Using Constraints in Teaching Software Modeling

Dan Chiorean[1], Ileana Ober[2], and Vladiela Petraşcu[1]

[1]Babeş-Bolyai University, Cluj-Napoca, Romania
{chiorean,vladi}@cs.ubbcluj.ro
[2]Université Paul Sabatier, Toulouse, France
ober@irit.fr

Abstract. This paper approaches the issue of teaching software modeling by using OCL specifications, in a context in which the web is a major source of information. The increasing interest towards modeling has induced a higher need for clear and complete model 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. OCL beginners, and not only, need to be trained on how to avoid potential specification traps.
Our proposal is based on a complete and unambiguous description of requirements, that represents the first step towards good OCL specifications. The paper highlights several major aspects that need to be understood and complied with to produce meaningful and efficient OCL specifications. This approach has been tested while teaching OCL at Babes-Bolyai University of Cluj-Napoca.

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

1 Introduction

OCL is a language whose spread has not met 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 some progress in the above mentioned fields, the developers' feedback is far from satisfactory. This may be due to both the lack of suggestive examples regarding 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, apart from providing positive recommendations (books, papers, tools), warning potential OCL users on the pitfalls enclosed by negative examples is mandatory. 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 respect 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, allowing 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 the application 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 [5], [3]. 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 [9]. 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" [10] is to be understood as "Modeling is Understanding", since "Object-oriented development promotes the view that programming is modeling" [7]. 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 understading trying to code it." [10].

The modeling example from [9], 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 [9] 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.

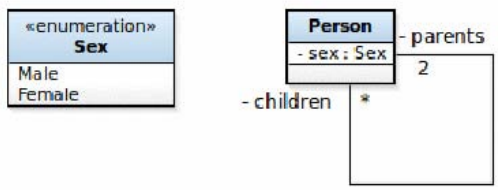

Fig. 1: Genealogical tree model [9]

Both this problem and its solution, consisting in relaxing the `parents` multiplicity to 0..2, are now "classical" [4]. 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 [9] 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 [9].

```
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 [1] and `undefined` in case of OCLE [6]). The results produced by OCLE comply with the latest OCL 2.3 specification [8]. 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 [4], 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 chidrenAge:
    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

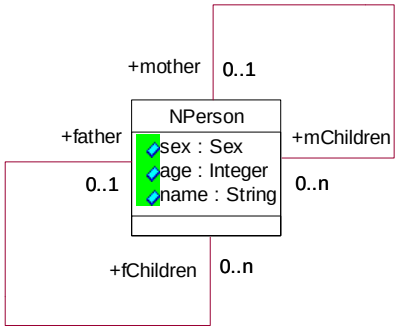

Fig. 2: An alternative model for expressing parents-children relationships

question contains two recursive 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' 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 explicitely 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 [9], 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 [2]: "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 libray 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 [9] 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`.

```
context Contract
```

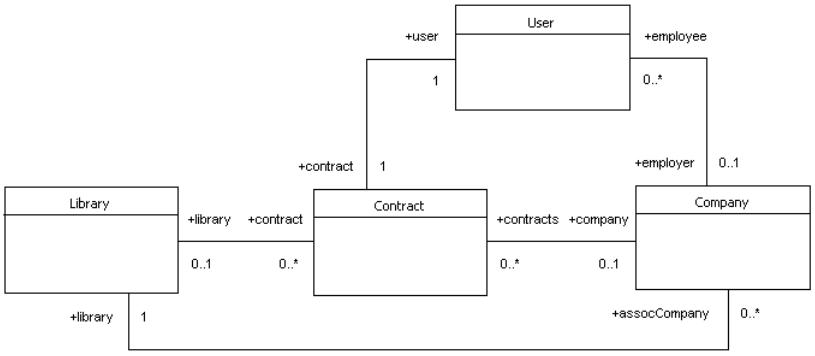

Fig. 3: A revised version of an excerpt of the library model from [9]

```
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 instantiations are not ruled out by the invariant

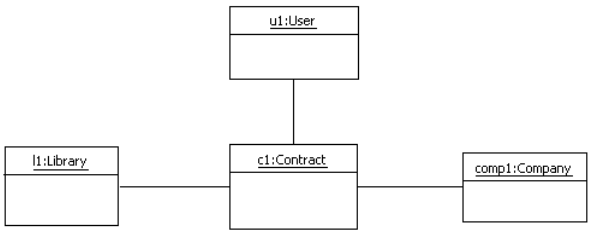

Fig. 4: The user has a contract with both the library and the company

proposed in [9] in the `User` context, namely `self.contract->notEmpty xor self.company <> null`.

Even more, in Figure 12 from [9], `contractB65` and `contractR43` have only one participant, `company80Y`, a stange situation in our oppinion. 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.

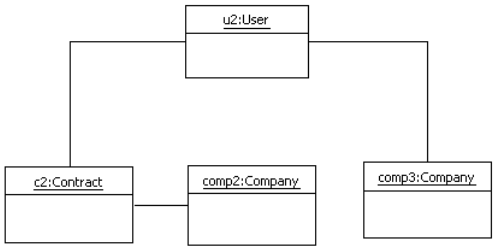

Fig. 5: The user is employed by `comp3`, but its contract `c2` is with `comp2`

Unfortunately, as stated before, our questions do not stop here. In Figure 10 from [9], 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 inteligibility 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.

References

1. A UML-based Specification Environment, http://www.db.informatik.uni-bremen.de/projects/USE
2. InvestorWords, http://www.investorwords.com/1079/contract.html
3. The OCL portal, http://st.inf.tu-dresden.de/ocl/index.php?option=com_content&view=category&id=5&Itemid=30
4. Cabot, J.: Common UML errors (I): Infinite recursive associations (2011), http://modeling-languages.com/common-uml-errors-i-infinite-recursive-associations/
5. Chimiak-Opoka, J., Demuth, B.: Teaching OCL Standard Library: First Part of an OCL 2.x Course. ECEASST 34 (2010)
6. LCI (Laboratorul de Cercetare în Informatică): Object Constraint Language Environment (OCLE), http://lci.cs.ubbcluj.ro/ocle/
7. Nierstrasz, O.: Synchronizing Models and Code (2011), invited Talk at TOOLS 2011 Federated Conference, http://toolseurope2011.lcc.uma.es/#speakers
8. OMG (Object Management Group): Object Constraint Language (OCL), Version 2.3 Beta 2 (2011), http://www.omg.org/spec/OCL/2.3/Beta2/PDF
9. Todorova, A.: 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/
10. Venners, B.: Abstraction and Efficiency. A Conversation with Bjarne Stroustrup - Part III (2004), http://www.artima.com/intv/abstreffi2.html