problem. This problem based classification is contrasted with the approach of [EBL10], which is syntax-semantics-pragmatics based.

Based on the above classification the catalog currently includes a total of 45 patterns: 15 patterns for finite satisfiability problems, 11 patterns for *consistency problems* and 17 patterns for *quality* problems. The catalog is still under development, and new patterns are being added.

**Pattern structure:** Pattern specification is a template consisting of nine entries: 1. Name; 2. Pattern problem – A textual description of the problem handled by the pattern; 3. Concrete example; 4. Pattern identification structure – A class diagram snippet in the enhanced class diagram language; 5. Involved meta-model elements; 6. Pattern verification – A formal constraint imposed on the pattern identification structure that verifies occurrence of a problem; 7. Repair

advice (refactoring); 8. Related patterns; 9. Pattern justification – a correctness proof for the pattern identification, verification, and advice.

*Example 1 (Pure-Multiplicity-Cycle Pattern)*

1. **Pattern name:** *Pure Multiplicity Cycle (PMC)*.
2. **Problem:** A cycle of associations with multiplicity constraints might introduce a finite satisfiability problem.
3. **Concrete Example:** See Figure 5a.
4. **Pattern Identification Structure:** See Figure 5b.

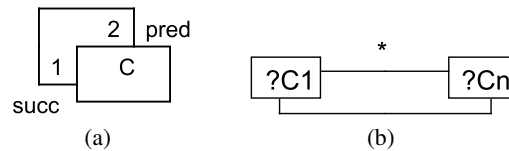


Figure 5

5. **Involved meta-model elements:** The meta-classes *Association* and *Class*.
6. **Pattern verification:** The pattern identification structure characterizes a necessary but not sufficient condition for existence of a finite satisfiability problem caused by the multiplicity constraints in an association cycle. The verification condition below provides the sufficient condition. Its specification requires an elaboration of the identification structure, as in Figure 6.

The cycle causes a finite satisfiability problem if one of the following conditions holds:

- $\prod_{i=1}^n m'_i < \prod_{i=1}^n n_i$ .
- $\prod_{i=1}^n m_i < \prod_{i=1}^n n'_i$ .

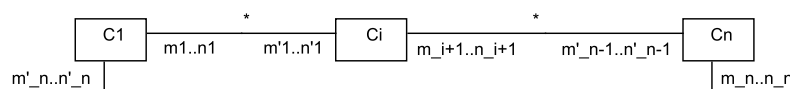


Figure 6: Pattern verification

7. **Repair advice:** Consider relaxation of the multiplicity constraints: decrease a minimal multiplicity value or increase a maximal multiplicity value.
8. **Related patterns:** The *Multiplicity Hierarchy Cycle (MHC)* pattern.
9. **Pattern justification:** Proof is omitted.

In order to emphasize the educational value of visual specification on the model level, we demonstrate, in Listing 1, the textual specification of the **PMC** identification structure in QVT, following [EBL10]<sup>1</sup>. The specification finds all associations that participate in association cycles.

<sup>1</sup> The specification is based on their *Classifier Has Generalization Cycle* anti-pattern.

```

top relation AssociationCycle {
  checkonly domain source _assoc: Association{ allconnectedassoications ->
    includes(_assoc) {} };
  enforce domain target _c: Category {name = 'Anti-Patterns', pattern = _p:
    Pattern {name = 'AssocCycle', rootBinding = _rb1: RoleBinding {role = '
    Association', element = _assoc}}};}

```

Listing 1: A QVT based rule for detecting associations that participate in association cycles

## 5 Putting the catalog into practice

In order to examine the extent to which patterns help in identifying erroneous models, we conducted a series of experiments that check various factors that affect the usage of the correctness patterns. Table 2 presents the experiment settings: The subjects' background (Software Engineering and Information Systems Engineering students) and numbers, the education type (by simple example or by formal and abstract demonstration of the patterns), the domain type (real or synthetic domains), and the size of the domain and the number of errors.

Exp	Subjects	Education Type	Domain Type	Additional Factors	Checking Mode
1	SE (49)	Examples	Real	Different domain	Class Assignment
2	ISE (49)	Examples	Synthetic	Different size	Class Assignment
3	SE (42)	SE (42)	Synthetic		Class Assignment
4	SE (61)	Patterns	Synthetic		Exam

Table 2: Experiment settings

In the first experiment we divided the subjects into two groups. In the first session (before introducing the patterns) each group got a different domain model (via a class diagram) of an academic and a genome domain. In the second session (after introducing the patterns by examples by the paper authors), each group had the other domain model. In each session the subjects were asked to identify places in which correctness problems occur. The results of the experiment show that the differences among the subjects' achievements before and after introducing the patterns are not statistically significant (see Table 3). Examining the differences across the two domains does not show any statistical significance as well. Our conjecture is that the students focused on the domain semantics rather on the model semantics.

These results have led us to perform another experiment that was based on synthetic domains. Here again, we divided the subjects into two groups, yet the two groups had domain models with different sizes (with 6 and 2 problems, respectively). We had the same setting as in the first experiment, in which the subjects had to identify existing problems before and after introducing the patterns. The results show that in the case of a large domain model with several correctness problems, there is a significant improvement in identifying the problems after introducing the patterns (see Table 3).

The third experiment followed the same setting, but the pattern introduction was more thorough, and we provided the subjects with a comprehensive explanation of the notion of correctness patterns, their formalism, abstraction and the pattern catalog. In this experiment we examined the improvement in one group and indeed, we found that the achievements made by the subjects

Exp	Before	After
1	42% (Academia); 49% (Genome);	39%(Genome); 48%(Academia)
2	28% (6 patterns) ; 52% (2 patterns)	51% (6 patterns); 50% (2 patterns)
3	42%	70%
4	Precision = 85.57%, Recall = 66.39%	

Table 3: Experiment results

were high (70 % identification of the existing problems) compared to previous experiments (see Table 3).

In the fourth examination, we checked the extent to which the patterns help in identifying only existing problems. For that purpose we calculated the recall measure which is the ability of the subjects to identify all relevant problems and the precision recall which is the ability of the subjects to identify only the relevant problems. The results (as appear in Table 3) show that the patterns indeed provide support for identifying only relevant problems. Yet, their support for identifying all relevant problems is limited.

Summarizing the results we found out that the correctness patterns indeed provide guidelines for identifying erroneous models and that their effectiveness increases in cases where the domain models are complex (and thus have more problems) and when their introduction is presented in a more formal and abstract form (instead of by examples).

**Pattern instruction:** Teaching catalogs of patterns or refactorings is hard [Pil10, Kop11]. This is a well known problem, that results from the monotonic repetitive character of catalogs. Not surprisingly, we have found through our experiments that we have been most successful when we taught only one or two patterns out of each category. The problem oriented organization of the catalog encourages a focused usage, such as when a new kind of constraint is introduced.

## 6 Conclusion

This paper presented a catalog of correctness and quality patterns for class diagram design, and discussed its role as an educational instrument. We argued that for educational purposes patterns should be visually formulated, using model-level concepts, and provided an appropriate enhancement to UML class diagram.

The correctness catalog is still under development. About 30 additional patterns are planned. In the future, we plan to include the catalog as a routine class material in our object modeling courses.

## Bibliography

- [BBC08a] D. Ballisa, A. Baruzzo, M. Comini. A Minimalist Visual Notation for Design Patterns and Antipatterns. In *Fifth International Conference on Information Technology: New Generations*. Pp. 51–56. 2008.
- [BBC08b] D. Ballisa, A. Baruzzo, M. Cominia. A Rule-based Method to Match Software Patterns Against UML Models. *Electronic Notes in Theoretical Computer Science* 219:51–66, 2008.

- [BCG05] D. Berardi, D. Calvanese, D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence* 168:70–118, 2005.
- [BGU10] BGU Modeling Group. UML Class Diagram Patterns. 2010. <http://www.cs.bgu.ac.il/~cd-patterns/>
- [BMS10] M. Balaban, A. Maraee, A. Sturm. Management of Correctness Problems in UML Class Diagrams – Towards a Pattern-Based Approach. *International Journal of Information System Modeling and Design* 1(1):24–47, 2010.
- [BSV09] N. Bolloju, C. Schneider, D. Vogel. Asymmetrical Effects of Using Positive and Negative Examples on Object Modeling. In *18th International Conference on Information Systems Development*. 2009.
- [EBL10] M. Elaasar, L. Briand, Y. Labiche. Metamodeling Anti-Patterns. 2010. <https://sites.google.com/site/metamodelingantipatterns>
- [EBL11] M. Elaasar, L. Briand, Y. Labiche. Domain-Specific Model Verification with QVT. In *ECMFA 2011*. 2011.
- [FDCB10] K. Figl, M. Derntl, M. Caeiro Rodriguez, L. Botturi. Cognitive effectiveness of visual instructional design languages. *Journal of Visual Languages & Computing*, 2010.
- [FKGS04] R. France, D. Kim, S. Ghosh, E. Song. A UML-Based Pattern Specification Technique. *IEEE Transactions on Software Engineering* 30(3):193–206, 2004.
- [GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Volume 206. Addison-wesley Reading, MA, 1995.
- [KE07] D. Kim, C. El Khawand. An approach to precisely specifying the problem domain of design patterns. *Journal of Visual Languages & Computing* 18(6):560–591, 2007.
- [Kim07] D. Kim. The Role-Based Metamodeling Language for Specifying Design Patterns. In Taibi (ed.), *Design Pattern Formalization Techniques*. Pp. 183–205. IGI Global, 2007.
- [Kop11] C. Koppe. A Pattern Language for Teaching Design Patterns. In *European conference on pattern languages of programs, EuroPLoP 2011*. 2011.
- [LP09] M. Llano, R. Pooley. UML Specification and Correction of Object-Oriented Anti-patterns. In *ICSEA '09*. Pp. 39–44. 2009.
- [MMB08] A. Maraee, V. Makarenkov, B. Balaban. Efficient Recognition and Detection of Finite Satisfiability Problems in UML Class Diagrams: Handling Constrained Generalization Sets, Qualifiers and Association Class Constraints. In *MCCM08*. 2008.
- [Moo09] D. Moody. The ”physics” of notations: towards a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering* 35(6):756 – 779, 2009.
- [Pil10] N. Pillay. Teaching Design Patterns. In *Southern African Computer Lecturers Association Conference, SACLA 2011*. 2010.
- [TB11] O. Tanriover, S. Bilgen. A framework for reviewing domain specific conceptual models. *Computer Standards & Interfaces* 33(5):448 – 464, 2011.



# Threshold Concepts in Object-Oriented Modelling

Ven Yu Sien<sup>1</sup>, David Weng Kwai Chong<sup>2</sup>

HELP University College<sup>1</sup>, Monash University Sunway Campus<sup>2</sup>, Malaysia

**Abstract:** Proponents of the object-oriented (OO) paradigm frequently claim that the OO paradigm is '*more natural*' than the procedural paradigm because the world is filled with objects that have both attributes and behaviors. However students in higher education generally experience considerable difficulty in understanding OO concepts and acquiring the necessary skills in object-oriented analysis and design. This paper proposes OO modelling to be a set of threshold concepts and describes a study that sought to improve undergraduate students' learning of OO modelling by adopting concept maps as 'stepping stones' to facilitate the development of analysis class and sequence diagrams.

**Keywords:** Threshold concept, OO modelling, Concept maps, Class diagram, Sequence diagram

## 1 Introduction

Object-oriented analysis and design (OOAD) is not easy to learn, and a particular challenge for students is developing the ability to abstract real-world problems [EMM+06a]. Despite claims made by advocates of the OO approach [ADS+00], [SBG06] novices in OO techniques in general have difficulties understanding OO concepts. Colbert [Col94] has found in his teaching and consulting experience that software developers have problems coping with the OO paradigm, especially with the concept of abstraction. When trying to conceptualise real-world problems as abstractions within the context of OO analysis and design (OOAD), he found that software developers experience particular difficulties in trying to classify real-world objects in generalisation-specialisation hierarchies or whole-part associations.

In this paper we introduce the notion of threshold concepts in OO modelling. A threshold concept or skill is one which when grasped, changes the way in which the learner views the discipline or approaches a task; i.e. it is *transformative*, and is thus likely to be *irreversible* or difficult to unlearn. This is certainly true of OO modelling. The journey across a conceptual threshold is likely to be difficult or *troublesome*, and may involve traversing the conceptual space (recursiveness) until 'the penny drops'. Difficulty may also arise from a concept being counter-intuitive or contrary to common sense. Threshold concepts are *integrative*, bringing together different aspects of a subject that previously appeared unrelated. Finally, a threshold concept delineates a particular conceptual space, i.e. it has *boundaries*, and to move beyond these boundaries the learner needs to grasp other threshold concepts.

The paper is organized as follows: Section 2 describes studies that investigated the problems novices have in understanding and developing OO models; and research on threshold concepts within computing education. In Section 3, we present our proposal on OO modelling to be a set of threshold concepts. Section 4 illustrates some of the difficulties experienced by students

in OO modelling and Section 5 provides a strategy on helping students overcome the thresholds. We conclude in Section 6 with a summary of the findings.

## 2 Background Review

### 2.1 OO Modelling

During the OOAD phases, models are produced to show the type of information processing that is required of the new system. A model of an OO system is an abstract representation of the system. It represents the problem domain and emphasises some characteristics of the real-world. Modelling a system, however, requires the representation of different perspectives or views of the system and therefore there are different types of diagrams for modelling each of these views. Models used for OO development can take the form of graphics, narratives, or formulae.

Proponents of the OO paradigm frequently claim that it is '*more natural*' than the procedural paradigm because the world is filled with objects that have both attributes and behaviors. Neubauer and Strong [NS02] assume that '*more natural*' implies more intuitive, that is, more easily understood and more consistent with existing patterns of thought. Martin [Mar93] similarly notes that we '*have a natural way of organising our knowledge about the world*' – hence, we will not find it difficult to '*analyse the world in a way that seems natural to human thinking*'.

However, although OO is representative of real-world problems, it does not always reflect the way in which people think [ADS+00], [NS02]. Rosson and Alpert [RA90] reported end-users' conceptual confusion about objects: OO analysts refer to objects which may not be considered 'natural objects' to users. For example, a 'customer' may be intuitively seen as a natural object as 'it' can perform some action. But seeing an 'order' an entity with responsibilities and ability to perform operations is likely to be counter-intuitive, possibly requiring crossing a conceptual threshold. Wirfs-Brock [Wir06] argues that it is a myth that objects in a computer system are representations of real-world entities – the domain concepts identified in the system are at '*best loosely connected to their real-world counterparts*'.

Bolloju and Leung's [BL06] study of errors produced by novices in class and diagrams reported errors to be –

- incorrect multiplicities;
- misassigned attributes;
- incorrect usage of generalisation-specialisation hierarchies;
- missing messages;
- missing objects; and
- incorrect delegation of responsibilities.

An OO system consists of interacting objects to fulfill certain responsibilities. This therefore implies that objects are 'animated' in some way. However, Neubauer and Strong [NS02] argue that we do not usually view our world as a collection of animated objects *interacting* with one another to create processes. Instead, we view our world as containing many inanimate objects that we control and manipulate. Similarly, Détienne [Det97] conducted a review of empirical

research on OO design (OOD) and discovered that the results of her research do not support the claims made for the naturalness and ease of OOD. Some of her findings are:

- The identification of objects from the problem domain is not easy. The entities that are identified may not necessarily be useful in the design solution.
- The mapping between the problem and programming domains is not easy. The analysis of the problem domain is not sufficient to structure the solution in terms of objects.

## 2.2 Threshold Concepts

The notion of threshold concepts was first presented in 2002 at the Tenth Improving Student Learning Conference in Brussels, and its conceptual framework emerged from the United Kingdom Economic and Social Research Council funded project ‘Enhancing Teaching and Learning Environments in Undergraduate Courses’ [DE05]. Seminal work within economics posited that certain concepts e.g., cost and elasticity held by economists to be central to the mastery of their discipline could be described as threshold concepts. Subsequent work has largely focussed on development of the conceptual framework and the identification of threshold concepts in various disciplines e.g.,

- complex numbers and limits in mathematics [ML03];
- confidence intervals in statistics [CB06]; and
- evolution in biology [TC07].

The following candidate threshold concepts within computing education research have been proposed:

- OO programming (OOP). This topic has been known to be both difficult to teach and to learn. Boustedt et al. [BEM+07] identified this to be a broad area within which thresholds exist. First-year students [ET05] who were interviewed after their first OOP course, reported that they found OOP to be troublesome to learn (it took a long time to ‘click’), the knowledge gained is irreversible (once understood the OO paradigm cannot be forgotten) and transformative (knowledge gained can be transferred to another programming language).
- Pointers. Boustedt et al. [BEM+07] identified this to be another threshold concept. First-year students reported in [ET05] they found pointers to be troublesome to learn (difficult to understand), the knowledge gained is irreversible (unforgettable); and integrative and transformative (knowledge gained can be used in other subjects).
- Abstraction. Eckerdal et al. [EMM+06b] identified this to be a candidate threshold concept as the ability to abstract and to be able to move from one level of abstraction to another is a key skill in computer science. In a later paper, Mostrom et al. [MBE+08] suggested that abstraction per se may not be a threshold concept – however, specific forms of abstraction e.g., modularity, data abstraction, object-orientation, etc. can be considered as threshold concepts.
- OO at its most basic – including classes, objects and encapsulation [EMM+06b]. First-year students [BEM+07] found basic OO troublesome to learn. Learning OO is transformative (it requires a change in mind-set when viewing OO as representative of real-world problems) and integrative.

- Recursion. Rountree and Rountree [RR09] proposed recursion to be a candidate threshold concept. Many novice programmers have found recursion to be an example of troublesome knowledge (they have difficulty with the concept of `self reference'). However, once the students have grasped this concept, the understanding is irreversible (new understanding cannot be unlearned); and transformative and integrative (the student is able to identify relationships with other materials e.g. all loops can be expressed as recursion). Recursion is a boundary marker for both Software Engineering and Theoretical Computer Science (it is useful in programming, and also defines what is computable).

### 3 OO Modelling: Thresholds in OOAD

Undergraduate IT students have in general found difficulty in grasping OO concepts and the role that UML diagrams play in the analysis and design phases of a software development lifecycle. It has been observed that students frequently produce class and sequence diagrams that are incomplete, with many concepts at inconsistent abstraction levels [BL06], [SBG06]. Some of these problems were discussed in Section 2. Most of the students are unable to effectively build class diagrams from the problem domain because they essentially do not know *'what'* to model.

The class diagram is fundamental to the object modelling process, representing the key concepts and relationships (e.g., generalisation-specialisation hierarchy, aggregation, composition and association) in the problem domain of the system. The class diagrams produced during the OO analysis are a logical model and shows classes of objects that are required to represent the problem domain. The sequence diagram is a kind of interaction diagram produced during the OO design phase and describes the flow of messages between objects. It is useful in describing the dynamic design, the invocation of methods, and the order of invocations.

Several aspects of OO modelling are consistent with the defining criteria for threshold concepts. The conceptual grasp of OO modelling is difficult to unlearn, it is a solid basis for effective application, and clearly distinguishes between practitioner and learner. Simultaneously, OO modelling is difficult both to teach and learn: it represents troublesome knowledge, not least because of the requirement for abstraction. Modelling is an important activity in system development: Models represent the system at different levels of abstraction. OO modelling transforms and integrates understanding of analysis and design of information systems. This transformative learning produces advanced learners who understand the complex notions underlying the development of information systems. Topics such as systems analysis and design, software engineering and advanced topics in OO programming languages are likely to have little meaning to students who do not grasp OO modelling.

It was previously discussed in Section 2.2, that OOP, abstraction and OO are broad areas within which thresholds exist. Based on data analysed from student interviews, Boustedt et al. [BEM+07] argued that more specific threshold concepts in OOP *'might include the way in which objects work together (i.e. concurrency), or the ability to see large problems as composed of a set of small sub-problems'*. Eckerdal et al. [EMM+06b] noted in their experiment that both lecturers and students considered OO as a threshold concept, but the authors argued that OO is too broad an area and some of the more specific threshold concepts could be polymorphism or object interactions.

OO modelling is also clearly a key concept within OOAD, but OO modelling may represent a family or group of associated thresholds including:

- classes;
- generalisation-specialisation hierarchies; and
- object interactions.

## 4 Difficulties in OO Modelling

This section presents some examples of the types of problems that novices face when producing UML class and sequence diagrams:

- identifying classes from functional requirements;
- assigning attributes to a class;
- selecting appropriate generalisation-specialisation hierarchies for classes; and
- assigning responsibilities to objects to realise a use case.

Participants in our study were fifty-one Year 2 IT undergraduate students enrolled in a BSc (Hons) programme in Computing at two universities in Kuala Lumpur, Malaysia. They worked on a Vehicle Rental case study containing four expanded use cases (detailed descriptions of processes) that describe the functional requirements of the system. The participants were asked to individually produce an analysis class diagram based on all the expanded use cases. Rubrics were used to describe assessment criteria for evaluating the appropriateness of diagrams, which were assessed for appropriate classes, attributes, associations and multiplicities.

### 4.1 Analysis of Class and Sequence Diagrams

The class diagram is the key artefact in the analysis phase as its *appropriateness* can have a significant impact on the design of the overall system.

#### 4.1.1 Classes

Liu et al. (2003) observed that the identification of classes is difficult even for experienced analysts and OO developers, citing three contributory factors-

- complexity, vagueness and ambiguity of natural language;
- lack of the domain knowledge and OO experience; and
- the absence of effective OO methods and well-developed guidelines.

Expected Classes. Table 1 shows the number of *expected classes* that participants were able to identify in their class diagrams. Only 8% identified all eight expected candidate classes, 35% identified seven classes and 2% identify only one appropriate class.

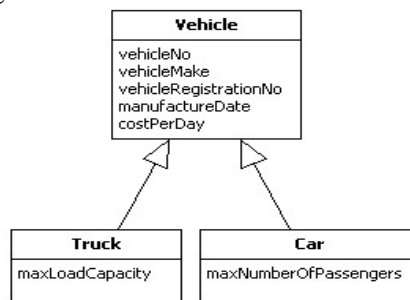
**Table 1. Analysis of class diagrams with expected classes**

Classes <sup>1</sup>							
1	2	3	4	5	6	7	8
2%	2%	8%	14%	10%	22%	35%	8%

<sup>1</sup> Total is not 100% because of rounding errors.

#### 4.1.3 Generalisation-Specialisation Hierarchies

The role of the generalisation-specialisation hierarchy is usually explained by showing examples of its use in models that feature common classification hierarchies e.g. Employee, PartTimeEmployee and FullTimeEmployee. While this approach seems simple, novices usually find defining hierarchies difficult. Detienne [Det97] finds that novices try to use inheritance as often as they can, but frequently use it incorrectly, especially when identifying the static characteristics (attributes) of the superclass and subclasses. She also finds that novices tend to use the generalisation-specialisation hierarchy to express a whole-part association. In our study we expected participant to identify a generalisation-specialisation hierarchy as exemplified in Fig. 1.



**Figure 1. Appropriate generalisation-specialisation hierarchy**

We found that participants' class diagrams did not include many generalisation-specialisation hierarchies (Table 2). Only 45% of participants defined appropriate inheritance hierarchies in their class diagrams; and 6% had a mixture of appropriate and inappropriate hierarchies.

**Table 2. Analysis of class diagrams with generalisation-specialisation hierarchies**

<b>Inappropriate Use of Inheritance Hierarchy</b>	<b>Appropriate Use of Inheritance Hierarchy</b>	<b>Inheritance Hierarchies not Defined</b>
12%	45%	49%

#### 4.1.4 Object Interactions

Students generally have difficulty identifying messages to be sent in UML sequence diagrams. They do not know how to fulfil the responsibilities of the use case by getting objects to pass messages to each other. Students also have difficulty understanding that the interaction diagrams are dependent on the analysis class diagram in terms of its classes, associations and multiplicities.

A sequence diagram focuses on the time ordering in which messages are sent. It is useful for describing the order of invocation of methods. Only 40% of sequence diagrams displayed some evidence of responsibilities delegated to the appropriate objects. None of the student diagrams fulfilled all the responsibilities of the use case. 52% of sequence diagrams did not include any parameters in the messages – we do not however consider this a serious design fault as parameters can be optionally defined in UML.

## 4.2 Discussion

*'Completeness of a design is concerned with the fact that the presence of information in some diagram requires the presence of other information in another part of the design. For instance,*

*if there is a use-case that describes some system functionality, then there should also be a collection of classes that provides this functionality. If some information that we can deduce from available diagrams is not present in a design, then there is an incompleteness in the design'* [LCM+03]. By description models produced by our study participants are incomplete because there are missing elements in the class and sequence diagrams that have not been defined to represent the requirements of the system.

## **5 Evaluation: Changes in Levels of Understanding**

### **5.1 Concept Mapping**

Concept mapping, a tool for facilitating learning, was developed by Joseph Novak [NC06] at Cornell University in 1972, and is commonly used for visualising relationships between concepts. Within the context of OOAD we use concept maps to graphically present fundamental concepts and their inter-relationships within a problem domain.

Our study (Section 4) initially investigated the difficulties undergraduate students have when producing UML class and sequence diagrams. The results indicate the fundamental problems that learners have with OO modelling, and suggest that learners have difficulty with certain concepts which may represent thresholds in understanding. In order to address these threshold concepts, a concept-driven approach is developed to help novices produce more appropriate UML class and sequence diagrams. The effectiveness of this approach is evaluated by three different experiments.

A static concept map diagrammatically describes the structure of a system by illustrating its component concepts and their inter-relationships: The concepts thus defined model classes and attributes in an analysis class diagram. A static concept map is constructed by identifying concepts and their relationships from expanded use cases, and is built incrementally from the use cases. Rules for producing a static concept map and transforming it to a class diagram are defined in [SC07].

A dynamic concept map provides a dynamic view of the system behaviour by showing the key responsibilities that need to be fulfilled by specific concepts in order to fulfil a particular scenario of a use case. The concepts defined in the dynamic concept map model *objects* in the sequence diagram. For each use case, its key responsibilities are identified and added to the static concept map so as to produce a dynamic concept map. Rules for producing a static concept map and transforming it to a class diagram are defined in [SC08].

### **5.2 Helping Students Through Thresholds**

Two studies [Sie10] have reported the effects of using concept mapping to assist learners in OOAD produce class and sequence diagrams. Some findings of the first study ('Study 1'), including common faults found in UML diagrams, are described in Section 4; participants in this study were not taught any concept mapping techniques. In Study 2 [Sie10] twenty-one Year 2 IT undergraduate students were taught the concept mapping techniques and this study found a statistically significant reduction in the number of faults produced in UML class and sequence diagrams, particularly in terms of:

- identification of expected classes representing the key concepts in the problem domain;
- assignment of attributes to appropriate classes;

- identification of appropriate generalisation-specialisation hierarchies; and
- assignment of responsibilities to fulfil a particular scenario of a use case.

However the results achieved by Study 2 participants may not be attributed solely to the effect of concept mapping – other contributory factors to quality improvement include:

- As the students in Study 2 were currently enrolled in an OOAD course, their knowledge of OOAD concepts and experience in OO modelling was likely to be fresh in their minds.
- The students in Study 2 may have been given a better foundation on OOAD concepts.
- Students in Study 2 were given more time to produce the concept maps and UML models.

Participants responded positively to the use of concept maps: This is consistent with research [Roy08] reporting the successful use of concept maps in teaching. The results presented in [Sie10] lead us to conclude that concept mapping has a positive impact on class and sequence diagram, and suggest that concept mapping effectively helps learners understand OO modelling. In particular the use of specifically defined labelled links (in the static concept maps) helps learners distinguish between classes and attributes, and more easily identifies relationship types. Other particular benefits are that

- mapping is relatively easy to teach; the two types of notations solely used are nodes and links [NC06];
- maps help clarify the meaning of concepts using propositions [NC06];
- substantial guidelines for producing concept maps have been developed. These are defined in [SC07], [SC08].

## 6 Conclusion

In this paper we have proposed the following topics to be considered threshold concepts in OO modelling:

- **Classes.** In general, students find this topic to be troublesome (the students experience difficulties in identify appropriate classes from the problem domain; and assigning appropriate attributes to classes). The knowledge gained is irreversible (once understood, it cannot be easily forgotten) and transformative and integrative (knowledge gained can assist in developing the logical design of databases).
- **Generalisation-specialisation hierarchies.** Novices usually find defining hierarchies troublesome (they have problems grouping real-world objects in terms of classification because they are not used to grouping objects in hierarchies). This is not intuitive enough to be mastered without training [RA96]. The knowledge gained is irreversible (once the concept is understood, students will invariably find it easier to identify generalisation-hierarchies). This is transformative and integrative as this hierarchy can be used in use case diagrams and data base models.
- **Object interactions.** Svetinovic et al. [SBG06] found some common errors committed by students when producing interaction diagrams:
  - assignment of a large business activity to a single object whilst it should be fulfilled through the collaboration with other objects;
  - missing responsibilities that should be assigned to objects; and
  - missing objects that should participate in the overall responsibilities.



This topic has been known to be both difficult to teach and to learn. Students in general have found this topic to be troublesome to learn (difficult to understand), the knowledge gained is irreversible (unforgettable); and integrative and transformative (knowledge gained can be used in OOP).

Meyer and Land [ML03] suggest that once a student has been introduced to a threshold concept, he/she enters a state of *'liminality'* – a state associated with being 'stuck' and not possessing a mastery of the concept – until the necessary transformation of understanding has taken place and the 'threshold' is crossed. Students who have problems producing analysis class diagrams will first need to cross the threshold in identifying appropriate classes (representing real-world objects) before they can assign attributes and relationships (e.g. associations, generalisation-specialisation hierarchies and whole-part hierarchies) to the classes. These thresholds have to be crossed before the students can successfully produce appropriate sequence diagrams.

Studies have been presented to support the adoption of concept maps as aids to novices for developing class diagrams. Participants in experiments conducted by the first author showed significant improvement in class diagram construction (after the participants have been taught the concept mapping techniques), in terms of identification of appropriate classes, generalisation-specialisation hierarchies, and the appropriate assignment of responsibilities to objects. When learners have made some progress in OO modelling they are likely to find drawing concept maps tedious and time-consuming. These techniques are likely therefore to be important in facilitating conceptual understanding – crossing the threshold of OO modelling.

## Bibliography

- [ADS+00] R Agarwal, P De, AP Sinha, M Tanniru. 'On the Usability of OO Representations. Communications of the ACM 43(10), pp. 83-89, 2000.
- [BEM+07] J Boustedt, A Eckerdal, R McCartney, JE Mostrom, M Ratcliffe, K Sanders, C Zander. Threshold concepts in computer science: do they exist and are they useful? In *Proc. 38<sup>th</sup> SIGCSE Technical Symposium on Computer Science Education*. 2007.
- [BL06] N Bolloju & F Leung. Assisting Novice Analysts in Developing Quality Conceptual Models with UML. Communications of the ACM 49(7), pp. 108-112, 2006.
- [Col94] E Colbert. Abstract Better and Enjoy Life. *Journal of Object-Oriented Programming (JOOP)*, 7(1), 1994.
- [CB06] CJ Cope & G Byrne. Improving Teaching and Learning about Threshold Concepts: The example of Confidence Intervals. In *Proc. Threshold Concepts within the Disciplines Symposium*. 2006.
- [DE05] D Hounsell & Noel Entwistle. Enhancing Teaching-Learning Environments in Undergraduate Courses. <http://www.etl.tla.ed.ac.uk/docs/ETLfinalreport.pdf>
- [Det97] F Détienné. Assessing the Cognitive Consequences of the Object-Oriented Approach: A Survey of Empirical Research on Object-Oriented Design by Individuals and Teams. *Interacting with Computers*, 9(1), pp. 47-72, 1997.
- [EMM+06a] A Eckerdal, R McCartney, JE Moström, M Ratcliffe, C Zander. Can Graduating Students Design Software Systems? In *Proc. 37<sup>th</sup> SIGCSE Technical Symposium on Computer Science Education*. 2006.
- [EMM+06b] A Eckerdal, R McCartney, JE Moström, M Ratcliffe, K Sanders, C Zander. Putting Threshold Concepts into Context in Computer Science Education. In *Proc. 11<sup>th</sup>*

- Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. 2006.
- [ET05] A Eckerdal, M Thuné . Novice Java Programmers' Conceptions of "object" and "class", and Variation Theory. In *Proc. 10<sup>th</sup> Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*. 2005.
- [LCM+03] C Lange, MRV Chaudron, J Muskens, LJ Somers, HM Dortmans. An Empirical Investigation in Quantifying Inconsistency and Incompleteness of UML Designs. In *Proc. 2<sup>nd</sup> Workshop on Consistency Problems in UML-Based Software Development*. 2003.
- [Mar93] J Martin. *Principles of Object-Oriented Analysis and Design*. Prentice-Hall International, Inc., Englewood Cliffs, New Jersey.
- [MBE+08] JE Moström, J Boustedt, A Eckerdal, R McCartney, Sanders K, L Thomas, C Zander. Concrete Examples of Abstraction as Manifested in Students' Transformative Experiences. In *Proc. 4<sup>th</sup> International Workshop on Computing Education Research*. 2008.
- [ML03] JHF Meyer, R Land. Threshold Concepts and Troublesome Knowledge (1): Linkages to Thinking and Practising within the Disciplines. *Improving Student Learning: Improving Student Learning Theory and Practice - Ten Years On*. Oxford: OCSLD, pp. 412-424, 2003.
- [NC06] JD Novak, AJ Cañas. The Origins of the Concept Mapping Tool and the Continuing Evolution of the Tool. *Information Visualization* 5(3), pp.175-184, 2006.
- [NS02] BJ Neubauer & DD Strong. The Object-Oriented Paradigm: More Natural or Less Familiar? *J. Comput. Small Coll.*, 18 (1), pp. 280-289, 2002.
- [RA90] MB Rosson & SR Alpert. The Cognitive Consequences of Object-Oriented Design. *Human-Computer Interaction*, vol. 5, 1998.
- [RA96] C Ryan & G Al-Qaimari. A Cognitive Perspective on Teaching Object Oriented Analysis and Design.  
<http://citeseer.ist.psu.edu/cache/papers/cs/3179/http://zSzzSzgoanna.cs.rmit.edu.auzSz~ghassanzSzrmit.pdf/a-cognitive-perspective-on.pdf>
- [RR09] J Rountree, N Rountree. Issues Regarding Threshold Concepts in Computer Science. In *Proc. 11<sup>th</sup> Australasian Conference on Computing Education*. 2009.
- [Roy08] D Roy. Using Concept Maps for Information Conceptualization and Schematization in Technical Reading and Writing Courses: A Case Study for Computer Science Majors in Japan. In *Proc. IEEE International Professional Communication Conference*, 2008.
- [SBG06] D Svetinovic, DM Berry, MW Godfrey. Increasing Quality of Conceptual Models: Is Object-Oriented Analysis that Simple? In *Proc. 2006 International Workshop on Role of Abstraction in Software Engineering*. 2006.
- [SC07] VY Sien, D Carrington. A Concepts-First Approach to Object-Oriented Modelling. In *Proc. Third IASTED International Conf on Advances in Computer Science and Technology*. 2007.
- [SC08] VY Sien, D Carrington. Using Concept Maps to Produce Sequence Diagrams. In *Proc. IASTED International Conference on Software Engineering*. 2008.
- [Sie10] VY Sien, Teaching Object-Oriented Modelling using Concept Maps. In *Proc. 6th Educators' Symposium: Software Modeling in Education at MODELS 2010*.
- [TC07] C Taylor & CJ Cope. Are There Educationally Critical Aspects in the Concept of Evolution? In *Proc. of UniServe*, 2007.
- [Wir06] RJ Wirfs-Brock. Looking for Powerful Abstractions [Object Oriented Technology]. *Software, IEEE*, 23(1), pp. 13-15, 2006.

# Teaching Modeling—An Initial Classification of Related Issues

Ludwik Kuzniarz, Jürgen Börstler

[{ludwik.kuzniarz,jurgen.borstler}@bth.se](mailto:{ludwik.kuzniarz,jurgen.borstler}@bth.se)

School of Computing

Blekinge Institute of Technology, Karlskrona, Sweden

**Abstract:** Modeling is an important skill needed in both science and engineering domains. In software engineering, in particular, models are ubiquitous artefacts. The development, manipulation and understanding of models is therefore an important learning objective. The paper presents the initial results of an attempt that has been carried out in order to classify issues related to the teaching and learning of modeling.

**Keywords:** Modeling, Teaching, Classification

## 1 Introduction

Constructing and using models is an essential element of any constructive human activity. Models are used for understanding the world around us, as well as for creating new things.

Modeling is therefore an important skill needed in all science and engineering domains. Solving complex problems requires knowledge and skills in modeling, to be able to focus on the properties that are most relevant in a specific context without being distracted by details that are irrelevant at a certain level.

In software engineering, in particular, models are ubiquitous artefacts. The development, manipulation and understanding of models is therefore an important learning objective. However, graduating students have problems with designing even small software systems [EMM<sup>+</sup>06, LTZ11] and “underestimate the importance of representing structural groupings and interactions between design parts” [TFB<sup>+</sup>05].

Education related to software engineering, and teaching modeling in particular, is getting increasing attention and understanding in the academic community. The related problems are highlighted and discussed on fora such as software engineering education related symposia and workshops. We attempt to provide a classification of issues, problems, and challenges related to teaching and learning modeling. We claim that the classification will help to better understand the problems related to teaching modeling, as well as to identify and position existing research in the area and in consequence to contribute in improving the state-of-the-art of modeling education.

The present paper reports on the first phase of developing such a classification – the elicitation process and its results, and presents the plans for further research towards a mature and approved classification framework.

## 2 Related Work

There is a large body of work related to a number of aspects relevant for the teaching of modeling. Virtually no work, though, is concerned with categorizations or taxonomies of teaching issues. Various issues regarding the teaching of modeling were discussed and summarized in earlier MoDELS panels (see for example [BFG<sup>+</sup>10]), but the results were not categorized in a systematic way.

Diethelm et al and Schulte and Niere show that UML-like modeling can be successfully introduced in secondary school and helps students grasp basic object-oriented concepts [DGZ05, SN02]. They especially observe that the models the students created helped them to discuss each other's modeling problems.

A general problem of teaching models and modeling is the lack of a commonly accepted definition of model quality. Nelson et al propose a conceptual modeling quality framework where they integrate product and process views from earlier frameworks [NPGP11].

Empirical research shows that there are various aspects of models that contribute to their understanding, like decomposition quality [BJ08], diagram layout [SW05], naming [BLMM09], and actual modeling language used [GPS05]. It is therefore important to be aware that there is no “one-size-fits-all” solution to improve model comprehensibility.

Kuzniarz and Staron propose a number of best practices for teaching modeling/UML and particularly emphasize the role of consistency between models and iterative teaching [KS06]. The role of model consistency is also highlighted in StudentUML, a teaching tool for UML diagrams that ensures consistency between class and sequence diagrams [RD07]. Brandscheidl et al discuss a large scale modeling course with specific emphasis on assessment and propose different types of exercises for assessing theoretical and practical knowledge and skill in UML modeling [BSK09].

A good source for teaching or learning issues are typical problems or errors appearing in students' models or designs. Thomasson et al found that many students failed to integrate some classes into their solutions, i.e. had problems of connecting the elements of a model to a meaningful “big picture” [TRT06]. Another common problem was attribute location or representation, leading to classes with low cohesion. In another empirical study, Siau and Loo showed that the problems in learning UML mainly can be grouped into two main categories: (1) problems inherent in UML, like its complexity and semantic issues, and (2) “peripheral” issues, like the learners' lacking prerequisites or problems with teaching materials [SL06].

There are also numerous modeling/UML tools<sup>1</sup>, but few of them are specifically designed for teaching. Those tools rarely support easy mechanisms for consistency checking and most of them have a very steep learning curve. Modern learning environments support features for collaboratively discussing modeling exercises and solution [BBB<sup>+</sup>10]. Besides reaching more learners, these tools help students understand that there is no single correct solution to a modeling problem.

---

<sup>1</sup> The list at <http://www.jeckle.de/umltools.htm> contains more than 100 UML tools and doesn't even include educational tools like StudentUML [RD07] or Violet (<http://violet.sourceforge.net>).

### 3 Initial Classification

A general observation regarding challenges and issues related to the teaching of modeling, is that some of them share a common denominator, a specific aspect of or a point of view on the teaching of modeling. We therefore propose a two level classification structure with the points of view, or perspectives on the top level, and the specific issues on the bottom level (following the vocabulary used for a similar purpose in [KA11]).

We observed that the perspectives could be characterized by question words and the specific issues could be elicited by formulating questions around those words in the following way:

**why to teach** – concerning the reasons for introducing the modeling into the teaching curricula,

Table 1: Initial classification scheme.

Perspectives	Issues	
<b>WHY:</b>  Why should modeling be included in a curriculum?	Way of thinking	to encourage and stimulate thinking at high abstraction levels
	Problem solving	to enable solving problems at higher abstraction level
	Successful research	to enable doing successful research
	Successful development	to enable and ensure successful development of software
	Being up-to-date	to be up to date with recent trends and developments in SE
	Being competitive	to be competitive in the labour market
<b>WHAT:</b>  What should be included in a modeling curriculum?	Creating models	knowledge and skills needed for the creation of models
	Using models	where and how models can be used
	Informal models	using sketchy drawings to convey basic characteristics or ideas
	Formal models	building models that conform to rigorous formal rules
	Integration of models	mixing and merging different models within a project
	Transformation	model transformations
	Code generation	code generation from models
	Languages	modeling languages and their proper usage
	Tools	modeling tools and environments that support modeling
	Best practices	proved approaches used in modeling
	Basic principles	commonly accepted modeling principles
Consistency	relationships between different (views of) models	
<b>HOW:</b>  How should modeling be taught?	Examples	using good examples
	Exercises	actual problem solving
	Projects	carrying out projects of different size
	Industrial practices	reflecting practices used in industry
	Industry lectures	guest lectures from industry
	Communication	communicating with models
	Presentations	encouraging student presentations
	Discussions	encouraging discussions between students
Teaching methods	using specific, dedicated teaching methods and processes	

**what to teach** – concerning the contents of the teaching,

**how to teach** – concerning the ways and means used for teaching.

The **why**, **what** and **how** perspectives were the starting point for an initial elicitation of issues and challenges based on (a) a structured panel discussion on the 8th Nordic Workshop on Model Driven Software Engineering and (b) interviews with some domain experts. The panelists and the approached experts were members of academia, who were involved in teaching different subjects related to software engineering that included elements of modeling. Both groups were asked the same questions, aimed at finding specific issues in the identified perspectives. The actual questions were the following:

1. Why do you think we should teach modeling?
2. What should be taught?
3. How we should teach modeling?

The questions represented the identified perspectives. Based on the answers specific issues in the perspectives were derived, named and described. The resulting initial classification is shown in Table 1.

In addition to our three initial perspectives, the initial elicitation revealed two additional perspectives that need to be further investigated:

**where to teach** – concerning the place where the education takes place (e.g., university, on-the-job training etc.), and

**when to teach** – concerning the time and place in a curriculum (in relation to other subjects).

## 4 Summary and Future Work

We described the first step of a project aimed at providing a justified and empirically evaluated classification of the issues and challenges related to the teaching and learning of modeling in the context of software development, together with the obtained results from that step.

After analysis of the data from the survey, we found that several details of the classification, concerning both perspectives and issues, need to be further investigated and improved.

In the next step, our goal is to develop an empirically justified classification scheme by targeting a wider population of experts involved in the teaching of modeling. The updated classification will then be extended with empirical data on relative prioritizations of the classified issues and challenges, using a similar approach as in our classification of consistency problems in model driven software development [KA11].

## Bibliography

- [BBB<sup>+</sup>10] J. Börstler, O. F. Bay, M. Baturay, S. Trapp, M. Heintz, S. Weber. embed4Auto: a PLE for software modelling. In *Proceedings of the 15th annual conference on Innovation and technology in computer science education*. P. 322. 2010.
- [BFG<sup>+</sup>10] J. Bezivin, R. France, M. Gogolla, O. Haugen, G. Taentzer, D. Varro. Teaching Modeling: Why, When, What? In *MODELS 2009 Workshops*. Pp. 55–62. Springer, 2010.

- [BLMM09] D. Binkley, D. Lawrie, S. Maex, C. Morrell. Identifier length and limited programmer memory. *Science of Computer Programming* 74(7):430–445, 2009.
- [BSK09] M. Brandsteidl, M. Seidl, G. Kappel. Teaching Models @ BIG: On Efficiently Assessing Modeling Concepts. In *Proceedings of the MoDELS 2009 Educators' Symposium*. 2009.
- [BJ08] A. Burton-Jones. The Effects of Decomposition Quality and Multiple Forms of Information on Novices' Understanding of a Domain from a Conceptual Model. *Journal of the Association for Information Systems* 9(12):748–802, 2008.
- [DGZ05] I. Diethelm, L. Geiger, A. Zündorf. Teaching Modeling with Objects First. In *Proceedings of the 8th World Conference on Computers in Education*. 2005.
- [EMM<sup>+</sup>06] A. Eckerdal, R. McCartney, J. E. Moström, M. Ratcliffe, C. Zander. Can graduating students design software systems? In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. Pp. 403–407. 2006.
- [GPS05] G. Guizzardi, F. Pires, M. van Sinderen. An ontology-based approach for evaluating the domain appropriateness and comprehensibility appropriateness of modeling languages. In *Proceedings MoDELS 2005*. Lecture Notes in Computer Science 3713, pp. 691–705. Springer, 2005.
- [KA11] L. Kuzniarz, L. Angelis. Empirical extension of a classification framework for addressing consistency in model based development. *Information and Software Technology* 53(3):214–229, 2011.
- [KS06] L. Kuzniarz, M. Staron. Best Practices for Teaching UML-based Software Development. In *Satellite Events at the MoDELS 2005 Conference*. Pp. 320–332. 2006.
- [LTZ11] C. Loftus, L. Thomas, C. Zander. Can graduating students design: revisited. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. Pp. 105–110. 2011.
- [NPGP11] H. J. Nelson, G. Poels, M. Genero, M. Piattini. A Conceptual Modeling Quality Framework. *Software Quality Journal*, pp. 1–28–28, Apr. 2011.
- [RD07] E. Ramollari, D. Dranidis. StudentUML: An Educational Tool Supporting Object-Oriented Analysis and Design. In *Proceedings of the 11th Panhellenic Conf. on Informatics (PCI 2007)*. 2007.
- [SL06] K. Siau, P. Loo. Identifying Difficulties in Learning UML. *Information Systems Management* 23(3):43–51, 2006.
- [SN02] C. Schulte, J. Niere. Thinking in Object Structures: Teaching Modelling in Secondary Schools. In *ECOOP Workshop on Pedagogies and Tools for the Learning of Object-Oriented Concepts*. 2002.

- [SW05] D. Sun, K. Wong. On Evaluating the Layout of UML Class Diagrams for Program Comprehension. In *Proceedings of the 13th International Workshop on Program Comprehension*. Pp. 317–326. 2005.
- [TFB<sup>+</sup>05] J. Tenenberg, S. Fincher, K. Blaha, D. Bouvier, T. Chen, D. Chinn, S. Cooper, A. Eckerdal, H. Johnson, R. McCartney et al. Students Designing Software: A Multi-national, Multi-institutional Study. *Informatics in Education* 4(1):143–162, 2005.
- [TRT06] B. Thomasson, M. Ratcliffe, L. Thomas. Identifying novice difficulties in object oriented design. In *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*. Pp. 28–32. 2006.



# Position Paper: Software Modelling Education

Martina Seidl<sup>1,2\*</sup>, Peter Clarke<sup>3</sup>

<sup>1</sup>Institute for Formal Models and Verification, Johannes Kepler University, Austria,

<sup>2</sup>Business Informatics Group, Vienna Technical University, Austria,

<sup>3</sup>Florida State University, USA

**Abstract:** Model-driven engineering (MDE) is a promising paradigm to deal with the ever increasing complexity of modern software systems. Its powerful abstraction mechanisms allow developers to focus on the essential challenges hiding away irrelevant aspects of the system under development. Within the last few years, noticeable progress has been made in putting the vision of MDE into practice, where the activity of textual coding is substituted by modeling. With matured concepts and stable tools available, MDE becomes more and more ready to be applied in software engineering projects.

Nevertheless, the best available technology is worthless, if it is not accepted and used by the developers. Also in MDE profound training is needed to fully exploit its power. In this paper, we discuss the efforts taken in educational environments to promote the application of modeling and MDE technologies for the software development process and discuss several challenges which still have to be faced.

**Keywords:** Software Modeling Education, Computer Science Education Research

## 1 Introduction

The way how large software systems are systematically developed has been steadily improving over the last decades in order to deal with the increasing size and complexity of such systems [Boe06]. One important driving force for the progressive advancements achieved in the discipline of software engineering is the concept of *abstraction* [Jac06] as it can be observed in the move from machine language to assembly, from assembly to structural high level languages which finally paved the way to modern object-oriented languages [GJ08]. The object-oriented paradigm aims at representing domain knowledge about the real world in a manner which is both natural for the human and easy to process for the computer. Hand in hand with the successful application of object-oriented programming languages is the introduction of object-oriented analysis and design methods relying on the notion of software models expressed in languages like the *Unified Modeling Language* (UML) [BM99]. However, the gap between the design of a software system and the actual realization introduces the problem of co-evolution. When both tasks—modeling and coding—are performed, an additional source for errors is introduced, because both sets of artifacts have to be kept in a synchronous state during the complete software life cycle. As a consequence, the solution is to merge modeling and programming and to provide

---

\* Parts of this work have been funded by the Vienna Science and Technology Fund (WWTF) under grant ICT10-018.

the means to generate executable code automatically from models as proposed by model-driven engineering [Sch06].

Based on this idea, modeling has experienced an enormous boost not only in academia but also in industry. With the concretization of the vision of *model-driven engineering* (MDE), models are lifted from mere documentation artifacts to full first class citizens in the software development process offering a tool which is expected to drastically change the way how software is developed. Still, it is a far way to go before the complete software development engineering process can be transformed using MDE techniques. In fact, it is not expected that traditional coding is eliminated over night, but slowly the new model-driven methods are incorporated to support programmers doing their work. Since developing software according to the MDE paradigm is different than traditional software engineering, programming knowledge and experience cannot directly be transferred to modeling [Fra09] and novel training methods are indispensable.

With this fast evolution of the development methodology, academia is confronted with a severe problem. On the one hand, students shall be offered profound knowledge on well-established MDE technologies and approaches, but on the other hand they shall have a competitive knowledge on the state-of-the-art when they leave university. So the question educators are confronted with is how to judge which approaches have a longer perspective of practical application, and which ones are only interesting in a short term perspective and will be forgotten when the first hype is over. This challenge is certainly not new for people who are responsible for the development of computer science curricula. Still there are several areas like theoretical computer science which are relatively stable, and other ones which are still experiencing an extreme progress such as the area of software development. Novel technologies can only be applied if people are available who are knowledgeable in such technologies, i.e., how to use them properly, otherwise the benefits are quickly lost and additional complexity is introduced.

In this paper, we give an overview of the activities necessary to train developers effectively in modeling. Therefore, we first consider the different of areas which cover the wide spectrum of modeling. Then we review the actions taken in the EduSymp, the premier forum for educators working in the field of modeling. Finally, we derive a set of urgent open issues.

## 2 Software Modeling Education

Software modeling education is a very broad topic since modeling can be applied during several phases of the software development process. In addition, identifying pedagogical techniques required to train students how to think abstractly while creating models is also a major challenge. The lack of proper approaches to educating students in software modeling may result in students creating models that are either not applied in the correct manner, or not being used at all. As a result, the cost of educating students would not result in any tangible benefits and developers trained in traditional programming may consider modeling as an additional or even useless task.

In the following, we shall discuss topics which shall be covered by a curriculum in order to assure that models are applied in the complete software development process.

**Introduction to Modeling Languages.** Before students are able to apply modeling techniques in the context of practical software engineering projects, they need to understand the concepts

of modeling languages, i.e., they have to gain profound knowledge on the syntax and semantics of modeling languages. For this first step, several different approaches can be taken [EHL06]. In fact, this shows some similarities with programming education, where a big point of discussion is the language (or even the language paradigm) with which beginners are confronted with. Modeling may be applied for different purposes like for the description of software systems, for the description of business processes, database design, etc. One language which covers a very wide spectrum is the Unified Modeling Language (UML), a general purpose language which was developed with the goal for supporting as many users' needs as possible. Therefore, UML is extremely flexible with respect to not only the provided syntactical concepts but also its semantics being defined over numerous variation points to adapt UML to specific purposes. This plethora of concepts and the impreciseness of the standard usually is a great challenge for beginners who have to deal not only with a new material, but also with context sensitive concepts. A solution to this dilemma would either be the education of UML using a set of well-defined core concepts, as proposed in [SHMA08], or to start with a Domain Specific Language (DSL), where the students get a more focused introduction. Furthermore, the way how a modeling language is used strongly depends on the application context. Therefore, students have to understand that in some situations informal descriptions are sufficient whereas in other situations the models have to provide concise formal specifications.

**Dealing with Abstractions.** One major issue about modeling is to comprehend how to develop models at an appropriate level of abstraction. Using the appropriate modeling techniques it is possible to focus on aspects of the systems those aspects that are not relevant at that time. Using this approach, the complexity of huge systems can be grasped in a more direct way, because otherwise it would quickly overwhelm the human intellect. In fact, many programming activities demand the ability to create abstractions, however, this type of abstraction is different when it comes to modeling. In programming, abstractions are necessary in order to describe a problem in such a way that it can be represented by the specific programming language and processed by a computer. Modeling languages, in contrast, provide the means to describe the world as it is seen by humans, the step to executable code comes much later. Novices are easily tempted to put everything into a model or mix the levels of abstractions, making the model harder to read, harder to use and harder to maintain. Therefore modelers have to be trained to choose the applied level of abstraction with care [Rob09].

**Practical Application.** Software modeling is usually done with the intention of building systems that have some practical application. From an educational perspective, building such systems requires that it is not sufficient to learn how to create models on paper, but be able to apply such modeling skills in practice on real world projects using the appropriate modeling tools. Using such an approach prepares students to work in industry and to realize software projects using modeling as a basis. Therefore, a tight integration of modeling into software engineering curricula is required [CS208]. Models can be applied in many different parts of the software development life cycle, it is therefore important that students are allowed to learn modeling techniques by building models for the various phases of the development process. This can be accomplished in two phases. Phase one involves creating special projects for students tailored

for didactical purposes and under the guidance of an instructor. Phase two involves applying the modeling techniques learned to a real world project under the supervision of an experienced industry trainer. Phase two requires the collaboration with various industry partners. Only with this background, students become aware of the impact of their modeling activities and how the quality of a model influences the end product in terms of performance, maintainability, etc.

**Model-Driven Engineering.** Models can not only be used as design artifacts, but also as substitution of traditional textual code. Then coding activities are replaced by modeling activities. The executable system is then (semi-)automatically derived from the models. This way of building software requires a new way of thinking from the developers, because again the level of abstraction is shifted. Now only limited knowledge about computer architecture is necessary, because such optimizations are performed by the code generators. In order to generate functional code, a good understanding of the MDE tools is necessary, and in some cases customizations of the generated code may become necessary for realizing the required functionalities. If the code is also modified by hand, techniques supporting the co-evolution of code and models have to be applied, otherwise modifications get lost when the system is rebuilt. Furthermore, since MDE techniques are at a relative early stage, a profound understanding is necessary for deciding when the application of which software development paradigm is more appropriate.

**Model Engineering.** For applying MDE techniques special environments are required. With the Eclipse modeling framework, a lot of tools are available out of the box. For customization purposes, it may be beneficial to develop dedicated code generators or to implement model transformation engines which support the conversion between different modeling languages or rewrite certain modeling concepts. Therefore, it is important to include model engineering in any software engineering education program. With a deep understanding, how the applied tools work—which is certainly more easily gained when students develop small MDE environments on their own (as proposed in [BKS09])—it is also easier to apply available tools in the correct manner. Also concepts such as metamodeling and metametamodeling become more natural when a one has designed and implemented a modeling language and the behavior of code generators is easier to understand when they have been used.

Ideally, industry and academia collaborate for training and education. Whereas industry could provide practical application scenarios to the university students giving them both a good motivation as well as valuable experiences, the theoretical background could be directly promoted from research institutions to people working at companies offering the state-of-the-art knowledge.

### 3 The Educators' Symposium

The Educators' Symposium (EduSymp) is an annual event collocated with the International Conference on Model-Driven Engineering Languages and Systems (MODELS)<sup>1</sup>. The goal is to provide a platform where modeling educators meet, exchange experiences, and develop new approaches for promoting the MDE paradigm in education. In this section, we first outline the

---

<sup>1</sup> <http://www.modelsconference.org>

typical organization of this symposium by analyzing the former six editions as far as the data is available. Each EduSymp has the focus set on a special issue and some peculiar organizational highlight. We exemplarily review the last edition of the EduSymp 2010, held in Oslo, Norway. This analysis of the EduSymp allows us to identify issues which have not been handled so far and to derive challenges which should be considered in future editions.

### 3.1 Organization

The EduSymp is always held as a satellite event of the MODELS conference, so it is usually in parallel either with tutorials and with workshops and may use the infrastructure of the conference. The EduSymp has no budget on its own, so no funding for speakers is available. In order to successfully hold the symposium, a couple of people are necessary for the organization as discussed in the following.

**Organizers.** The EduSymp is organized by one or two researchers/educators working in the modeling area. The organizers of the next edition are usually announced at the closing session of the MODELS conference, so the designation usually takes place during the EduSymp. Usually very active participants are invited to do the organization for the next year. The tasks include the formulation of the call for papers, the advertisement of the call for papers, the selection of the program committee members, the guidance of the reviewing process, the notifications of the authors, etc. In short, the organizers have the typical responsibilities similar to the chairs of a research event. In the past six editions of the EduSymp, the organization has been done by one or two professors or associate professors. Additionally, assistants have been involved in some years. So far, no two editions of the symposium have been organized from the same person indicating that there is high interest in the community for such an event.

**Program Committee.** The diagram on the left in Figure 1 shows the ratio between people from industry and from academia. The majority of the people are working at universities which is on the one hand no surprise, because reviewing activities are the daily business of academic researchers. On the other hand, many calls of the EduSymp foster investigations on the synergy between industry and academia in education and therefore, opinions from people of a practical background would be extremely valuable. The diagram on the right of Figure 1 indicates the distribution of the program committee members. In general, people from Europe are dominating. Here it has to be mentioned, that it rarely happens that more than two people from the same university are involved. The PC members from America are not only from the US, but also from Canada and countries of South America like Argentina and Brazil. Since modeling research is done all over the world, as is reflected in the participants of the main conference, and software engineering is part of almost all computer science curricula, a higher distribution and heterogeneity of papers presented at EduSymp should be possible. Such a mix of papers is preferable because then different viewpoints could be collected, analyzed, discussed and published.

**Participants.** Currently no statistics is available on the participants attending EduSymp. From our experience we can conclude that the EduSymp is mainly visited by the people who actively

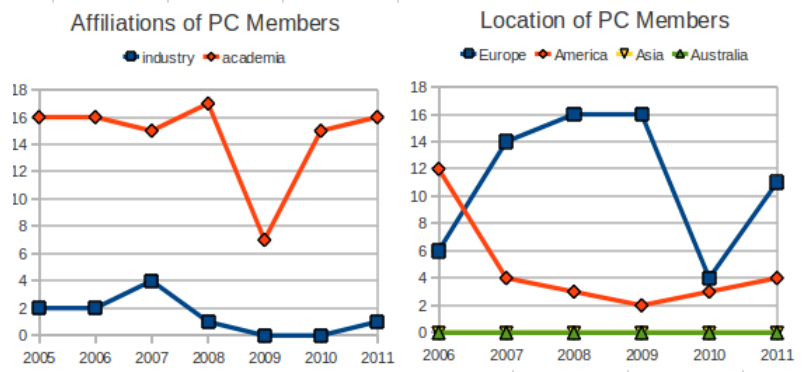


Figure 1: Statistics on the program committee.

contribute to the event, i.e., who present papers (cf. next subsection). Furthermore, there is a small core of professors who regularly attend the event. For those parts of the symposium where well known speakers are invited or renowned researchers or educators lead a panel discussion, the number of participants increases drastically.

**Agenda.** The EduSymp is a full day event usually consisting of four sessions. In the past, the symposium usually started with an invited talk, followed by the presentations of the accepted papers, both of short and long papers. Short discussions with the authors are possible. Highlighting the high interactivity of the symposium, either panels or working group or both are organized offering room for discussing pedagogical related issues with experts.

### 3.2 Contributions

The presentations of the peer-reviewed papers constitute an important part of the EduSymp. In the following, we review the topics for which contributions are invited, the topics of the actual contributions, and finally we analyze who submits the papers to the EduSymp in order to get an impression on the audience.

**Call for Papers.** In general, the EduSymp is intended to provide a forum for educators and trainers to exchange approaches and experiences on pedagogy for modeling education, experiences with various technologies and tools.

The accepted papers are published in terms of technical reports. In order to increase the visibility, the electronic journal of the EASST<sup>2</sup>, an indexed, open access journal, offers a publication channel since the last EduSymp. The two best papers as well as a short summary are included in the postproceedings of the MODELS.

**Topics of Papers.** Table 1 summarizes the types of contributions presented at the various editions of the EduSymp. Please note that for the year 2006 no list of accepted papers was available. The the first three categories (A, B, C) contain the papers on descriptions and experiences on

<sup>2</sup> <http://journal.ub.tu-berlin.de/eceasst>

	2005	2007	2008	2009	2010
A: Object-Oriented Modeling and UML	3	1	4	2	3
B: Modeling and Software Engineering	3	1	1	1	1
C: Model Engineering Techniques	0	0	1	1	2
D: Abstraction, Design Patterns, Formal Methods	2	1	0	1	0
E: Research on Modeling Education	1	1	2	0	0
F: Collaboration between Industry and Academia	0	0	1	0	1

Table 1: Types of contributions.

courses and curricula, whereas the later three categories (D, E, F) cover the papers on general methodologies and research on modeling education. Experience reports pose the majority of the presented contributions, where lecturers report on how they do the teaching. Only two papers discuss collaboration between industry and academia in modeling education. Topics like tool support are treated very sparsely. Interestingly, in first years of the EduSymp almost all papers were about UML related issues which changed over the years. This might be seen as an indicator how trends in research also influence education.

**Authors.** The geographical distribution of the authors is shown in Table 2. The majority of the authors are from Europe, even if the symposium is hosted by a non-European country. Since the organizers and the PC members are mainly from Europe, maybe the promotion is better in Europe than in other parts of the world. If between 10 and 15 papers are submitted to the EduSymp, then it might be considered as a success which might be easily beaten by good workshops. The problem is that the educators do not have a specific subcommunity in the modeling community as it is the case for example for model evolution, model verification, etc. Somehow everyone is involved in teaching, but only very few feel obligated to push education research. It also happens although not very often that the same author submits more than once to EduSymp.

### 3.3 Case Study: EduSymp 2010

The first highlight of the EduSymp 2010 was the room filling keynote “Formality in Education—Bitter Medicine or Bitter Administration” presented by Thomas Kühne where it was illustrated very vividly how an inadequate presentation of formal methods frightens off students. In the following sessions seven papers were presented covering innovative education approaches. One paper of note, which also won the best paper award, was by Brandsteidl et al.[BWH11] where the usage of novel media like document cameras in modeling education was shown in a live demo.

An online survey conducted prior to EduSymp served as the input for the interactive afternoon sessions consisting of working groups and a panel. The survey consisting of 18 questions was launched two weeks before the symposium and promoted via relevant mailing lists. More than 60 persons participated stating how and when they teach modeling. We used the answers for the categorization discussed in Section 2. Most of the participants use UML or Ecore in their courses, but also EER, BPMN and a diverse set of DSLs were named. The dominating tool in modeling education is the Eclipse platform. It showed that educators are in general happy with the tool support but there are several points of improvements like poor usability, poor support

	2005	2007	2008	2009	2010
Europe	3	4	8	4	5
America	5	0	0	1	0
Asia	0	0	1	0	2
Australia	1	0	0	0	0

Table 2: Geographical distribution of paper authors.

of standards, poor interchange facilities, etc. These issues gave a good starting point for the discussions led during the EduSymp. Details can be found in the summary [SC11]. Finally, a steering committee was established to watch and influence the development of the EduSymp.

## 4 Challenges

We conclude this paper with a set of challenges which are derived from the previous sections of this paper. These challenges will have to be handled in order to promote modeling education.

**Promotion.** Unfortunately, teaching has not the same impact as research. Concerning the career in academics, papers on new research results are highly renowned and are important milestones for the CV of an individual researcher. While teaching, on the other hand, it is sufficient to have a list of courses where some involvement has been taken, so no formal evaluation criteria are available for this thing. This is also the reason why teaching sometime comes up very short and is often passed to junior researchers. In consequence, courses are often repeated and innovations are kept at a minimum in order to reduce effort necessary to spend on teaching. This fact results in a major challenge for EduSymp, i.e., getting more educators involved in research and publishing in software modeling education. Although the majority of the MODELS' attendees are involved in teaching, only very few people participate in the EduSymp. Discussions on teaching are kept very private and so no community collected knowledge is possible.

**Repository of Teaching Artifacts.** One of the most time consuming tasks in course preparation, is the design of exercises and questions for the tests. The time for this task can be drastically reduced if good samples are available which have to be adapted to the current context only. Over the years individual teachers obtain huge local repositories with their exercises, for example if an exam has to be offered six times per year. An example is the ReMoD-Repository.<sup>3</sup>

**Archival Collection of Knowledge.** One of the highlights of the EduSymp are the interactive sessions with working groups, panels, and open discussions. There, the participants discuss teaching related topics and develop novel strategies. Unfortunately, the ideas and thoughts are hardly collected, usually only a short summary is provided by the organizers published on the three pages included in the collection of the postproceedings of the MODELS satellite events. In this way, the results of the EduSymp get lost and interesting ideas are never put into practice. To

<sup>3</sup> [www.cs.colostate.edu/remodd/publications\\_files/ReMoDD-MiSE09.ppt](http://www.cs.colostate.edu/remodd/publications_files/ReMoDD-MiSE09.ppt)



alleviate this problem, it would require the organizers to define precise protocols to capture and store this knowledge in a central repository that is accessible to all modeling educators.

**Dedicated Tool Support.** One of the greatest challenges educators and students are confronted with is the handling of the tools. In software engineering hands-on-experience is of particular importance in order to understand the techniques applied to build software of high quality. Since modeling is a quite young research area, tools work often well enough to use them in research projects, but the user has to know how to treat them. So the tools introduce additional complexity which shift the attention away from the actual problems to be solved. Furthermore, the tools usually offer too many features distracting the attention from the essential problems to be solved and the documentation is often insufficient. Teachers, who usually have little or no classroom support, this problem is a difficult one to solve. Therefore, it would be nice to have dedicated tool support for education where the environment is tailored and configured to the specific needs for the specific course. This can be partly be achieved by providing complete Eclipse bundles.

**Evaluation Criteria.** The output of teaching activities are very hard to evaluate and to compare. In the papers describing courses, typical measures are: the grades of the students, results of questionnaires where the students had to provide some feedback, which are very often the subjective impression of the teacher. In order to obtain meaningful results, time has to be spend on the elaboration of quality measures for models applicable on students' work.

**Industrial Commitment.** Education performed at universities is not only done to produce researchers but also to prepare software engineers and developers for industry. Without close cooperation with industry, the danger is inherent to produce academics who are able to use the most recent concepts theoretically, but who have no experience for practical tasks.

**Student Involvement.** The EduSymp is only attended by educators and trainers, giving students no chance to present their experiences and ideas when they have been recently trained on modeling. In fact, after a course they should have a profound understanding on the techniques and tools and they could give good feedback on the pros and cons of the taught content, especially when they did some advanced work in the context of practicals or theses. One reason, why a student discussion group is not so easy to organize is for traveling the budget is often very low and that only in very rare cases funding for sending students to conferences is available.

## 5 Conclusion

The EduSymp is small, well-established event for the promotion of various kinds of modeling techniques in computer science education. In this paper, we gave a short overview of the past editions of this symposium which allowed us to derive challenges which will have to be faced in the future in order to establish modeling as a basic discipline in computer science curricula. We conjecture that in future only with an adequate understanding of the abstraction power of models, complex computer systems can be handled appropriately. Therefore, the research community together with the practitioners from industry must decline the competencies required for the

efficient application of modeling in the software engineering process. Then it is possible to set up curricula which cover the wide spectrum of modeling.

**Acknowledgements:** The authors would like to thank Andreas Winter and Marion Brandsteidl for their valuable comments.

## Bibliography

- [BKS09] P. Brosch, G. Kappel, M. Seidl, M. Wimmer. Teaching Model Engineering in the Large. In *Educators' Symposium @ Models 2009*. 2009.
- [BM99] J. Bézivin, P. Muller. UML: The Birth and Rise of a Standard Modeling Notation. In *Proc. of UML 1998*. Pp. 514–514. Springer, 1999.
- [Boe06] B. Boehm. A View of 20th and 21st Century Software Engineering. In *Proc. of ICSE 2006*. Pp. 12–29. 2006.
- [BWH11] M. Brandsteidl, K. Wieland, C. Huemer. Novel Communication Channels in Software Modeling Education. In *Workshops and Symposia at MODELS 2010*. LNCS 6627, pp. 40–54. Springer, 2011.
- [CS208] CS2008 Review Taskforce. Computer Science Curriculum 2008. Technical report, ACM and IEEE, 2008.
- [EHLS06] G. Engels, J. H. Hausmann, M. Lohmann, S. Sauer. Teaching UML Is Teaching Software Engineering Is Teaching Abstraction. In *Proc. of Satellite Events at the MoDELS 2005 Conference*. LNCS 3844, pp. 306–319. Springer, 2006.
- [Fra09] R. France. Why Johnny cant model. *SoSym* 8(2):163–164, 2009.
- [GJ08] C. Ghezzi, M. Jazayeri. *Programming Language Concepts*. Wiley, 2008.
- [Jac06] D. Jackson. *Software Abstractions*. The MIT Press, 2006.
- [Rob09] P. Roberts. Abstract thinking: a predictor of modelling ability? In *Educators' Symposium @ Models 2009*. 2009.
- [SC11] M. Seidl, P. J. Clarke. Software Modeling in Education: The 6th Educators' Symposium at MODELS 2010. In *Workshops and Symposia at MODELS 2010*. LNCS 6627. Springer, 2011.
- [Sch06] D. Schmidt. Model-driven engineering. *Computer* 39(2):25–31, 2006.
- [SHMA08] J. Sourrouille, M. Hindawi, L. Morel, R. Aubry. Specifying consistent subsets of UML. In *Educators' Symposium @ Models 2008*. 2008.