Testing-Oriented Improvements of OCL Specification Patterns

Dan CHIOREAN[1], Vladiela PETRASCU[1], Ileana OBER[2]

[1]Babeş-Bolyai University, Cluj-Napoca
[2]Paul Sabatier University, Toulouse

AQTR 2010 - Cluj-Napoca – 30 May

Presentation Overview

- Introduction
- Background & Related Work
- Our Approach:
  - Proposed OCL specification patterns
    - The case of invariants
    - The case of pre/post conditions
  - Tool support & Validation
- Conclusions and future work
- References

Introduction

- Detailed and unequivocal model specifications are a prerequisite for attaining the automated software development goal as promoted by the Model Driven Engineering paradigm.
- Obtaining as much information as possible, about causes of constraint failure, goes beyond testing requirements. It is intimately related to the reasons of using constraints in modeling and, more general, in software specification.
- The four main advantages of using the Design by Contract technique [1] are:
  1. help in writing correct software,
  2. documentation aid,
  3. support for testing, debugging and quality assurance,
  4. support for software fault tolerance.

Background & Related Work

- In software modeling, *constraint patterns* [3] may be used to capture frequently occurring restrictions imposed on models. This paper is focused on describing and specifying such constraint patterns.
- We will clearly distinguish among the concepts of *constraint pattern* and that of *OCL specification pattern*, although the phrases appear to be used interchangeably in the literature ([6], [2]).
  - a *constraint pattern* denotes a logical constraint/restriction on a model
  - an *OCL specification pattern* refers to a proposed way of specifying constraints
- We describe solutions to the *constraint patterns* of interest as *OCL specification patterns*
- The OCL specifications found in the literature, including those for constraint patterns, are only focused on expressions' clearness. During testing and debugging however, just knowing that a system state is inconsistent or that a method pre/postcondition is not fulfilled is not enough

Background & Related Work - 2

- Identifying the exact failure reasons is of utmost importance for error correction. This is the core-idea of the OCL specification approach that we promote. In this respect, in the following **we give improvements of existing OCL specification patterns, considering both the case of invariants and that of pre/post-conditions**

Our Approach - Proposed OCL specification patterns -

The case of invariants

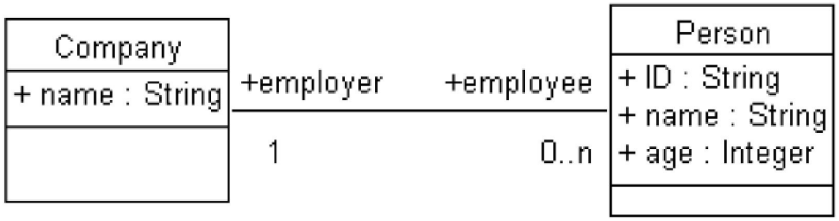


Figure 1 – A simple class model

- "All employees of a company should be aged at most 65". We will refer to this particular constraint as *Complying with Retirement Age Limit (CRAL)*

- **Existent specifications:**

```
context Company
    inv CRAL_E:
        self.employee->forAll( e | e.age <= 65)
```

- **Drawback**: we have no useful hint regarding the identity of those employees which are over the age limit!

Our Approach - Proposed OCL specification patterns -

The case of invariants - 2

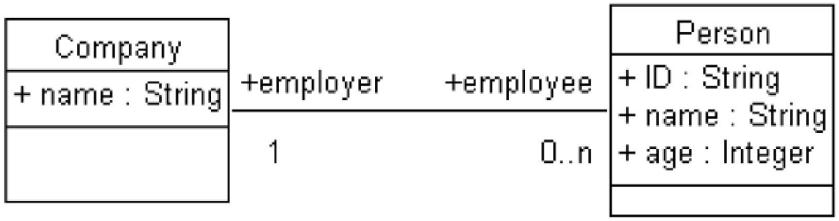


Figure 1 – A simple class model

- "All employees of a company should be aged at most 65". We will refer to this particular constraint as *Complying with Retirement Age Limit (CRAL)*

- **Our proposal** is convenient not only when modeling, but also at runtime:

```
context Company
   inv CRAL_P:
      self.employee->reject( e | e.age <= 65)->isEmpty()
```

Our Approach - Proposed OCL specification patterns -

The case of invariants - 3

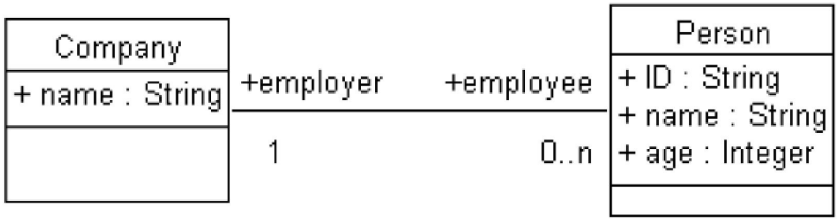


Figure 1 – A simple class model

- Two equivalent OCL specification patterns for the forAll modeling constraint:

```
pattern ForAll_Reject(objects: Collection(Object), properties: Set(Constraint))=
        objects->reject(y | oclAND(properties, y))->isEmpty()
```

- The CRAL_P specification is an instantiation of the ForAll_Reject specification pattern:

```
context Company
   inv CRAL_P:
       ForAll_Reject(self.employee, Set{AttributeValueRestriction(age, <=, 65)})

pattern ForAll_Select(objects: Collection(Object), properties: Set(Constraint))=
        objects->select(y | not oclAND(properties, y))->isEmpty()
```

Our Approach - Proposed OCL specification patterns -

The case of invariants - 4

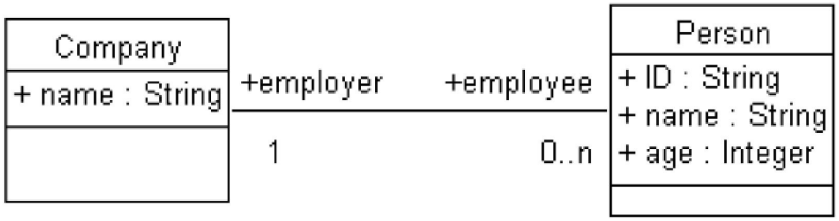


Figure 1 – A simple class model

- **"Global" uniqueness case (*GUID*):** Let us consider an application whose model contains a Person class having the same attributes as the one in Figure 1. Almost all the OCL specification proposals existent in the literature have one of the following shapes:

```
context Person
   inv GUID_E1:
      Person.allInstances()->forAll(p, q| p<>q implies p.ID <> q.ID)
context Person
   inv GUID_E2:
      Person.allInstances()->isUnique(ID)
```

- **Drawbacks**: both GUID_E1 and GUID_E2 break the semantics of invariants. Both specifications do not provide appropriate debugging support!

Our Approach - Proposed OCL specification patterns -

The case of invariants - 5

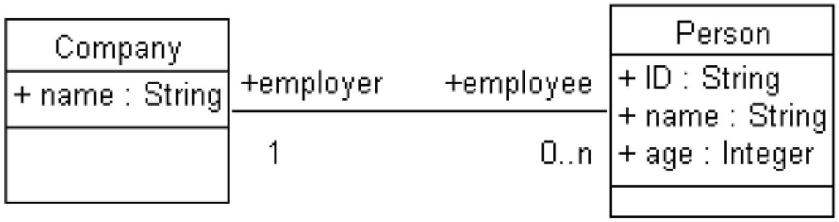


Figure 1 – A simple class model

```
context Person
   inv GUID_P:
      Person.allInstances()->select(p | p.ID = self.ID)->size() = 1
```

- The pattern corresponding to this specification is:

```
pattern GloballyUniqueIdentifier(class: Class, attribute: Property)=
        class.allInstances()->select(i | i.attribute = self.attribute)->size() = 1
```

- **"Container-relative" uniqueness case (*CUID*):** in the context of Figure 1, supposes a constraint requiring that employees of a should be uniquely identified by their IDs:

```
context Company
   inv CUID_P1:
      self.employee->reject(e | self.employee.ID->count(e.ID)=1)->isEmpty()
```

Our Approach - Proposed OCL specification patterns -

The case of invariants - 6

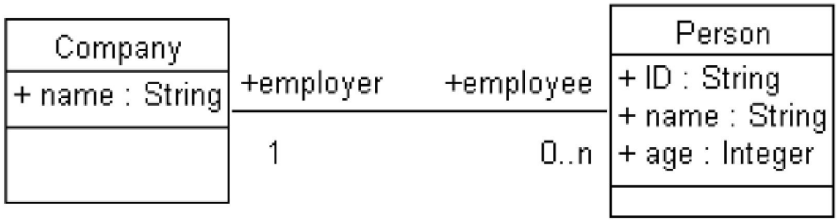


Figure 1 – A simple class model

- A more efficient specification:

```
context Company
   inv CUID_P2:
      let allIDs:Bag(String) = self.employee.ID in
      self.employee->reject(e | allIDs->count(e.ID)=1)->isEmpty()
```

- with the corresponding pattern:

```
pattern ContainerRelativeUniqueIdentifier(class:Class, navigation:Property,
   attribute:Property)=
   let bag:Bag(OclAny) = self.navigation.attribute in
   ForAll_Reject(navigation, UniqueOccurenceInBag(bag, attribute))
```

- where, UniqueOccurenceInBag, shortly UOB is a new atomic pattern requiring that "A given element has exactly one occurrence in a given bag of elements of the same type"

Our Approach - Proposed OCL specification patterns -

The case of pre/post conditions

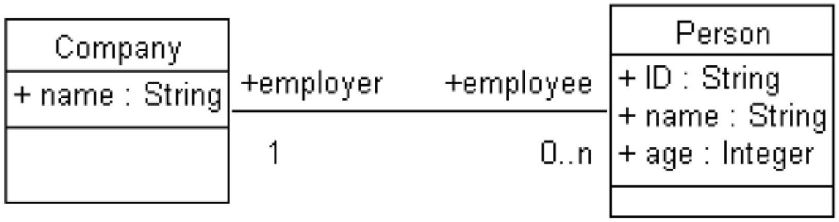


Figure 1 – A simple class model

- Using pre/post-conditions represents the correct attitude to adopt from a dynamic perspective "it is better to prevent than to cure". Breaking the uniqueness constraint imposed on IDs of employees can be prevented by means of appropriate pre/post-condition pairs for the modifiers that may violate this constraint. The modifiers concern adding an employee to a company and setting the name of an employee, respectively.

```
context Company::addEmployee(p:Person)
   pre CUID_preAdd:
      self.employee->reject(e | e.ID <> p.ID)->isEmpty()

post CUID_postAdd:
      self.employee = self.employee@pre->including(p)
```

Our Approach - Proposed OCL specification patterns -

The case of pre/post conditions - 2

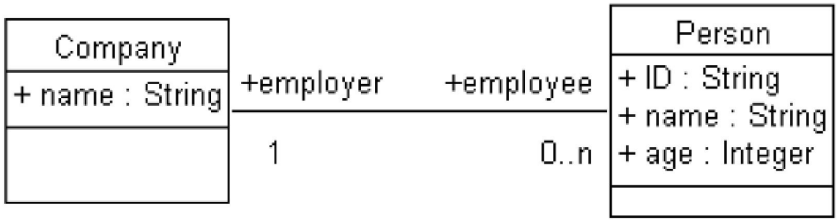
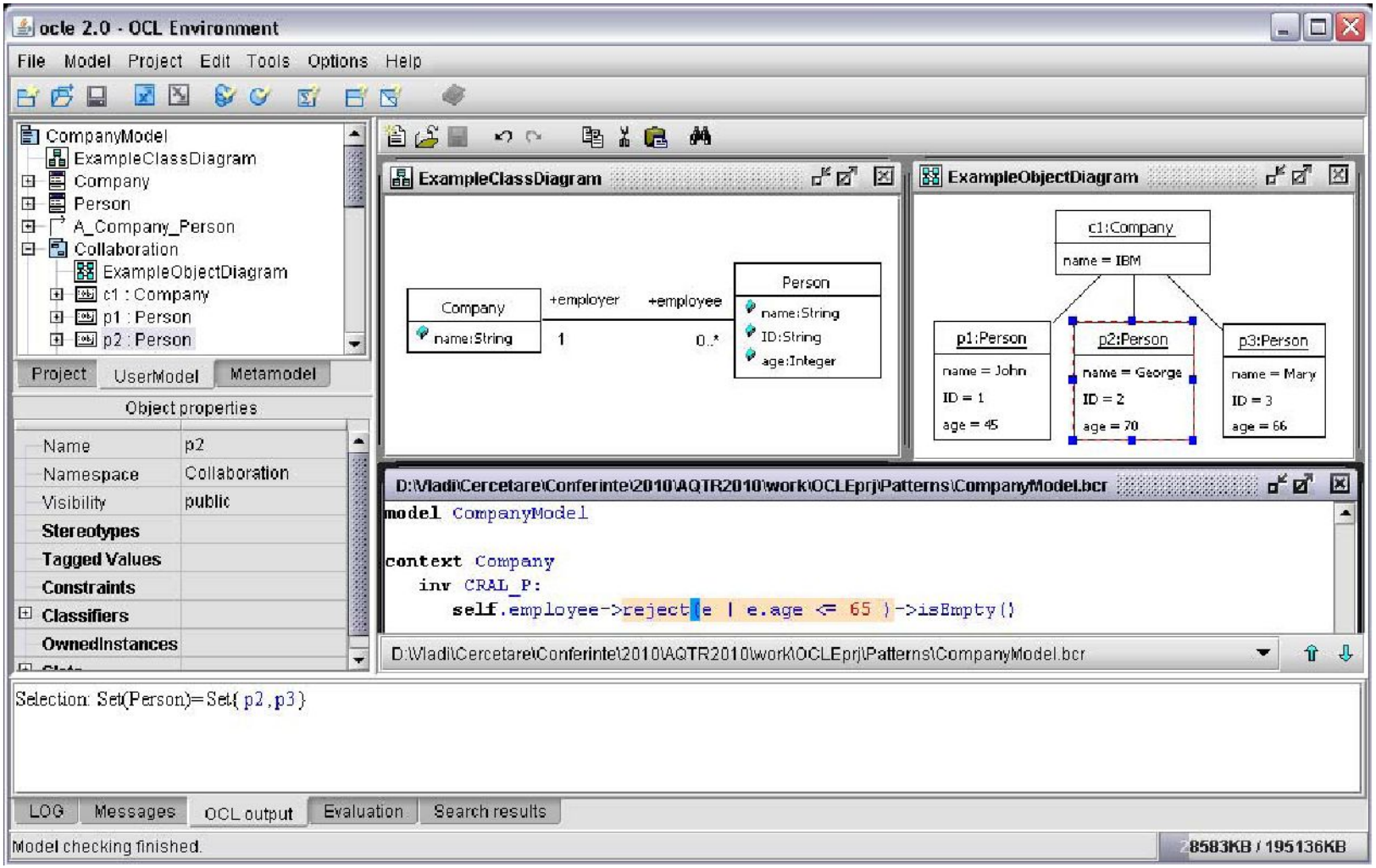


Figure 1 – A simple class model

```
context Person::setID(id: String)
   pre CUID_preSet:
        self.employer.employee->reject(e | e.ID <> id)->isEmpty()

   post CUID_postSet:
         self.ID = id
```

Our Approach - Tool support & Validation



Conclusions and future work

- Related to the best known approaches in the field [6], [7], [3], [2], our contribution consists of:
  1. proposal of a pair of "efficient testing-oriented" OCL specification patterns for the *For All* constraint pattern;
  2. a deeper analysis of the *Unique Identifier* constraint pattern, with respect to the particular type of uniqueness imposed (global vs. container-relative);
  3. Proposal of correct/appropriate OCL specifications patterns for each uniqueness context;
  4. approaching the constraint patterns' problem from both a static and a dynamic perspective, with appropriate specification patterns for each case;
  5. Validation of our proposed approach using appropriate tool-support.
- To our knowledge, this is the only approach to constraint patterns' specification aimed at maximizing the amount of relevant testing/debugging related information

Conclusions and future work - 2

- Future work targets at:
  1. identifying new constraint and OCL specification patterns, along with improving some of the existing ones;
  2. automating the instantiation of proposed patterns by means of an appropriate tool;
  3. developing an automated test-data generator;
  4. a detailed study on run-time exception handling.
- This work was supported by the research project ID 2049, sponsored by the Romanian National University Research Council (CNCSIS)

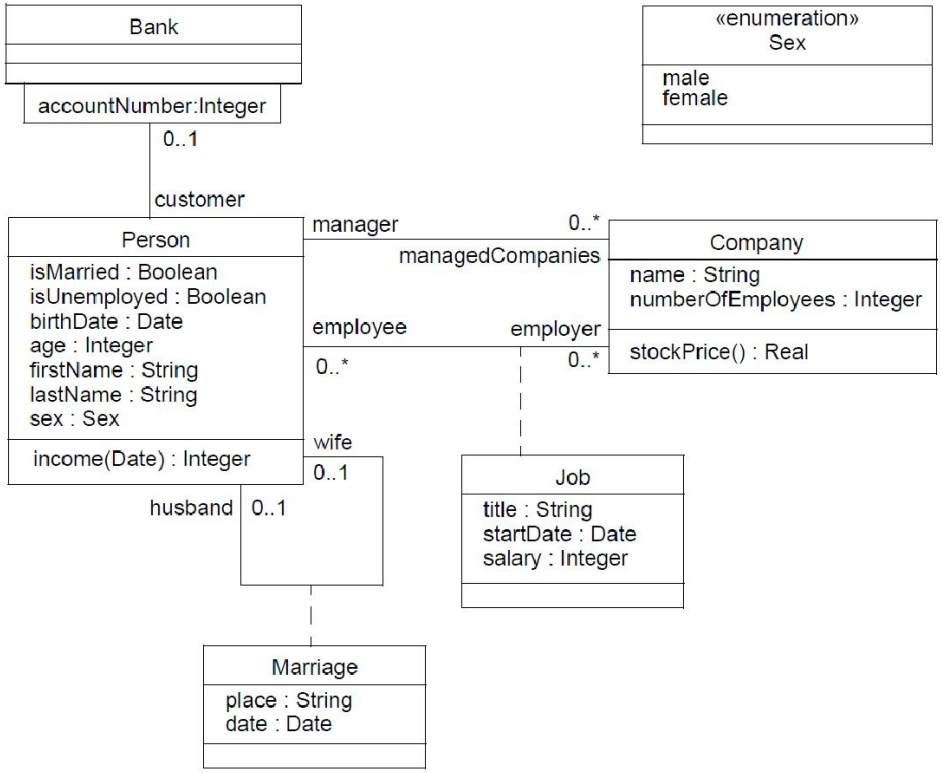
References

- [1] B. Meyer, *Object-Oriented Software Construction, 2nd ed. Prentice* Hall, 1997.
- [2] M. Wahler, "Using Patterns to Develop Consistent Design Constraints," Ph.D. dissertation, ETH Zurich, Switzerland, 2008. [Online]. Available: http://e-collection.ethbib.ethz.ch/eserv/eth:30499/eth-30499-02.pdf
- [3] M. Wahler, J. Koehler, and A. D. Brucker, "Model-Driven Constraint Engineering," *Electronic Communications of the EASST, vol. 5, ISSN* 1863-2122, 2006.
- [4] OMG, "Unified Modeling Language: Superstructure Version 2.2," 2009. [Online]. Available: http://www.omg.org/docs/formal/09-02-02.pdf
- [5] ——, "OCL Specification v2.0," 2006. [Online]. Available: http://www.omg.org/spec/OCL/2.0/PDF
- [6] J. Ackermann, "Frequently Occurring Patterns in Behavioral Specification
- of Software Components," in *COEA, 2005, pp. 41–56.*
- [7] ——, "Formal Description of OCL Specification Patterns for Behavioral Specification of Software Components," in *Proceedings of the MoDELS'*

References - 2

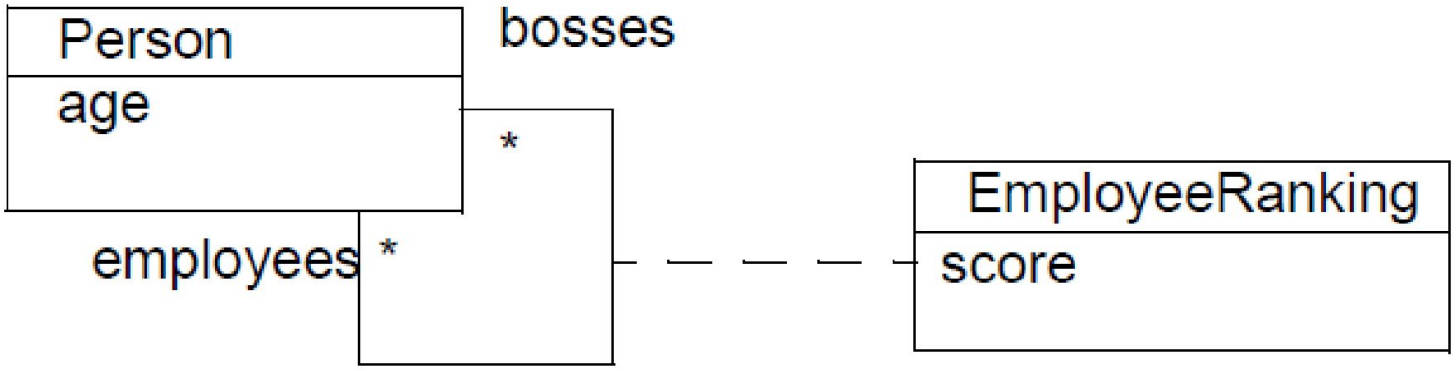
- [7] ——, “Formal Description of OCL Specification Patterns for Behavioral Specification of Software Components,” in *Proceedings of the MoDELS’ 05 Conference Workshop on Tool Support for OCL and Related Formalisms - Needs and Trends, Montego Bay, Jamaica, October 4, 2005, ser. Technical Report LGL-REPORT-2005-001, T. Baar, Ed.* EPFL, 2005, pp. 15–29.
- [8] E. Miliauskaite and L. Nemuraite, “Representation of Integrity Constraints in Conceptual Models,” *Information Technology and Control,* vol. 34, no. 4, pp. 355–365, 2005.
- [9] D. Costal, C. G´omez, A. Queralt, R. Ravent´os, and E. Teniente, “Facilitating the Definition of General Constraints in UML,” in *MoDELS,* 2006, pp. 260–274.
- [10] LCI, “Object Constraint Language Environment (OCLE),” 2005. [Online]. Available: http://lci.cs.ubbcluj.ro/ocle/

Qualified associations



context Bank **inv**:
self.customer[8764423]

Navigating recursive association classes

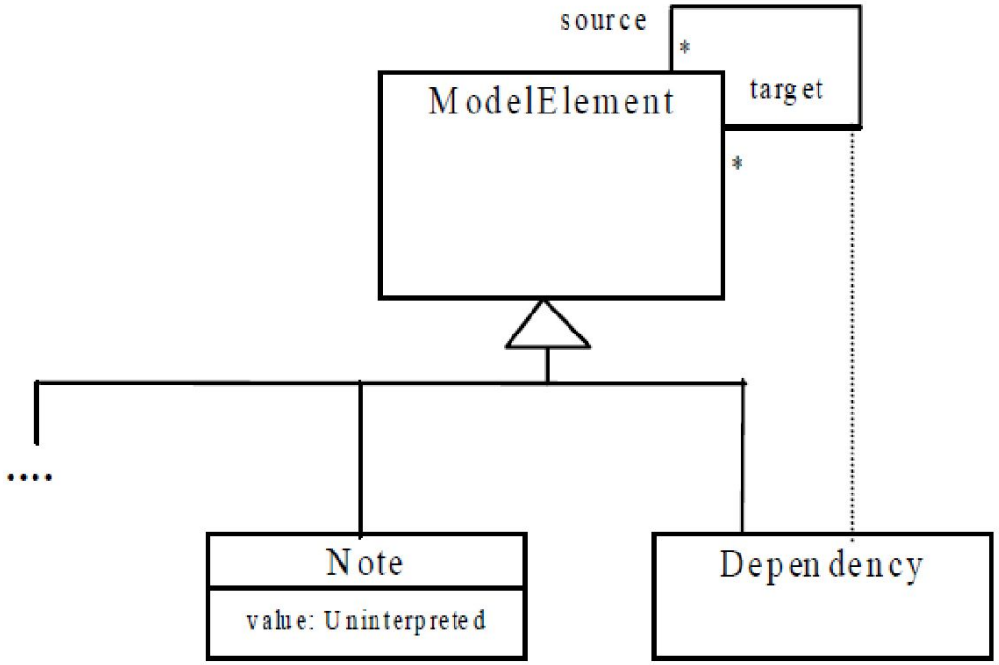



```
context Person inv:
self.employeeRanking[bosses]->sum() > 0

context Person inv:
self.employeeRanking[employees]->sum() > 0

context Person inv:
self.employeeRanking->sum() > 0 -- INVALID!
```

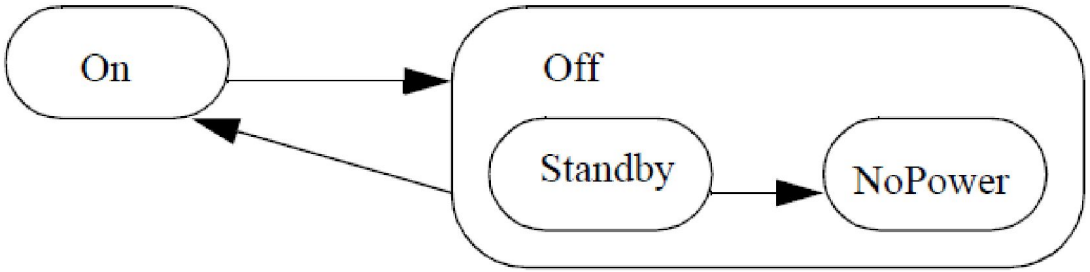
Accessing overridden properties of supertypes




```
context Dependency
inv: self.oclAsType(Dependency).source <> self

context Dependency
inv: self.source <> self  is an ambigue specification
```

Accessing overridden properties of supertypes



The operation *oclInState(s)* results in true if the object is in the state *s*. Values for *s* are the names of the states in the statemachine(s) attached to the Classifier of *object*. For nested states the statenames can be combined using the double colon '::'

```
object.oclInState(On)
object.oclInState(Off)
object.oclInstate(Off::Standby)
object.oclInState(Off:NoPower)
```

Previous Values in Postconditions

- The value of a property in a postcondition is the value upon completion of the operation. To refer to the value of a property at the start of the operation, one has to postfix the property name with the keyword '***@pre***':

- **`context`** `Person::birthdayHappens()`
- **`post:`** `age = age@pre + 1`
- object.oclIsNew() : Boolean

`oclIsNew()` in Postconditions

- `object.oclIsNew() : Boolean`

- Can only be used in a postcondition.
- Evaluates to true if the *object* is created during performing the operation. That is it didn't exist at precondition time.