Teaching Touchy Transformations

Martin Gogolla

University of Bremen (D), CS Department, Database Systems Group

Abstract. This paper reports on a teaching unit on model development and model transformation. One example model is first developed and considered as the source of various possible transformations. These transformations are explained implicitly afterwards by showing the different target models obtained by the transformations. The source model and the target models each emphasize a particular aspect, and an appropriate teaching method is chosen in order to communicate central ideas in a well-understandable way. The chosen teaching methods stress active student participation in the development of models and transformations.

1 Motivation and Context

For the success of model-centric software development it is crucial to convince software developers that models and transformations help to produce software more efficiently. Developers to be convinced include our students, and therefore good teaching practice is an important ingredient in bringing model-driven techniques into practice. The teaching unit described here introduces a complete UML and OCL example model with invariants and pre- and postconditions. The example model is transformed into different other models where each model underlines a particular modeling aspect. We have decided to call the transformations touchy (in the sense of delicate) because they sometimes seemingly introduce minor modifications but modifications which substantially modify the accepted system states and allowed operation sequences in the models. The studied transformations are explained by showing the source model and the target models. The paper does not explicitly explain how the transformations are realized in an operational way, but points to the major properties of the target models, and is therefore more related to transformation models in the sense of [BBG+06] than to general model transformations.

All models in this paper are executable. This means that all operations possess an operational realization which can be executed in the USE [GBR07] system, the UML and OCL tool that we employ for teaching. After carrying out a particular operation, the achieved system state can be checked and inspected with the USE system in a number of different graphical and textual ways. Thus modeling gets close to programming, a peculiarity which students usually do like much because executable models (like programs) give immediate and incremental feedback during development.

Teaching UML and OCL means to teach syntax and semantics of these languages. But for the formation of development skills it is in addition necessary

to teach the pragmatics and use of UML and OCL. One good approach to do this are well-chosen examples. For our teaching unit, we have taken an example with 3 classes, 2 associations, 7 attributes, and 7 operations. Our aim is to show this example from different viewpoints and to be able to compare the different solutions and transformations. In order to keep the comparison manageable, we have decided in favor of a relatively small model. The teaching unit is designed for two to four 2-hour lectures depending on the depth of the presented details and an audience up to 40 students because of the chosen interactive teaching methods. It assumes good knowledge on UML and OCL and is placed at the end of a weekly 4-hour course on UML and OCL.

Our work shares motivation and goals with similar approaches. [EHLS05] discusses the importance of teaching modeling in software engineering. In [KS05] the authors show how to integrate best modeling practices for teaching UML. Related teaching techniques to ours have been used in courses for reactive systems [Hau06], for development with components like J2EE or DOTNET [CDM+07,Var06], and for petri nets, metamodels and graph grammars [vGSDJ07]. [SS06] uses a teaching method close to one of our methods where student groups have to work with artifacts elaborated by other student groups. In contrast to the mentioned approaches, our focus is on working out and comparing one source model and different target models, and we are not aware of a published teaching unit on this topic.

The rest of this paper is structured as follows. Section 2 introduces our teaching goals and our fundamental teaching methods. In Sect. 3 we discuss our teaching subject by explaining the example system in verbal form and with six formal models as well as the transformations and a comparison between the source models and the target models. Section 4 shows the employed teaching methods. The paper is finished with a conclusion.

2 Teaching Goals and Teaching Methods

Our teaching goals are expressed as follows.

- Learning to work with models in different styles and to transform them.
- Recognizing the pros and cons of different target models.
- Learning that there is no unique canonical model for a problem domain.
- Development of alternative solutions and assessing their value.
- Combining different modeling styles as appropriate.
- Validation of developed models (classes, attributes, associations, invariants, pre- and postconditions, operation implementations) and their transformation with tools like USE [GBR07].
- Checking properties of source and target models and development of missing models element.
- Strong active student involvement.

The applied teaching method for one of the various models are chosen from the following options.

– Presentation of model fragments as a cloze test.
– Presentation of a complete model by the teacher.
– Presentation of a partial model by the teacher where missing elements are completed by the students.
– Presentation of an erroneous model by the teacher where faulty elements have to be corrected by the students.
– Presentation of an analogous but different example solution by the teacher and development of the aimed model by students through drawing analogies.
– Presentation of the desired model style in verbal form by the teacher and presentation of a solution proposal which does not follow that style with subsequent corrections by the students.

The chosen options for the student contributions are as follows.

– Student solution moderated by the teacher in front of the class where the additions are made by student acclamation.
– Student solution produced in smaller student groups during the lecture and following presentation of the results to all course participants.
– Student solution developed in smaller student groups as homework and presentation of the results to all course participants.
– Student solution organized to be produced in phases and in groups as homework or in the classroom. After completion of a phase the student groups exchange their solutions and work further with a solution made by another student group. Subsequent discussion of pros and cons among the students.

3 Teaching Subject

The teaching unit comprises several descriptions of the same simple library system formulated on different abstraction level (from informal over formal to executable) and in different styles (e.g., in an 'object-oriented' design-like style, in a relational database style or in a Java-like style). The diagram in Fig. 1 gives an overview.

Informal: The starting point is an informal English text describing the intended system.
MaxInvsMinPrepos: The first model formulates as much as possible with invariants and as less as possible with prepos (pre- and postconditions).
MaxPrepos: The next solution does not use invariants at all, but encodes invariant conditions as operation preconditions.
Assoc2Attr: The third model does not use associations but object- and collection-valued attributes instead.
RelDB1NF: The following model describes the system state in form of a relational database schema in first normal form.
Invs2Super: The fifth model factors out common invariants into a new abstract generalized class responsible for a particular invariant form.

CompFrame: The last model specifies complete frame conditions for the operations describing not only what the operations are expected to do but also what the operations are *not* allowed to do.

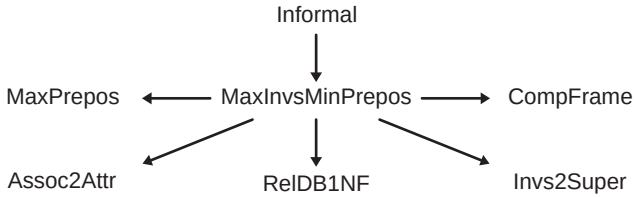

Fig. 1. Overview on Example Models and Transformations

3.1 Informal Description

The informal description presents in the first part an English text for the system and in the second part a cloze test to be filled in by the students.

```
The example describes a digital support system for a library. The
library offers book copies to users. A user can borrow a copy or in
other words, an exemplar, of a book. A book is characterized by an
author list, a year of publication, and a unique title. A copy is
determined by the number of return actions of the copy, the book of
which the copy is an exemplar of, and a unique signature. A user has
an address and a unique name. At most one user can borrow a copy of a
book at one particular point in time. Book, copy, and user properties
are first manipulated by initialization actions. Both users and copies
are able to perform actions for borrowing and returning.

Additionally, certain conditions must hold. If properties such as
author, title, ---------, address, and ---- are described by
strings, the string is not allowed to be undefined or to be equal to
the empty string. A year of ----------- is equal to or greater than
1455 (the year in which the Gutenberg bible was published). A ----
having borrowed a copy of a particular ---- is not allowed to borrow
another copy of the same book at the same time. An ------ can appear
at most once in an ------ list. Finally, as already indicated above,
certain properties such as -----, ---------, and ---- are unique.

Initialization, borrow and ------ actions have to respect the above
----------. They can only be performed meaningfully in reasonable
situations. They have to fulfill their expected functionality.
```

3.2 Model MaxInvsMinPrepos

Figure 2 shows the class diagram for the model MaxInvsMinPrepos and the model MaxPrepos. We identify the 3 classes, the associations, the attributes, and the operations. Each class has an initialization operation. The borrow and

return operations basically manipulate the Borrows association, and the return operations additionally modify the attribute numReturns in the class Copy. The BelongsTo association is managed by the initialization operation in the class Copy. The operations are characterized by the pre- and postconditions in the left side of Fig. 3 in which first the 10 invariants and afterwards the pre- and postconditions are stated (only the names are shown). As an example let us consider the details of the invariant User::noDoubleBorrowings which realizes a condition stated in the second paragraph of the verbal description.

```
context u:User inv noDoubleBorrowings:
  not(u.copy->exists(c1,c2|c1<>c2 and c1.book=c2.book))
```

However, this invariant can also be equivalently formulated in the context of the class Copy or of the class Book.

```
context c1:Copy inv noDoubleBorrowings:
  not(Copy.allInstances->exists(c2| c1<>c2 and c1.user=c2.user and
    c1.user.isDefined and c1.book=c2.book))

 context b:Book inv noDoubleBorrowings:
  not(Copy.allInstances->exists(c1,c2| c1<>c2 and c1.user=c2.user and
    c1.user.isDefined and c1.book=b and c2.book=b))
```

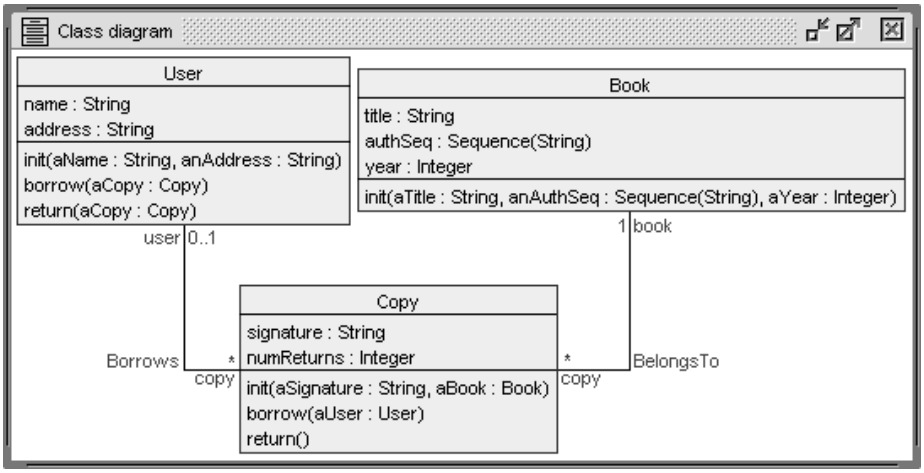

Fig. 2. Class Diagram for MaxInvsMinPrepos and MaxPrepos

In Fig. 4, we show how some interesting operations from this model are realized with command sequences. The formal paramaeters from the operations may be used as ordinary variables in the command sequences. They are actualized with actual values and objects when the operations are called.

3.3 Model MaxPrepos

The model MaxPrepos has the same class diagram as the model MaxInvsMin-Prepos. The two models differ in that the invariants have been transformed

Constraints in MaxInvsMinPrepos

```
inv  User::nameAddressFormatOk
inv  User::nameIsKey
inv  User::noDoubleBorrowings
inv  Copy::signatureFormatOk
inv  Copy::signatureIsKey
inv  Book::titleFormatOk
inv  Book::titleIsKey
inv  Book::authSeqFormatOk
inv  Book::authSeqExistsAndUnique
inv  Book::yearPlausible

pre  User::init freshUser
post User::init attrsAssigned

pre  User::borrow copyOk
post User::borrow linkAssigned

pre  User::return aCopyOk
post User::return linkRemoved
post User::return numReturnsIncreased

pre  Copy::init freshCopy
pre  Copy::init bookOk
post Copy::init attrsAndLinkAssigned

pre  Copy::borrow userOk
pre  Copy::borrow notBorrowed
post Copy::borrow linkAssigned

pre  Copy::return copyOk
post Copy::return linkRemoved
post Copy::return numReturnsIncreased

pre  Book::init freshBook
post Book::init attrsAssigned
```

Constraints in MaxPrepos

```
* pre  User::init nameAddressFormatOk
* pre  User::init nameIsKey
  pre  User::init freshUser
  post User::init attrsAssigned

* pre  User::borrow noDoubleBorrowings
  pre  User::borrow copyOk
  post User::borrow linkAssigned

  pre  User::return aCopyOk
  post User::return linkRemoved
  post User::return numReturnsIncreased

* pre  Copy::init signatureFormatOk
* pre  Copy::init signatureIsKey
  pre  Copy::init freshCopy
  pre  Copy::init bookOk
  post Copy::init attrsAndLinkAssigned

* pre  Copy::borrow noDoubleBorrowings
  pre  Copy::borrow userOk
  pre  Copy::borrow notBorrowed
  post Copy::borrow linkAssigned

  pre  Copy::return copyOk
  post Copy::return linkRemoved
  post Copy::return numReturnsIncreased

* pre  Book::init titleFormatOk
* pre  Book::init titleIsKey
* pre  Book::init authSeqFormatOk
* pre  Book::init authSeqExistsAndUnique
* pre  Book::init yearPlausible
  pre  Book::init freshBook
  post Book::init attrsAssigned
```

Fig. 3. Constraints in MaxInvsMinPrepos and MaxPrepos

```
-- User::init(aName:String, anAddress:String)
!set self.name:=aName
!set self.address:=anAddress

-- User::borrow(aCopy:Copy)
!insert (self,aCopy) into Borrows

-- User::return(aCopy:Copy)
!set aCopy.numReturns:=aCopy.numReturns+1
!delete (self,aCopy) from Borrows

-- Copy::borrow(aUser:User)
!insert (aUser,self) into Borrows

-- Copy::return()
!set self.numReturns:=self.numReturns+1
!delete (self.user,self) from Borrows
```

Fig. 4. Realization of Operations with Command Sequences

into operation preconditions. In the right side of Fig. 3, one recognizes that the invariants have been transformed into preconditions. Preconditions marked with the star `*` correspond to invariants. From the 10 invariants in the first model we obtain 11 preconditions in this second model: This arises from the fact

that the invariant User::noDoubleBorrowings must be respected by the operation User::borrow *and* the operation Copy::borrow.

```
context User::borrow(aCopy:Copy)
pre noDoubleBorrowings: self.copy.book->excludes(aCopy.book)

context Copy::borrow(aUser:User)
pre noDoubleBorrowings: aUser.copy.book->excludes(self.book)
```

For example, the uniqueness condition for User names, is represented by checking the following precondition of User::init.

```
context User::init(aName:String, anAddress:String) pre nameIsKey:
  User.allInstances->collect(u|u.name)->excludes(aName)
```

One advantage of this model is that the global invariants have been localized to the respective operations which could violate the invariants. This means that the conditions can be checked in a more effective way. One drawback of this model may be seen in the fact that the invariant properties of the system states cannot be seen directly. Without having the invariants explicitly available, one has to deduce the system state properties from the operation descriptions.

3.4 Model Assoc2Attr

As presented in Fig. 5, the model Assoc2Attr realizes the associations by object- and collection-valued attributes. The invariants from the model MaxInvsMinPrepos remain unchanged. Additional invariants take care that the respective attributes representing the association ends are inverse to each other. For example, the requirements for the association Borrows are characterized as follows.

```
context u:User inv userCopyUserEQuser:
  u.copy<>oclEmpty(Set(Copy)) implies u.copy.user->asSet()=Set{u}
context c:Copy inv copyUserCopyEQcopy:
  c.user<>oclUndefined(User) implies c.user.copy->includes(c)
```

In addition, one invariant is needed for the multiplicity 1 in the BelongsTo association and a number of preconditions have to be changed, for example in the context of User::init, instead of requiring something like `self.copy->isEmpty()` one must demand `self.copy = oclUndefined(Set(Copy))`.

3.5 Model RelDB1NF

In Fig. 6 the model RelDB1NF is shown. The class Library is a singleton class with a single, complex structured attribute. Thus the system state is represented by a single, complex structured value. All operations modify this single attribute. User has the key name, Copy possesses signature as the key, Book has the key title, and authSeq has the attribute set {title, pos} as its key.

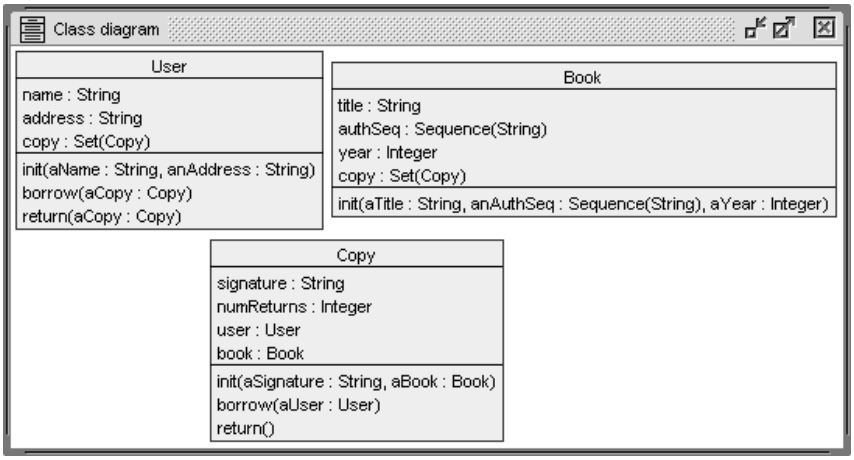

Fig. 5. Model Assoc2Attr

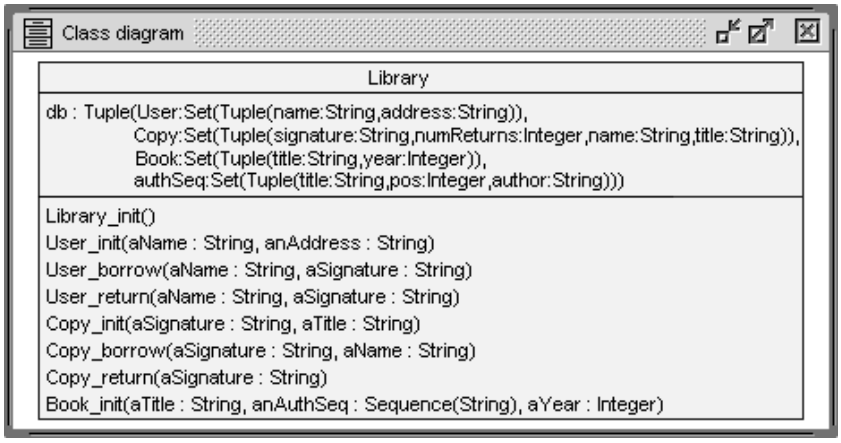

Fig. 6. Model RelDB1NF

The User operations can be described as follows (other operations work similar): The operation User_init(aName, anAddress) adds a User tuple; the operation User_borrow(aName, aSignature) updates the name component in a Copy tuple with the parameter aName; the operation User_return(aName, aSignature) also updates a Copy tuple by setting the name component to undefined.

Apart from considering the relational model, we have ready solutions for the data modeling part which follow the hierarchical, the network, the object-oriented or the semi-structured (XML-like) data model.

3.6 Model Invs2Super

The model Invs2Super in Fig. 7 is based on the observation that the classes User, Copy and Book share key constraints which in all three classes have the same

structure. This model introduces the abstract superclass Keyed which embodies the key constraint and two operations which allow to connect a subclass to the constraint. The key constraint in the superclass requires that two different and comparable objects must have different key values. The operation key in the subclass indicates how its key value is computed, and the operation comparableTo characterizes which objects from the most general class OclAny are comparable to the subclass. The advantage of this model is that the key constraint is formulated only once.

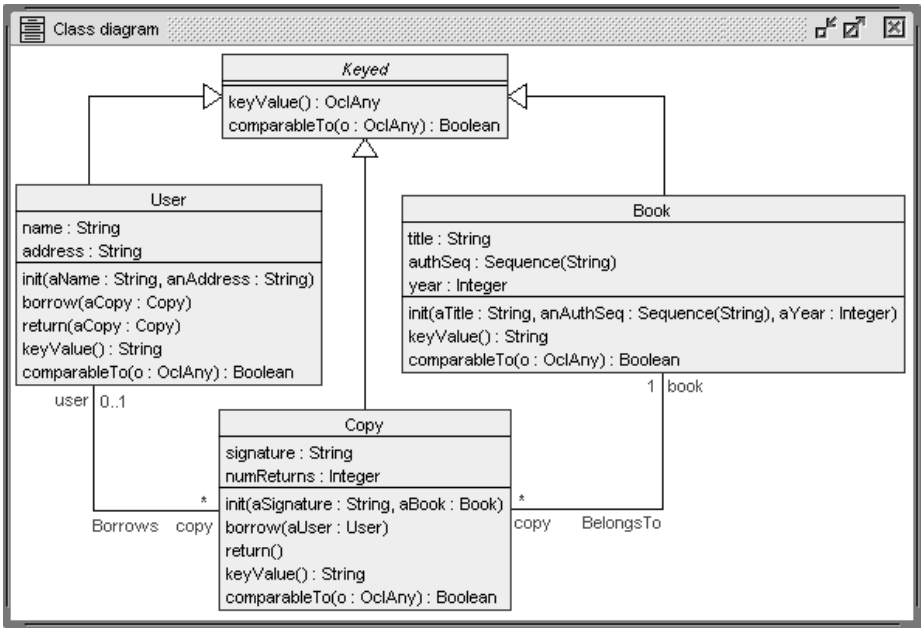

Fig. 7. Model Invs2Super

```
abstract class Keyed
operations
  keyValue():OclAny=oclUndefined(OclAny)
  comparableTo(o:OclAny):Boolean=oclUndefined(Boolean)

context self:Keyed inv differentObjectsDifferentKeys:
  Keyed.allInstances->forAll(self2|
    self<>self2 and self.comparableTo(self2) implies
      self.keyValue()<>self2.keyValue())

class User < Keyed
attributes
  name:String -- key
  ...
operations
  ...
  keyValue():String=name
```

```
comparableTo(o:OclAny):Boolean=o.oclIsTypeOf(User)
```

In the example, factoring out invariants into a generalized superclass could also be applied to the invariants having names ending with 'FormatOk' by introducing an abstract class FormattedStringSet and defining in each class an appropriate set.

3.7 Model CompFrame

The model CompFrame has the same class diagram and includes the same invariants, and pre- and postconditions as the model MaxInvsMinPrepos, but new postconditions are added. These postconditions are so-called frame conditions, which assure that the operations do not do something which they are not supposed to do. The frame is determined by the properties in the class diagram, i.e., the class instances (set of objects), the attributes and the roles. For example, the postconditions in User::init require state changes in a User object, but they make not requirement about the book objects. One would typically assume that everything that is not mentioned in the postcondition does not change, but this is not formally required. For example, an implementation of User::init which resets the attribute numReturn in all Copy object to zero would satisfy the postconditions in MaxInvsMinPrepos. The new postconditions require that only the 'intended' changes take place. Below we show the names of the pre- and postconditions in the class User and indicate which are frame conditions.

```
User::init pre  freshUser
User::init post attrsAssigned
User::init post userNearlyUnchanged -- F R A M E
User::init post copyUnchanged       -- F R A M E
User::init post bookUnchanged       -- F R A M E

User::borrow pre  copyOk
User::borrow post linkAssigned
User::borrow post userNearlyUnchanged -- F R A M E
User::borrow post copyNearlyUnchanged -- F R A M E
User::borrow post bookUnchanged       -- F R A M E

User::return pre  aCopyOk
User::return post linkRemoved
User::return post numReturnsIncreased
User::return post userNearlyUnchanged -- F R A M E
User::return post copyNearlyUnchanged -- F R A M E
User::return post bookUnchanged       -- F R A M E
```

In the following, we show the postconditions userNearlyUnchanged and copyUnchanged of the operation User::init in detail. For example, the postcondition userNearlyUnchanged demands that all User object except the object on which init is called are unchanged. The fact that, for example, the operation

User::borrow induces changes on User *and* Copy objects, is formally reflected by the postconditions userNearlyUnchanged and copyNearlyUnchanged.

```
context User::init(aName:String, anAddress:String)
post userNearlyUnchanged:
  User.allInstances@pre=User.allInstances and
  User.allInstances->forAll(u|
    (u<>self implies u.name@pre=u.name) and
    (u<>self implies u.address@pre=u.address) and
    u.copy@pre=u.copy)
post copyUnchanged:
  Copy.allInstances@pre=Copy.allInstances and
  Copy.allInstances->forAll(c|
    c.signature@pre=c.signature and c.numReturns@pre=c.numReturns and
    c.user@pre=c.user and c.book@pre=c.book)
```

Changes can be classified into changes regarding the set of objects in the respective class, the attributes in the class and the roles in the class.

3.8 Comparison

The following table compares the developed models with the start model MaxInvsMinPrepos and shows the main difference. If an entry in the table is empty, the respective model part is essentially identical to the model part of MaxInvsMinPrepos. The model MaxPrepos has no invariants, but preconditions instead. Assoc2Attr drops the associations in favor of attributes and adds invariants to guarantee inverse attributes. RelDB1NF has a single class and a single attribute. Invs2Super adds one abstract class and merges three invariants into a single one. CompFrame adds stronger postconditions.

Δ MaxInvsMinPrepos	Classes Assocs	Attrs Roles	Invs	Prepos
MaxPrepos			No invs	More preconds
Assoc2Attr	No assocs	Inverse attrs	More invs	
RelDB1NF	One class	One attr		
Invs2Super	Keyed		Key constr	
CompFrame				More postconds

Basically we have shown above five single transformations possessing MaxInvsMinPos as the source model. Three of these transformations can be combined to yield a target model where (A) the invariants are encoded as preconditions, (B) the associations are represented by object- and collection-valued attributes, and (C) complete frame conditions are achieved by adding strong postconditions.

4 Chosen Teaching Methods for the Different Models

For the different models different teaching methods are chosen as follows.

Informal: The first part of the informal description is presented by the teacher, and a cloze test for the second part is completed through acclamation by the students.

MaxInvsMinPrepos: The first model is presented without invariant names, but with the invariant text. Again by acclamation, the students have to find expressive, but not too long invariant names.

MaxPrepos: This solution is presented partly by the teacher. The solution for the transformation of the invariants of one class, User or Copy, is shown. The students have to complete it for the other classes in classroom groups.

Assoc2Attr: For this model an analogy solution, a simple Company-worksFor-Person model, is shown by the teacher, and the students have to find the Library solution by analogy in classroom groups.

RelDB1NF: The relational first normal form database schema is introduced by the teacher with small inconsistencies and errors. For example, the teacher will show the attribute set {signature, name} as the key for Copy or will include the non-first normal form attribute author:Sequence(String) in Book. The students have to identify the flaws by acclamation in the classroom.

Invs2Super: This solution is presented in the classroom by the teacher for one class. The students have to work out the remaining solution in groups as homework.

CompFrame: This model is to be developed basically in the classroom in phases following the class structure: (1) User, (2) Copy, and (3) Book. The solution for class User is introduced by the teacher. Afterwards each student group works on the class Copy. Then the solutions are exchanged among student groups, i.e., group A works further with the solution of group B and vice versa. Results are discussed afterwards in the classroom.

Alternatively, variations of the above teaching methods could be chosen, if suitable. The approach in [TS07] viewing software design as a game (games are usually attractive for students) will be considered by us for the next iteration of our course as a further teaching method.

5 Conclusion and Outlook

This paper has introduced a teaching unit for developing one system in different styles. The transition from one model to another model was explained as a model transformation which can be executed in automatic way (e.g., from MaxInvsMinPrepos to RelDB1NF) or must be performed by hand (e.g., from MaxInvsMinPrepos to MaxPrepos).

The teaching unit has been sucessfully implemented in a running course. Students have given quite positive feedback on the teaching unit. Solutions for all styles have been tried out and could be provided as sample solutions. Although we have only sketched this here, all operations in all models are not only specified by pre- and postconditions but are also implemented in an executable way, i.e., all models are executable in our tool USE (UML-based Specification Environment). The students in the course have deep knowledge in USE and were able to

apply it for implementation, testing and validation purposes. Further feedback from students and the community will help to improve the current state of the teaching unit.

References

[ACRR07] E. Astesiano, M. Cerioli, G. Reggio, and F. Ricca. A Phased Highly Interactive Approach to Teaching UML-Based Software Development. In M. Staron, editor, *Proc. 3rd MODELS Educators Symposium'07*, 9–18, 2007.

[BBG+06] J. Bezivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, A. Lindow. Model Transformations? Transformation Models! In O. Nierstrasz, J. Whittle, D. Harel, G. Reggio, Editors, *Proc. 9th Int. Conf. MODELS'06*, LNCS 4199, 440–453, 2006.

[CDM+07] J. Cabot, F. Duran, N. Moreno, R. Romero, and A. Vallecillo. From Programming to Modeling: Evolving the Contents of a Distributed Software Engineering Course. In M. Staron, editor, *Proc. 3rd MODELS Educators Symposium'07*, 29–33, 2007.

[dP05] P.F.W. de Padua. A Model-Driven Software Process for Course Projects. In H. Giese and P. Roques, editors, *Proc. 1st MODELS Educators Symposium'05*, 33–40, 2005.

[EHLS05] G. Engels, J.H. Hausmann, M. Lohmann, and S. Sauer. Teaching UML is Teaching Software Engineering is Teaching Abstraction. In H. Giese and P. Roques, editors, *Proc. 1st MODELS Educators Symposium'05*, 25–32, 2005.

[GBR07] M. Gogolla, F. Büttner, and M. Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34, 2007.

[Hau06] O. Haugen. Teaching Modeling of Reactive Systems. In L. Kuzniarz, editor, *Proc. 2nd MODELS Educators Symposium'06*, 30–44, 2006.

[KS05] L. Kuzniarz and M. Staron. Best Practices for Teaching UML-Based Software Development. In H. Giese and P. Roques, editors, *Proc. 1st MODELS Educators Symposium'05*, 9–16, 2005.

[SS06] R. Szmurlo and M. Smialek. Teaching Software Modeling in a Simulated Project Environment. In L. Kuzniarz, editor, *Proc. 2nd MODELS Educators Symposium'06*, 16–29, 2006.

[TS07] J. Tenzer and P. Stevens. GUIDE: Games with UML for Interactive Design Exploration. *Knowledge Based Systems*, 20(7):652–670, 2007.

[Var06] D. Varro. UML-Based Modeling and Development of J2EE Applications: Course Experiences. In L. Kuzniarz, editor, *Proc. 2nd MODELS Educators Symposium'06*, 45–60, 2006.

[vGSDJ07] P. van Gorp, H. Schippers, S. Demeyer, and D. Janssens. Students Can Get Excited about Formal Methods: A Model-Driven Course on Petri Nets, Metamodels and Graph Grammars. In M. Staron, editor, *Proc. 3rd MODELS Educators Symposium'07*, 19–28, 2007.