Avoiding OCL specification pitfalls

Teaching Software Modeling by Using Constraints

Dan CHIOREAN, Ileana OBER and Vladiela PETRASCU

Teaching Software Modeling by Using Constraints Motivation

- MDE technologies requires **a clear and complete model specification**
- Design by Contracts **supports** producing rigorous models
- Using this technique is not so widespread as expected
  - this state of facts was caused by different rationales
    - people are not yet convinced about the advantages that can be obtained
      - => they should be motivated about using constraints
      - this shape their mentality

Teaching Software Modeling by Using Constraints Our proposal

- To prove the necessity of employing constraints, by means of relevant examples
- To presents best principles &
- valid practices for specifying constraints

 Our approach is focused on the:

✔ role of complete and unequivocal requirements,

✔ importance of the rigor in specifying constraints,

✔ specification process that supports the return in earlier phases, including requirements

 And highlights on the:

✔ conformance between requirements & specifications,

✔ importance of choosing a design model from different proposals,

✔ need of testing specifications by using snapshot

Teaching Software Modeling by Using Constraints
OCL specifications – state of facts

- very good books, articles and slides about OCL: Warmer, Gogolla, Hussmann, Demuth, Atkinson, etc
- many OCL examples (including the UML static semantics specification) are hasty,
- avoiding hasty specification through some best practice & principles

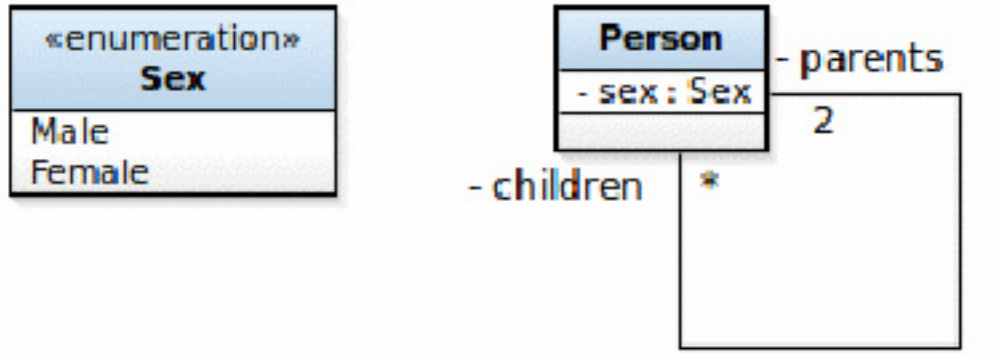
 Aspects related to OCL specs

- ✔ usefulness of information when constraints are broken
- ✔ intelligibility and suggestiveness of specification
- ✔ use of constraint specification patterns
- ✔ conformity between evaluation of constraints specified:
  - in OCL and
  - in programming languages (generated from specs)
- ✔ the independence of the results obtained from the OCL tools used

Teaching Software Modeling by Using Constraints

Understanding model semantics

- in MDD technologies models are used to produce software




```
self.parents->asSequence()->at(1).sex <>
self.parents->asSequence()->at(2).sex
context Person
   inv parentsSex:
      self.parents->size = 2 implies
      self.parents->first.sex = Sex::female and
      self.parents->last.sex = Sex::male
```

Teaching Software Modeling by Using Constraints Understanding model semantics_2

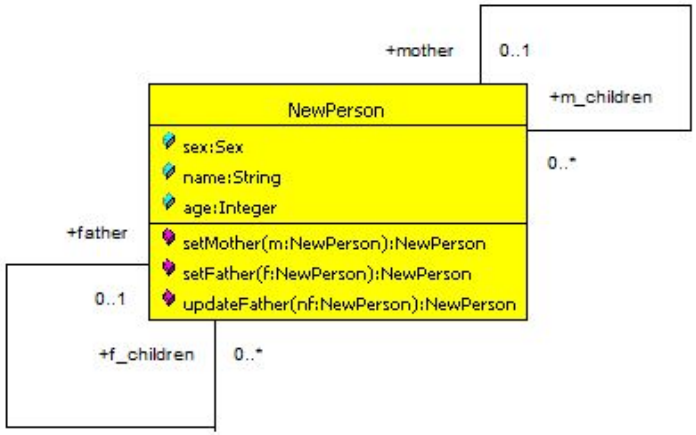

```
context Person
   inv notSelfParent:
      self.parents->select(p | p = self)->isEmpty
context Person
   inv parentsAge:
      self.parents->reject(p|p.age - self.age >= 16)->isEmpty
context Person::addChildren(p:Person)
   pre childrenAge:
      self.children->excludes(p) and self.age - p.age >= 16
   post chidrenAge:
      self.children->includes(p)
```

Teaching Software Modeling by Using Constraints Understanding model semantics 2

To avoid comparisons involving undefined values, whose results may vary with the OCL-supporting tool used, the equality tests of the parents' sex with the corresponding enumeration literals should be conditioned by both being specified.

```
context Person
  inv parentsSexP2 :
    self.parents -> size () = 2 implies
    ( let mother = self . parents -> first () in let father = self.parents -> last () in
    if ( not mother.sex. oclIsUndefined () and not father.sex. oclIsUndefined ())
      then mother.sex = Sex :: Female and father.sex = Sex :: Male
      else false
    endif
    )
```

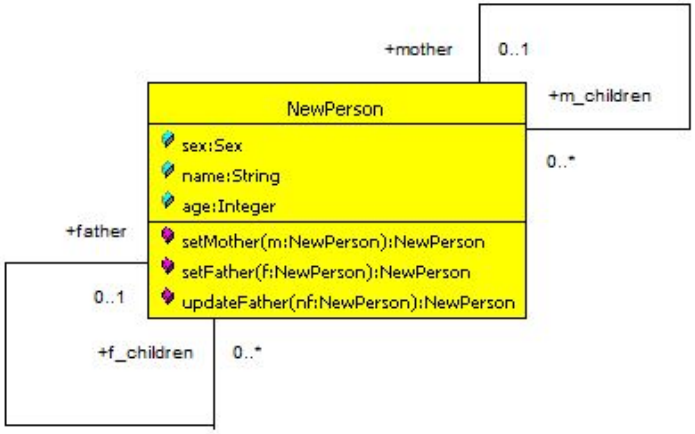
Teaching Software Modeling by Using Constraints Modeling Alternatives



Within this model version, the constraint regarding the parent' sex can be stated as:

context NPerson
  inv parentsSexP1 :
    (**self**.mother -> size () = 1 **implies** Sex :: Female = **self**.mother.sex) **and**
    (**self**.father -> size () = 1 **implies** Sex :: Male = **self**.father.sex)

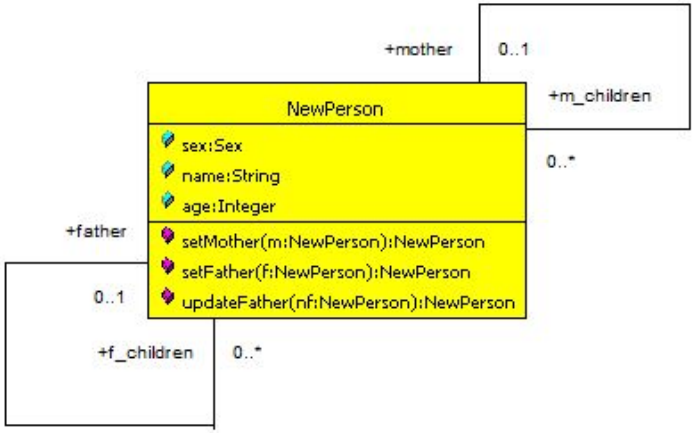
Teaching Software Modeling by Using Constraints Modeling Alternatives_2



An alternative invariant shape, providing the same evaluation result in both OCLE and Dresden OCL, for both OCL and Java code is the one below.

```
context NPerson
  inv parentsSexP2 :
    ( self.mother -> size () = 1  implies
      ( let ms: Sex = self.mother.sex in
        if not ms.oclIsUndefined ()  then ms = Sex :: Female
                                     else false endif )
    ) and ( self.father -> size () = 1  implies
      ( let fs: Sex = self.father.sex in
        if not fs.oclIsUndefined ()  then fs = Sex :: Male
                                     else false endif )
    )
```

Teaching Software Modeling by Using Constraints Modeling Alternatives_3



With respect to the parentsAge invariant, we propose the following specification:

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

If we were to follow the classical specification patterns available in the literature, the invariant would have looked as follows.

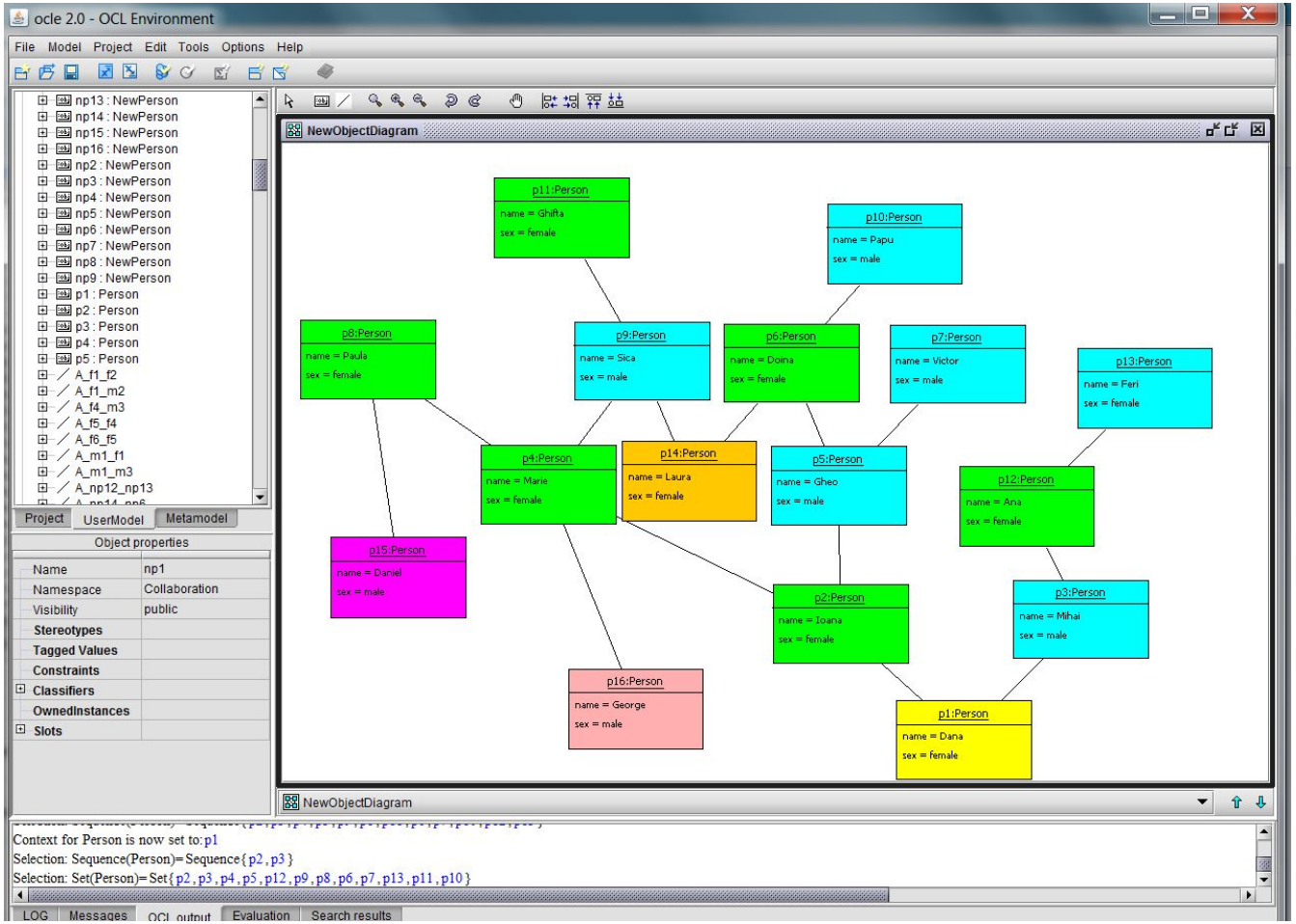
context NPerson
  inv parentsAgeL :
    self.mChildren -> forAll (p | **self**.age - p.age >= 16) **and**
    self.fChildren -> forAll (p | **self**.age - p.age >= 16)

Teaching Software Modeling by Using Constraints Explaining the Intended Model Uses

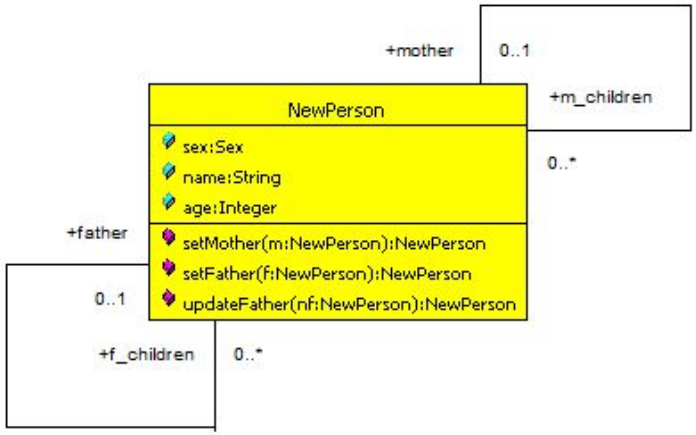
The result obtained when querying

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

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

Teaching Software Modeling by Using Constraints
Querying the initial model



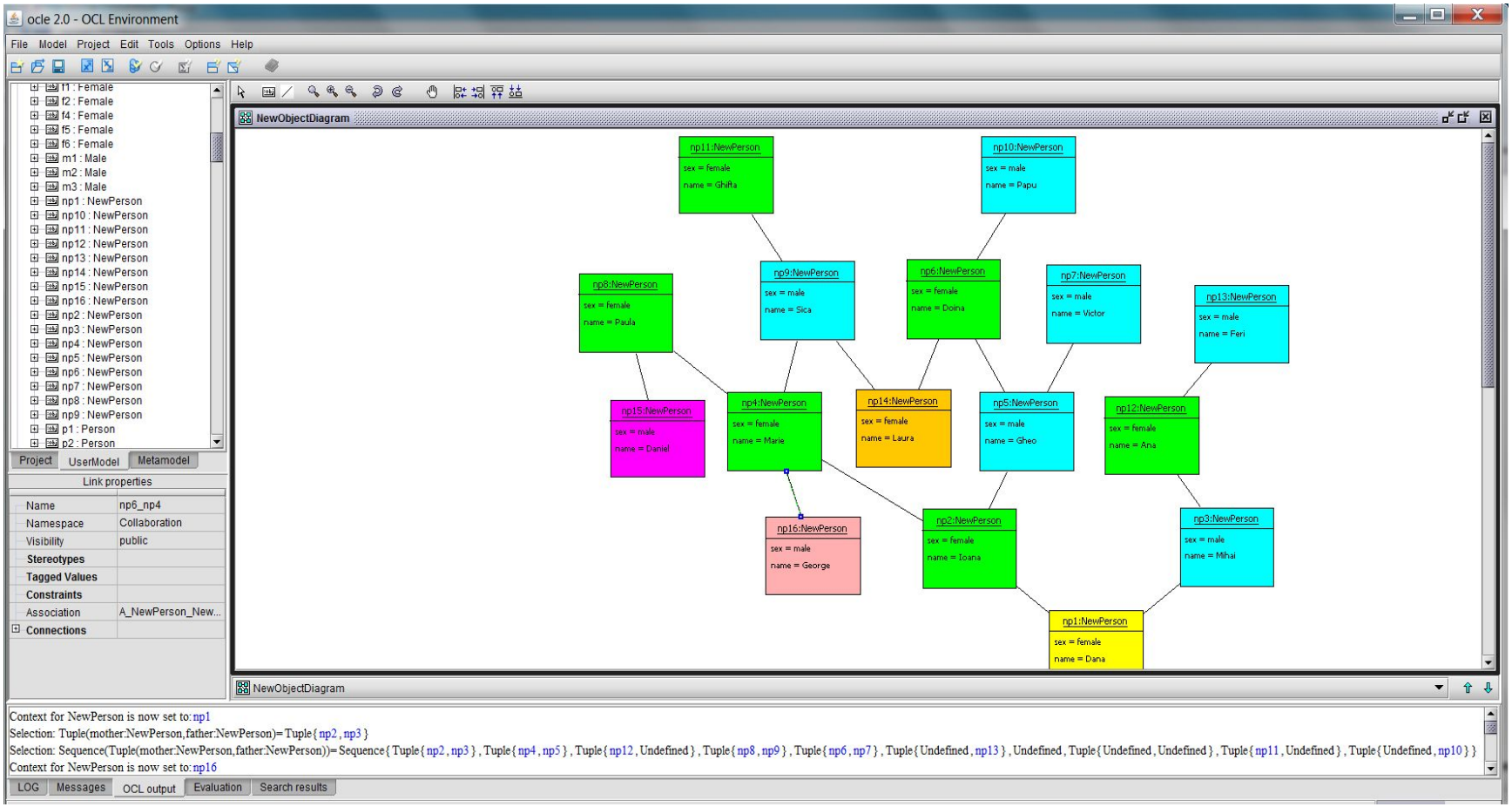
Teaching Software Modeling by Using Constraints Explaining the Intended Alternative Model Uses



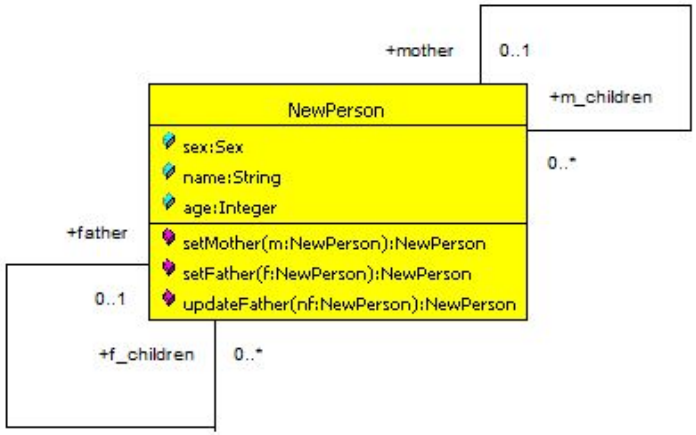

```
context NewPerson
   def parents:TupleType(mother:Nperson, father:NPerson) =
       Tuple{mother = self.mother, father = self.father}

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

Teaching Software Modeling by Using Constraints

The result obtained when querying the NewPerson tree



Teaching Software Modeling by Using Constraints Explaining the Intended Alternative Model Uses

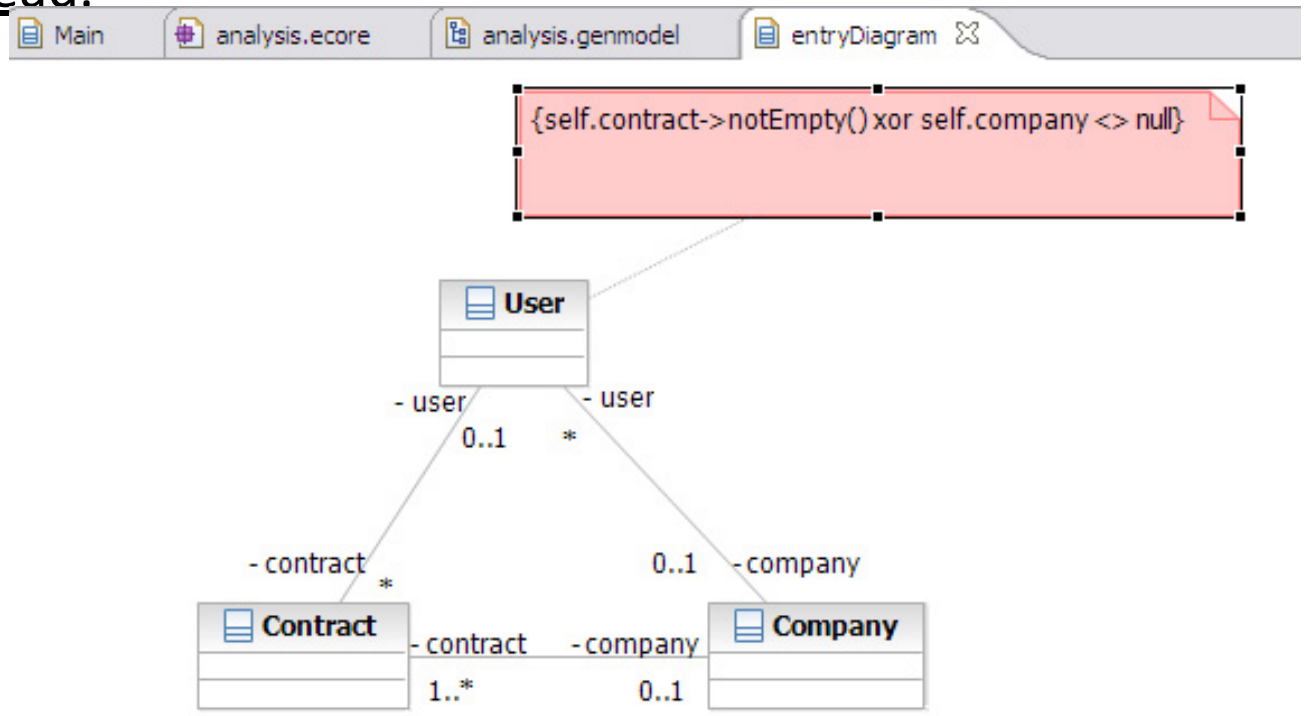



```
context NPerson
  def children : Set( NPerson ) =
    if self sex = Sex :: Female
      then self.m_children
      else  self. f_children
    endif
```

Teaching Software Modeling by Using Constraints

Modeling requires **understanding the problem and the problem domain**

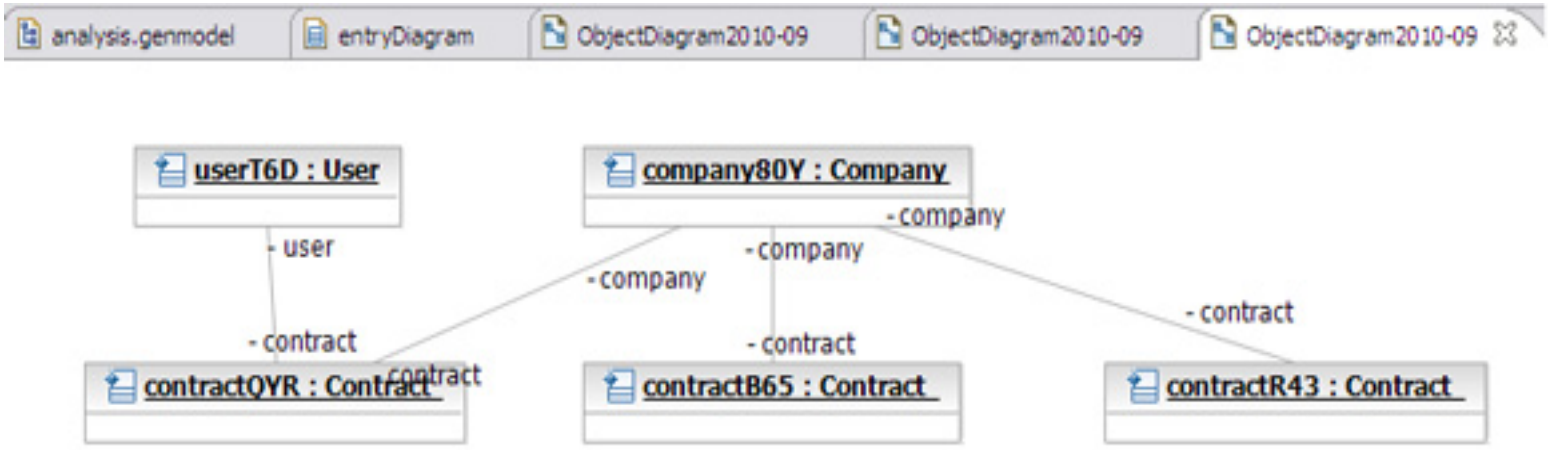
- "… the library offers a subscription to each person employed in an associated company. In this case, the employee does not have a contract with the library but with the society he works for, instead."



Teaching Software Modeling by Using Constraints

Understanding the concepts and improving the requirements

- snapshots must clarify requirements and the model

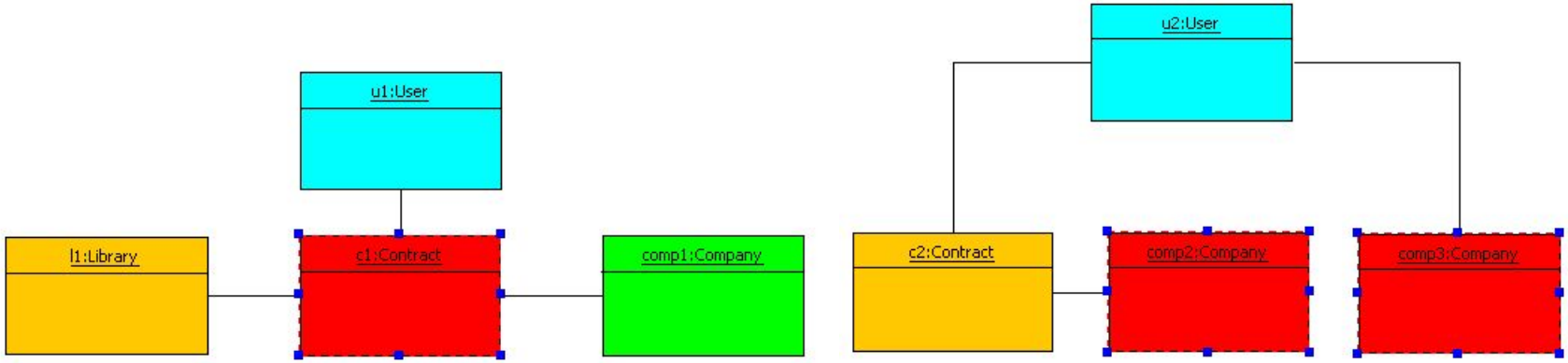



```
context User
   inv TodConstraint:
      self.contract->notEmpty() xor self.company <> null
```

Teaching Software Modeling by Using Constraints

Using suggestive snapshots to test specifications

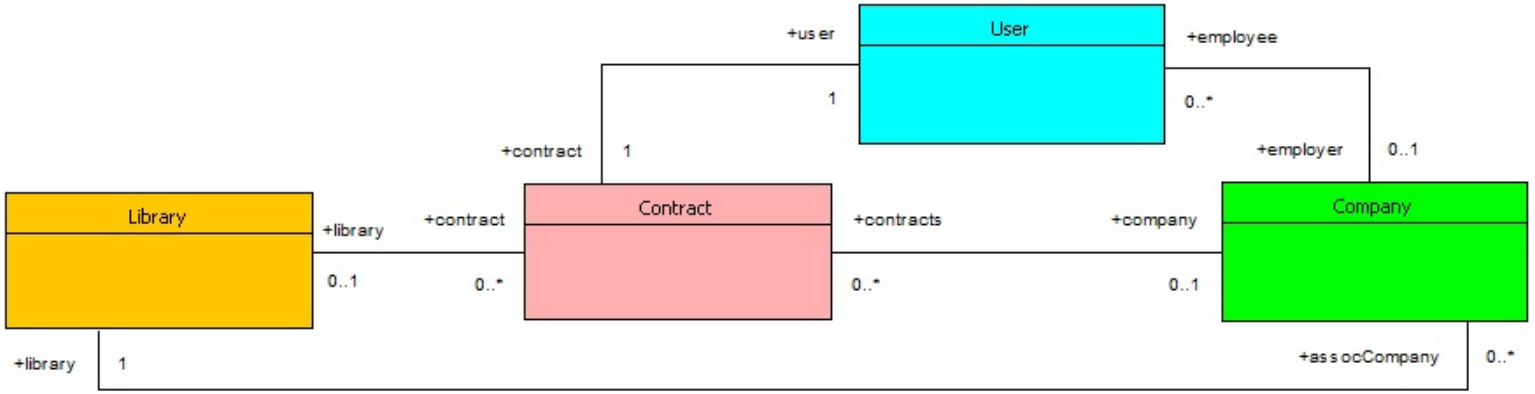
- Unwanted model instantiations that are not cached by the invariant proposed in [Tod11]




```
context User
   inv TodConstraint:
      self.contract->notEmpty() xor self.company <> null
```

Teaching Software Modeling by Using Constraints Understanding and improving the requirements_3

- Improving the solution proposed by choosing an appropriate context & updating constraints,




```
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
```

Teaching Software Modeling by Using Constraints Conclusion

- Complementing model specification with constraints is crucial. This supports:
  - a rigorous model description and
  - detection of unwanted model instantiation
- Conformance with requirements is the first criterion quality criterion in constraint specification
  - This requires a thorough understanding of the problem and problem domain
- Abstraction is the first principle that have to be applied when modeling because identify the main concepts to use
- Identifying and specifying constraints follow abstraction because restricts on relationships between concepts and on their values

Teaching Software Modeling by Using Constraints Conclusion_2

- The main attitude in specifying constraints is the rigor
- In our opinion, hastiness is the main enemy in the constraint specification process
- The quality of constraint specification includes also different other criteria
- Requirements must include the description of the intended usage of the system
- The constraints role is to support a better quality of the model
- When judging the quality of the model, all descriptions including constraints must be considered
- Choosing the most appropriate model, supposes analyzing many proposals

Teaching Software Modeling by Using Constraints References

14. 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/