

Exercises for Teaching OCL Constraints

Martin Gogolla

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

Abstract: This paper is loosely connected to the paper 'Teaching Touchy Transformations' presented at the Educator's Symposium during the MODELS conference in 2008 by the same author. That paper reports on a teaching unit on model transformations within a course on UML and OCL. The current paper puts forward course exercises dealing with the same example model as the 'Teaching Touchy Transformations' paper, however the exercises can be also studied independently. The exercises concentrate on syntactically and semantically different versions of invariants.

Context

The teaching unit from the paper at the Educator's Symposium during the MODELS conference in 2008 is part of a course on UML and OCL. The course formally explains OCL syntax and semantics, see [Richters and Gogolla, 2001]. This paper describes exercises which delve into the details and language features of OCL by starting from a simple example model and by discussing small differences of invariants. The small syntactic differences may lead to semantically equivalent or non-equivalent invariants.

Exercises

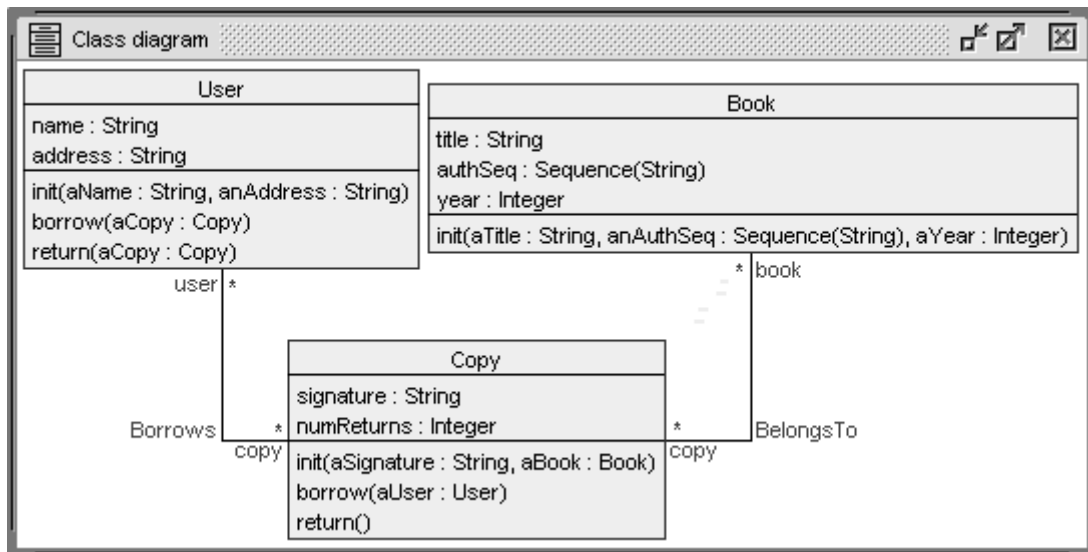
This exercise starts from a simple example model describing a digital support system for a library.

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

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

This information is roughly represented in the following class diagram.



Below we show a USE model (for USE see [Gogolla et al., 2007]) with different formulations for particular invariants mentioned above. For each version carry out the following.

- Explain in verbal form how each version $V_k_{\langle \text{name} \rangle}$ with $k \geq 2$ is syntactically different from the version $V1_{\langle \text{name} \rangle}$.
- Discuss whether each version $V_k_{\langle \text{name} \rangle}$ with $k \geq 2$ is semantically equivalent or non-equivalent to version $V1_{\langle \text{name} \rangle}$. In the case of equivalence, argue why the equivalence holds. In the case of non-equivalence, develop an object diagram that is valid for version $V1_{\langle \text{name} \rangle}$ and that is invalid for version $V_k_{\langle \text{name} \rangle}$.
- Develop three more syntactically different versions for each invariant. At least one of your versions has to be semantically equivalent to $V1_{\langle \text{name} \rangle}$ and at least one has to be semantically non-equivalent to $V1_{\langle \text{name} \rangle}$.

```

----- Library
model Library
----- class User
class User
attributes
  name:String -- key
  address:String
operations
  init(aName:String, anAddress:String)
  borrow(aCopy:Copy)
  return(aCopy:Copy)
end
  
```

```

----- class Copy
class Copy
attributes
  signature:String -- key
  numReturns:Integer
operations
  init(aSignature:String, aBook:Book)
  borrow(aUser:User)
  return()
end
----- class Book
class Book
attributes
  title:String -- key
  authSeq:Sequence(String)
  year:Integer
operations
  init(aTitle:String, anAuthSeq:Sequence(String), aYear:Integer)
end
----- association Borrows
association Borrows between
  User[0..*] role user
  Copy[0..*] role copy
end
----- association BelongsTo
association BelongsTo between
  Copy[0..*] role copy
  Book[0..*] role book
end
----- constraints
constraints
-----
context c:Copy inv V1_multiplicity_0_1:
  0<=c.user->size() and c.user->size()<=1
context c:Copy inv V1_multiplicity_1:
  1=c.book->size()
-----
context u:User inv V2_multiplicity_0_1:
  0<=u.copy->size() and u.copy->size()<=1
context b:Book inv V2_multiplicity_1:
  1=b.copy->size()
-----
context c:Copy inv V3_multiplicity_0_1:
  c.user->size()<=1
context c:Copy inv V3_multiplicity_1:
  1=c.book->size()
-----
context c:Copy inv V4_multiplicity_0_1:
  0<=c.user->size() and c.user->size()<=1
context c:Copy inv V4_multiplicity_1:
  1<=c.book->size() and c.book->size()<=1
-----
context c:Copy inv V5_multiplicity_0_1:
  let aSize=c.user->size() in 0<=aSize and aSize<=1
context c:Copy inv V5_multiplicity_1:
  1=c.book->size()

```

```

-----
context c:Copy inv V6_multiplicity_0_1:
  c.user<>oclUndefined(Set(User)) implies
    0<=c.user->size() and c.user->size()<=1
context c:Copy inv V6_multiplicity_1:
  c.book<>oclUndefined(Set(Book)) implies 1=c.book->size()
-----
context c:Copy inv V7_multiplicity_0_1:
  c.user<>oclUndefined(Set(User)) and
    0<=c.user->size() and c.user->size()<=1
context c:Copy inv V7_multiplicity_1:
  c.book<>oclUndefined(Set(Book)) and 1=c.book->size()
-----
context c:Copy inv V8_multiplicity:
  0<=c.user->size() and c.user->size()<=1 and 1=c.book->size()
-----
context c:Copy inv V9_multiplicity:
  let aSize=c.user->size()+c.book->size() in 0<=aSize and aSize<=2
-----

-----
context u1:User inv V1_nameIsKey: User.allInstances->forall(u2 |
  u1<>u2 implies u1.name<>u2.name)
-----
context User inv V2_nameIsKey: User.allInstances->forall(u2 |
  self<>u2 implies name<>u2.name)
-----
context u1:User inv V3_nameIsKey: User.allInstances->forall(u2 |
  u1.name=u2.name implies u1=u2)
-----
context u1:User inv V4_nameIsKey: User.allInstances->forall(u2 |
  u1=u2 or u1.name<>u2.name)
-----
context u1:User inv V5_nameIsKey: User.allInstances->forall(u2 |
  u1.name<>u2.name or u1=u2)
-----
context User inv V6_nameIsKey: User.allInstances->isUnique(u|u.name)
-----
context u1:User inv V7_nameIsKey: User.allInstances->forall(u2 |
  u1=u2 implies u1.name=u2.name)
-----
context u1:User inv V8_nameIsKey: User.allInstances->forall(u2 |
  u1.name<>u2.name implies u1<>u2)
-----
context User inv V9_nameIsKey:
  User.allInstances->one(u|u.name=self.name)
-----

-----
context u:User inv V1_noDoubleBorrowings:
  not(u.copy->exists(c1,c2|c1<>c2 and c1.book=c2.book))
-----
context c1:Copy inv V2_noDoubleBorrowings:
  not(Copy.allInstances->exists(c2| c1<>c2 and c1.user=c2.user and
    c1.user<>oclUndefined(Set(User)) and c1.book=c2.book))

```

```
-----  
context b:Book inv V3_noDoubleBorrowings:  
  not(Copy.allInstances->exists(c1,c2| c1<>c2 and c1.user=c2.user and  
    c1.user<>oclUndefined(Set(User)) and  
    c1.book=Set{b} and c2.book=Set{b}))  
-----
```

```
context u:User inv V4_noDoubleBorrowings:  
  u.copy->forall(c1,c2|not(c1<>c2 and c1.book=c2.book))  
-----
```

```
context u:User inv V5_noDoubleBorrowings:  
  u.copy->forall(c1,c2|c1=c2 or c1.book<>c2.book)  
-----
```

```
context u:User inv V6_noDoubleBorrowings:  
  u.copy->forall(c1,c2|c1.book<>c2.book or c1=c2)  
-----
```

```
context u:User inv V7_noDoubleBorrowings:  
  u.copy->forall(c1,c2|c1.book=c2.book implies c1=c2)  
-----
```

```
context u:User inv V8_noDoubleBorrowings:  
  u.copy->forall(c1,c2|c1<>c2 implies c1.book<>c2.book)  
-----
```

```
context u:User inv V9_noDoubleBorrowings:  
  Copy.allInstances->forall(c1,c2|c1.book=c2.book implies c1=c2)  
-----
```

```
-----  
context b:Book inv V1_authSeqExistsAndUnique: b.authSeq->size()>0 and  
  Set{1..b.authSeq->size()-1}->forall(i|  
    Set{i+1..b.authSeq->size()}->forall(j|  
      authSeq->at(i)<>authSeq->at(j)))  
-----
```

```
context b:Book inv V2_authSeqExists:  
  b.authSeq->size()>0  
context b:Book inv V2_authSeqUnique:  
  Set{1..b.authSeq->size()-1}->forall(i|  
    Set{i+1..b.authSeq->size()}->forall(j|  
      authSeq->at(i)<>authSeq->at(j)))  
-----
```

```
context b:Book inv V3_authSeqExists:  
  b.authSeq->notEmpty()  
context b:Book inv V3_authSeqUnique:  
  Set{1..b.authSeq->size()}->forall(i|  
    Set{1..b.authSeq->size()}->forall(j|  
      i<>j implies authSeq->at(i)<>authSeq->at(j)))  
-----
```

```
context b:Book inv V4_authSeqExists:  
  b.authSeq->exists(a|true)  
context b:Book inv V4_authSeqUnique:  
  Set{1..b.authSeq->size()-1}->forall(i|  
    Set{i+1..b.authSeq->size()}->forall(j|  
      i<>j implies authSeq->at(i)<>authSeq->at(j)))  
-----
```

```
context b:Book inv V5_authSeqExists:  
  b.authSeq->asSet()->notEmpty()  
context b:Book inv V5_authSeqUnique:  
  Set{1..b.authSeq->size()}->forall(i|  
    Set{1..b.authSeq->size()}->forall(j|  
      authSeq->at(i)<>authSeq->at(j)))  
-----
```

```

-----
context b:Book inv V6_authSeqExists:
  b.authSeq->one(a|true)
context b:Book inv V6_authSeqUnique:
  Set{1..b.authSeq->size()-1}->forall(i|
    authSeq->at(i)<>authSeq->at(i+1))
-----
context b:Book inv V7_authSeqExists:
  b.authSeq->reject(a|a<>a)->size()>=1
context b:Book inv V7_authSeqUnique:
  b.authSeq->isUnique(a|a)
-----
context Book inv V8_authSeqExistsAndUnique: authSeq->size()>0 and
  Set{1..authSeq->size()-1}->forall(i|
    Set{i+1..authSeq->size()}->forall(j|
      authSeq->at(i)<>authSeq->at(j)))
-----
context Book inv V9_authSeqExistsAndUnique: self.authSeq->size()>0 and
  Set{1..self.authSeq->size()-1}->forall(i|
    Set{i+1..self.authSeq->size()}->forall(j|
      self.authSeq->at(i)<>self.authSeq->at(j)))
-----

```

References

[Gogolla et al., 2007] Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27-34, 2007.

[Richters and Gogolla, 2001] Mark Richters and Martin Gogolla. OCL - Syntax, Semantics and Tools. In Tony Clark and Jos Warmer, editors, *Advances in Object Modelling with the OCL*, pages 43-69. Springer, Berlin, LNCS 2263, 2001.