CIS 771: Software Specifications

Lecture 13: Mining UML Class Diagrams for OCL Invariants

Copyright 2001-2002, Matt Dwyer, John Hatcliff, and Rod Howell. The syllabus and all lectures for this course are copyrighted materials and may not be used in other course settings outside of Kansas State University in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.

Identifying Constraints

- OCL provides a language for defining invariants, but where do those invariants come from?
- Start from class diagram
  - Gives a vocabulary for your invariants
  - Gives very weak invariants, e.g., attribute types and association multiplicities
  - View constraint definition as enriching the class diagram
- What structural aspects of a class diagram suggest the need for a constraint?
Class Diagrams Include...

- Subclass relationships
- Attributes
- Operations
- Associations with multiplicities
- Derived attributes/associations
- Certain relationships between associations

Person

- age : Integer
- ancestors() : Set(Person)
- married() : Boolean
- siblings() : Set(Person)

Man

- married() : Boolean
- spouse : 0..1
- husband : 0..1

Woman

- married() : Boolean
- spouse : 0..1
- wife : 0..1

Children: *

Offspring: 0..2

Parents: 0..2

A person can have children, which are offspring. They can also have ancestors and siblings. A man and a woman can be married and have a spouse. A man can have a brother-in-law and a woman can have a sister-in-law. Parties can order food and be served. The number of parties can vary, and each party can order food.
Advanced UML Features

- We haven’t studied these UML features in detail
- Their semantics can be captured by OCL invariants

- Derived InLaw association
  
  ```
  association InLaw between
  Man[*] role BrotherInLaw
  Woman[*] role SisterInLaw
  end
  ```
Advanced UML Features

context w:Woman
inv BrotherInLaw:
  w.brotherInLaw =
  w.husband.parents.children->
  select(p | p.oclIsKindOf(Man) and
  p <> w.husband)->
  union(
    w.parents.children->
    select(s | s.oclIsKindOf(Woman) and
    s <> w)->
    collect(s | s.oclAsType(Woman).husband)) as Set

Detailed Models...

```
Guitar 0..1
  ElectricGuitar 0..1
  ClassicGuitar 0..1

GuitarString 0
  MetalString
  PlasticString

{subset}
{subset}
```
..vs. Constraints

<table>
<thead>
<tr>
<th>Guitar</th>
<th>0..1</th>
<th>*</th>
<th>GuitarString</th>
</tr>
</thead>
<tbody>
<tr>
<td>isElectric : Boolean</td>
<td></td>
<td>strings</td>
<td>isMetal : Boolean</td>
</tr>
</tbody>
</table>

context Guitar

inv ElectricHaveMetal:

isElectric implies strings->forAll(s | s.isMetal)

inv ClassicHavePlastic:

not isElectric implies strings->forAll(s | not s.isMetal)

Types of Constraints

- Some constraints can be completely specified by the class diagram
  - E.g., abstract classes, subclasses, multiplicities
- Other constraints may be specified only partially or not at all by the class diagram
  - In what follows, we will focus on identifying these constraints
Several examples that we have looked at equate identity (in terms of some sort of ID) with equality

Students and faculty have unique IDs

```plaintext
context Person
inv UniqueIds:
    Person.allInstances ->
    forAll (p1, p2 | p1.id <> p2.id implies p1 <> p2)
```

In general, we should ask the question, “Should equality imply identity?”

In some cases, the answer is “No”

- Two people may have the same name
- Two people should not have the same Social Security Number

One way to identify uniqueness is to look for “sharing”

- If an instance can be shared (as the image of some relation), then it is not unique
For You To Do (pause here)

- What kinds of “sharing” is present in the
  - Academia model
  - Railroad model
  - GUI model (from midterm)
- State the uniqueness constraints in these models
  - in English
  - in OCL

Optional Multiplicities

- A multiplicity by itself doesn’t always completely specify the size of an association
- Sometimes the values of other attributes or associations constrain the size of a particular association
Example

context Car
inv PassiveRestraintLaw:
   model > 1996 implies restraints->notEmpty

Cycles in Class Diagrams

- Cycles imply that an object is related to itself
- This may be acceptable, unacceptable, or required
- We should always examine cycles to see if constraints need to be added to enforce the desired outcome
- Note that cycles need not be directed
  - Two navigation paths from same class that reach another class
Example

An Extended Example

- Recall the most complex version of the academia example:
  - Person, Student, Graduate, Undergrad, Faculty, Instructor
  - Department
  - Course
  - Id
Academia Class Diagram

Self Teaching
No Teaching While Enrolled

context i:Instructor
inv NoTeachingWhileEnrolled:
i.instructor.oclIsKindOf(Grad) implies
i.coursesTaught->intersection(
i.instructor.oclAsType(Grad).taking)->IsEmpty
No Waiting Unless Enrolled

context c:Course
inv NoWaitingUnlessEnrolled:
c.waitList->notEmpty implies
c.enrolled->notEmpty
Advisor On Committee

color orange context f:Faculty
color orange inv AdvisorOnCommittee:
f.gradCommittees ->
  includesAll(f.advisees)

Here we switch context for the specifying the invariant

Faculty Teach Required Courses
Faculty Teach Required Courses

class d:Department
  inv FacultyTeachReqCourses:
    d.required.taughtBy ->
    forAll(i:Instructor |
      i.instructor.oclIsKindOf(Faculty))

Students Take One Major Course
Students Take One Dept Course

class s:Student
inv OneCourseFromMajor:
  s.taking->
  intersection(s.major.offers)->
  notEmpty()

For You To Do

- What other cycles can you find?
  - There are several more, e.g.,
    - Prerequisites required
    - No retaking courses
    - Required courses from major

- What invariants do they suggest?
  - Code those as OCL invariants and explore your sample model