Fixing-Up Non-executable Operations in UML/OCL schemas

Xavier Oriol, Ernest Teniente & Albert Tort
UNIVERSITAT POLITÈCNICA DE CATALUNYA
{xoriol,teniente}@essi.upc.edu
albert.tort@sogeti.com
UML/OCL Schema Motivating Example

Physician
Name: String

MedicalSpeciality
Name: String

MedicalTeam
Code: String

CriticalTeam

HasSpeciality 1..*
specialization

IsExpertIn 1
* expertise

IsMemberOf 0..*
member

Manages 1..*
manager

Manages 0..1
managedTeam
**UML/OCL Schema Motivating Example**

**context** MedicalTeam inv ManagersAreMembers:
self.member->includesAll(self.manager)
UML/OCL Schema Motivating Example

context MedicalTeam inv ManagersAreMembers:
   self.member->includesAll(self.manager)

context CriticalTeam inv ExclusiveMembership:
   self.member.team->forAll(t | t = self)
**UML/OCL Schema Motivating Example**

**MedicalSpeciality**
- Name: String

**MedicalTeam**
- Code: String

**Physician**
- Name: String

**CriticalTeam**

---

**HasSpeciality**
- 1..* specialization
- *

**IsMemberOf**
- 0..* member
- 1..*

**Manages**
- 1..*
- 0..1

**IsExpertIn**
- 1..*
- *

**context** MedicalTeam **inv** ManagersAreMembers:
self.member->includesAll(self.manager)

**context** CriticalTeam **inv** ExclusiveMembership:
self.member.team->forAll(t | t = self)

**Operation**:
`newCriticalTeam(p: Physician, s: MedicalSpeciality, cd: String)`

**Pre**:
`p.managedTeam->isEmpty()`

**Post**:
`CriticalTeam.allInstances()->exists(c | c.oclIsNew() and c.code = cd and c.expertise = s and c.manager = p)`
UML/OCL Schema Motivating Example

**context** MedicalTeam **inv** ManagersAreMembers:
self.member->includesAll(self.manager)

**context** CriticalTeam **inv** ExclusiveMembership:
self.member.team->forAll(t | t = self)

**Operation:** newCriticalTeam(p: Physician, s: MedicalSpeciality, cd: String)
**Pre:** p.managedTeam->isEmpty()
**Post:** CriticalTeam.allInstances()->exists(c | c.oclIsNew() and c.code = cd and c.expertise = s and c.manager = p)

The operation is not-executable
UML/OCL Schema Motivating Example

context MedicalTeam inv ManagersAreMembers:
self.member->includesAll(self.manager)

context CriticalTeam inv ExclusiveMembership:
self.member.team->forAll(t | t = self)

Operation: newCriticalTeam(p: Physician, s: MedicalSpeciality, cd: String)
Pre: p.managedTeam->isEmpty()
Post: CriticalTeam.allInstances()->exists(c | c.oclIsNew() and c.code = cd and c.expertise = s and c.manager = p and c.member->includes(p))
UML/OCL Schema Motivating Example

**MedicalSpeciality**
- Name: String
- 1..* specialization
- 1 IsExpertIn expertise

**Physician**
- Name: String
- 0..* IsMemberOf member
- 1..* Manages manager
- 0..1 managedTeam

**MedicalTeam**
- Code: String
- 1..* manages
- 0..1 managedTeam

**CriticalTeam**

**context** MedicalTeam **inv** ManagersAreMembers:
self.member->includesAll(self.manager)

**context** CriticalTeam **inv** ExclusiveMembership:
self.member.team->forAll(t | t = self)

**Operation:** newCriticalTeam(p: Physician, s: MedicalSpeciality, cd: String)
**Pre:** p.managedTeam->isEmpty()
**Post:** CriticalTeam.allInstances()->exists(c | c.oclIsNew() and c.code = cd and c.expertise = s and c.manager = p and c.member->includes(p))

The operation is still not-executable
The problem

Given:
Class Diagram + Constraints + Operations

Some operation(s) might not be executable because it might violate some constraint(s)

But it might be possible to fix up such violation by adding some extra effects in the operation
The difficulty

When adding some extra effects, other violations may occur.

Some violations might be impossible to fix.

Some violations might be fixable in several ways.
Our contribution

A method for, given a non-executable operation

Compute all the combinations of missing effects s.t when added in the operation definition, ensures its executability
A set of objects/instances of all Classes/Associations that satisfies all the constraints and the precondition can be obtained via:

Satisfiability Checker, Snapshot Generator, etc
**Operation:** `newCriticalTeam(p: Physician, s: MedicalSpeciality, cd: String)`

**Pre:** `p.managedTeam->isEmpty()`

---

**Information Base state**

- $P = \text{Foreman}$
- $S = \text{Neuro}$
- $cd = \text{‘CTeam’}$

- `HasSpeciality(Foreman, Neuro)`
- `HasSpeciality(Cameron, Neuro)`

- `IsMemberOf(Foreman, Team1)`
- `IsMemberOf(Cameron, Team1)`

- `IsExpertIn(Team1, Neuro)`

- `Manages(Cameron, Team1)`

- **Speciality(Neuro)**

---

**Diagram Elements:**

- **Physician(Foreman)**
- **Physician(Cameron)**
- **MedTeam(Team1)**
Roadmap

Set of insertions/deletions of instances

Can be obtained via:
Imperative Interpretation of Postcondition
**Operation:** `newCriticalTeam(p: Physician, s: MedicalSpeciality, cd: String)`  
**Post:** `CriticalTeam.allInstances()->exists(c | c.oclIsNew() and c.code = cd and c.expertise = s and c.manager = p)`
Logic Rules for detecting violated constraints and deriving new structural events for repairing them (whenever possible)
**context** MedicalTeam **inv** ManagersAreMembers:

self.member->includesAll(self.manager)
Execution of RGDs example

\[
\begin{align*}
&\text{ins \ CriticalTeam(CTeam)} \\
&\text{ins \ MedicalTeam(CTeam)} \\
&\text{ins \ Manages(Foreman, CTeam)} \\
&\text{ins \ IsExpertIn(Cteam, Neuro)} \\
&\text{HasSpeciality(Foreman, Neuro)} \\
&\text{HasSpeciality(Cameron, Neuro)} \\
&\text{Speciality(Neuro)} \\
&\text{IsExpertIn(Team1, Neuro)} \\
&\text{ins \ IsMemberOf(Foreman, CTeam)} \\
&\text{del \ IsMemberOf(Foreman, Team1)} \\
&\text{IsMemberOf(Cameron, Team1)} \\
&\text{Physician(Foreman)} \\
&\text{Physician(Cameron)} \\
&\text{Manages(Cameron, team1)} \\
&\text{MedTeam(Team1)} \\
&\text{ins \ Manages(p, t) \land \neg \ MemberOf(p, t) \rightarrow \text{ins \ Member(p, t)}} \\
&\text{ins \ Member(p, t) \land \text{ins \ CriticalTeam(t)} \land \ Member(p, t2) \rightarrow \text{del \ Member(p,t2)}}
\end{align*}
\]
We obtain the missing effects of some Operation required to satisfy the constraints in the form of structural events.
Roadmap

Operation
Precondition

Operation
Postcondition

UML diagram
OCL Constraints

Information
Base state

Structural
Events

Repair
Generating
Dependencies

How can we obtain such RGDs?
From UML/OCL to RGDs

context MedicalTeam inv ManagersAreMembers:
self.member->includesAll(self.manager)

1. Translate to Logics

∀x,y. manager(x, y) ∧ ¬member(x, y) -> ⊥

2. Augment the literals with Struct. Events

ins manager(x, y) ∧ ¬ member(x,y) ∧ ¬ ins member(x, y) -> ⊥
mananger(x,y) ∧ ¬ del manager(x, y) ∧ del member(x, y) -> ⊥
ins manager(x, y) ∧ del member(x, y) -> ⊥

manager'(x,y) <- ins manager(x,y)
manager'(x,y) <- manager(x,y) ∧ ¬del manager(x,y)

¬member'(x,y) <- del member(x,y)
¬member'(x,y) <- ¬member(x,y) ∧ ¬ins member(x,y)
From UML/OCL to RGDs

3. Move negated Struct. Events to Right Hand Side

\[
\text{ins } \text{manager}(x, y) \land \lnot \text{member}(x,y) \land \lnot \text{ins } \text{member}(x, y) \rightarrow \bot
\]

\[
\text{manager}(x,y) \land \lnot \text{del } \text{manager}(x, y) \land \text{del } \text{member}(x, y) \rightarrow \bot
\]

\[
\text{ins } \text{manager}(x, y) \land \text{del } \text{member}(x, y) \rightarrow \bot
\]
Automatically Translatable OCL

Syntax of OCL fragment we automatically translate:

```
ExpBool  ::= ExpBool \& ExpBool
          | ExpOp
ExpOp    ::= Path->excludesAll(Path)
          | Path->excludes(Path)
          | Path->isEmpty()
          | Path OpComp Constant
          | Path OpComp Path
Path     ::= Var.Navigation
Navigation ::= Member.Navigation
            | Member
            | oclAsType(Class)
| Var.Member->includesAll(Path)
| Var.Member->includes(Var)
| Path->forall(Var| ExpBool)
| not Path.oclIsKindOf(Class)
| Path.oclIsKindOf(Class)
| Class.allInstances().Navigation
| oclAsType(Class).Navigation
| Attribute
```

Ongoing Work: Dealing with OCL\textsubscript{FO}

OCL limited to First-Order constructs

(No Bags, No Tuples, No Aggregates)

Expressively equivalent to Relational Algebra
Experiments Specification

Class Diagram: DBLP Case Study
- 17 classes
- 9 hierarchy specifications
- 18 associations
- 25 OCL constraints

128 Constraints (implicit + graphical + textual)
Experiments

Specification

9 operations for adding/deleting instances of Classes/Associations

For each operation,
We took out 0, 1 and 2 structural events and measured the time needed to recompute the missing structural event.
## Experiments Results

<table>
<thead>
<tr>
<th>Operation</th>
<th>Initial IB</th>
<th>Struct. Events</th>
<th>Time</th>
<th>1 Miss</th>
<th>2 Miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>newAuthoredBook</td>
<td>6</td>
<td>5</td>
<td>50.72s</td>
<td>50.86s</td>
<td>52.21s</td>
</tr>
<tr>
<td>newEditedBook</td>
<td>5</td>
<td>6</td>
<td>50.91s</td>
<td>85.21s</td>
<td>52.80s</td>
</tr>
<tr>
<td>newBookSeriesIssue</td>
<td>6</td>
<td>9</td>
<td>51.74s</td>
<td>59.87s</td>
<td>52.64s</td>
</tr>
<tr>
<td>newJournalPaper</td>
<td>14</td>
<td>5</td>
<td>51.23s</td>
<td>51.68s</td>
<td>52.61s</td>
</tr>
<tr>
<td>delAuthoredBook</td>
<td>11</td>
<td>5</td>
<td>52.31s</td>
<td>50.64s</td>
<td>51.39s</td>
</tr>
<tr>
<td>delBookChapter</td>
<td>10</td>
<td>4</td>
<td>52.96s</td>
<td>50.58s</td>
<td>50.97s</td>
</tr>
<tr>
<td>delBookSeriesIssue</td>
<td>16</td>
<td>8</td>
<td>52.23s</td>
<td>50.92s</td>
<td>51.11s</td>
</tr>
<tr>
<td>delEditedBook</td>
<td>11</td>
<td>6</td>
<td>50.92s</td>
<td>50.86s</td>
<td>51.11s</td>
</tr>
<tr>
<td>dellJournalPaper</td>
<td>17</td>
<td>5</td>
<td>51.23s</td>
<td>51.47s</td>
<td>51.34s</td>
</tr>
<tr>
<td><strong>AVG</strong></td>
<td></td>
<td></td>
<td><strong>51.58s</strong></td>
<td><strong>55.79s</strong></td>
<td><strong>51.80s</strong></td>
</tr>
</tbody>
</table>
Summary

✓ A method for fixing-up operation definitions.

✓ Based on finding the missing effects of some operation to ensure that it can be executed in some concrete IB state without violating any constraint.

✓ The method uses some logic rules called Repair-Generating Dependencies for finding them.
Your ideas required (Open issues)

- Translating logics to RGDs leads to an exponential number of rules depending on OCL length.
- Disjunctions and free variables in RGDs leads to an exponential number of possible solutions.
- Dealing with aggregations.
- From structural events to OCL postcondition.
Thanks for your attention

Questions?
Fixing-Up Non-executable Operations in UML/OCL schemas

Xavier Oriol, Ernest Teniente & Albert Tort
UNIVERSITAT POLITÈCNICA DE CATALUNYA
{xoriol,teniente}@essi.upc.edu
albert.tort@sogeti.com