Mapping OCL as a Query and Constraint Language

Carolina Inés Dania Flores

PhD defense

Supervisors:
Manuel García Clavel - Marina Egea González

Universidad Complutense de Madrid, Madrid, Spain
30th of June, 2017.
Outline

• Motivation
• Background
• Mapping OCL to SQL-PL
• Mapping OCL to MS-FOL
• Application domains:
  • checking model unsatisfiability
  • analysing security and privacy models
  • checking data invariants preservation across states
This research focused on providing methods and tool support which help building complex systems within the Model Driven Architecture framework.
MDA (Model Driven Architecture)

- It supports the development of complex systems by generating software from models.

PIM (Platform Independent Model), PSM (Platform Specific Model)
MDA (Model Driven Architecture)

- It supports the development of complex systems by generating software from models.

PIM (Platform Independent Model), PSM (Platform Specific Model)
Motivation

Why models?

We always create models
Motivation
Why models?

A model can be used in a different ways during the development process:

• for communication purposes to discuss design decisions.
• to provide a detailed specification of the system.
• to develop the system.
Motivation

Why UML?

• UML is the de-facto language for Object-Oriented analysis and design of information systems.
• UML is a standard of the Object Management Group (OMG) (1997), and it is also an ISO standard (2005).
• UML sustains many aspects of software engineering, but it does not provide enough level of precision.
Motivation
Why OCL?

OCL was born as a constraint language to add precision to UML like models and evolved as a query language. It is a declarative language, and OMG and ISO standard.
Motivation

A variety of applications arises for OCL as a query language.

OCL as a constraint language helps to add precision to UML like models with detailed formal semantics.
OCL as a query language

The limitations of OCL as a query language can be solved by mapping it to the most commonly used query systems, i.e. databases.
OCL as a constraint language

Our goal is provide a formal semantics that support automatic reasoning to a great extent so it can be used by software engineers.
Motivation

The quality of the generated code depends on the quality of the source models.

- About 90% of security software incidents are caused by known software defects.
- A study of 45 e-business applications showed that 70% of software failures are related to design.
- One million lines of code can have approximately between 1000 and 5000 software defects in production.

We want to prevent, detect, and correct errors as early as possible.

Motivation

USS Yorktown, smartship

- Crew member entered 0 in a data field and cost a “divide by 0” error
- it down the propulsion
- ship was dead in the water for 2:45mins
Motivation

Mars Climate Orbiter (MCO)

- NASA lost a $125 million
- Metric System Mixup (metrix vs imperial)
Motivation
Motivation
Motivation
Motivation
Motivation

This doctoral dissertation aims to help the current status of methodology and tools for building complex software systems
Background
UML (Unified Modeling Language)  
Ex. Social Network

Class diagram

- classes
- attributes
- associations (association-ends)
- inheritance

Object diagram

- objects
- values
- links
OCL (Object Constraint Language)

- It is a general-purpose (textual) formal language that allows:
  - retrieve objects and their values
  - navigate through related objects
- It supports a set of types with a set of operations over them, and
  - primitive types (Integer, String, Boolean), and
  - collection types (Set, Bag, OrderedSet, and Sequence), and
  - operators like: +, -, >, <, size, isEmpty, notEmpty, characters, and
  - iterators like: forAll, exists, collect
OCL (Object Constraint Language)

- All instances of Timeline
  
  $\text{Timeline.allInstances()}$

- Number of instances
  
  $\text{Timeline.allInstances()} \rightarrow \text{size()}$

- Every profile is older than 18 years old
  
  $\text{Profile.allInstances()} \rightarrow \text{forAll}(p \mid p.\text{age} > 18)$

- There isn’t any profile older than 18
  
  $\text{Profile.allInstances()} \rightarrow \text{select}(p \mid p.\text{age} > 18) \rightarrow \text{isEmpty()}$

- Convert the string ‘hi’ in a sequence of characters
  
  ‘hi’.\text{characters}()
Mapping OCL to SQL-PL
Mapping OCL to SQL-PL

From OCL to SQL-PL
Mapping data/object models.

Data model
• a table with a column for each class
• a column for each attribute
• a table with two columns for each association

Object model
• a row for each object in the table associated with the class
• a row for each link in the corresponding table
From OCL to SQL-PL
Mapping OCL expressions

Every expression is mapped into a stored procedure

```
create procedure name
begin

  OCL to SQL-PL expression

end; //

call name(); //
```

Depending on the complexity of the OCL expressions, they are mapped:
- into a SQL query
- into a SQL query and need an auxiliary block definition
From OCL to SQL-PL
Mapping OCL expressions (cont.)

- Expressions that are mapping into a SQL query

```sql
create procedure name
begin
    select Timeline.pk as val
    from Timeline

    call name();

end;//
```

```sql
create procedure name
begin
    select Timeline.pk as val
    from Timeline

    call name();

end;//
```
From OCL to SQL-PL
Mapping OCL expressions (cont.)

• Expressions that are mapping into a SQL query

Timeline.allInstances()

create procedure name
begin

select Timeline.pk as val
from Timeline ;
end;

call name(); //
From OCL to SQL-PL
Mapping OCL expressions (cont.)

• Expressions that are mapping into a SQL query

```java
Timeline.allInstances()
select Timeline.pk as val
from Timeline

Timeline.allInstances() -> size()
select count(t1.val) as val
from
( ( ) as t1

create procedure name
begin
end;

select count(t1.val) as val
from
( ( ) as t1

call name();
```

```sql
Timeline.allInstances()
select Timeline.pk as val
from Timeline

Timeline.allInstances() -> size()
select count(t1.val) as val
from
( ( ) as t1

create procedure name
begin
end;

select count(t1.val) as val
from
( ( ) as t1

call name();
```
From OCL to SQL-PL
Mapping OCL expressions (cont.)

• Expressions that are mapping into a SQL query

Timeline.allInstances()

create procedure name
begin

Timeline.allInstances() -> size()

select count(t1.val) as val
from
( select Timeline.pk as val
from Timeline ) as t1

end;//
call name(); // /
From OCL to SQL-PL
Mapping OCL expressions (cont.)

• Expressions that are mapping into a SQL query

```
Timeline.allInstances()
```

```
create procedure name
begin
  select count(t1.val) as val
  from
    (select Timeline.pk as val
     from Timeline)
    as t1;
end;//
call name(); //
```
From OCL to SQL-PL
Mapping OCL expressions (cont.)

• Expressions that are mapped into a SQL query and need an auxiliary block definition

```sql
'hi'.characters()
create procedure name
begin
begin
begin
    drop table if exists wchars;
    create temporary table wchars (pos int not null auto increment,
        val varchar(250), primary key(pos));
    insert into wchars(val) (select 'h' as val);
    insert into wchars(val) (select 'i' as val);
end;
select val from wchars order by pos;
end; /
```

<table>
<thead>
<tr>
<th>pos</th>
<th>val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>h</td>
</tr>
<tr>
<td>2</td>
<td>i</td>
</tr>
</tbody>
</table>
begin
  declare done int default 0;
  declare var;
  declare crs cursor for (cursor-specific type - src);
  declare continue handler for sqlstate '02000' set done = 1;
  drop table if exists blq_name;
  create temporary table blq_name (value-specif type )
  open crs;
repeat
  fetch crs into var;
  [Iterator-specific body query]
  if not done then
    [Iterator-specific processing code]
  end if;
until done end repeat;
  close crs;
end;
SQL-PL4OCL
tool component architecture
SQL-PL4OCL
Benchmark

- Vendor specific supported: MySQL/MariaDB, PostgreSQL, SQL Server DBMS
- MariaBD works faster in most of the cases

<table>
<thead>
<tr>
<th></th>
<th>MySQL</th>
<th>MariaDB</th>
<th>PostgreSQL</th>
<th>MSSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.19s</td>
<td>0.13s</td>
<td><strong>0.10s</strong></td>
<td>0.12s</td>
</tr>
<tr>
<td>Q2</td>
<td>0.25s</td>
<td><strong>0.20s</strong></td>
<td>0.33s</td>
<td>0.28s</td>
</tr>
<tr>
<td>Q3</td>
<td>0.36s</td>
<td>0.35s</td>
<td>0.27s</td>
<td><strong>0.26s</strong></td>
</tr>
<tr>
<td>Q4</td>
<td>0.04s</td>
<td><strong>0.04s</strong></td>
<td><strong>0.04s</strong></td>
<td>0.05s</td>
</tr>
<tr>
<td>Q5</td>
<td>0.55s</td>
<td><strong>0.40s</strong></td>
<td>0.40s</td>
<td>0.42s</td>
</tr>
<tr>
<td>Q6</td>
<td>1.05s</td>
<td><strong>0.55s</strong></td>
<td>1.06s</td>
<td>1.03s</td>
</tr>
<tr>
<td>Q7</td>
<td>2.07s</td>
<td><strong>1.56s</strong></td>
<td>1.99s</td>
<td>2.08s</td>
</tr>
<tr>
<td>Q8</td>
<td>50.02s</td>
<td><strong>43.08s</strong></td>
<td>57.04s</td>
<td>53.47s</td>
</tr>
<tr>
<td>Q9</td>
<td>9.14s</td>
<td><strong>8.00s</strong></td>
<td>8.18s</td>
<td>8.89s</td>
</tr>
<tr>
<td>Q10</td>
<td>0.05s</td>
<td><strong>0.04s</strong></td>
<td>0.07s</td>
<td>0.05s</td>
</tr>
<tr>
<td>Q11</td>
<td>49.56s</td>
<td><strong>40.02s</strong></td>
<td>40.10s</td>
<td>43.46s</td>
</tr>
<tr>
<td>Q12</td>
<td>59.58s</td>
<td><strong>51.23s</strong></td>
<td>51.25s</td>
<td>54.82s</td>
</tr>
<tr>
<td>Q13</td>
<td><strong>1.67s</strong></td>
<td>1.98s</td>
<td>2.35s</td>
<td>1.90s</td>
</tr>
<tr>
<td>Q14</td>
<td>59.52s</td>
<td><strong>54.33s</strong></td>
<td>63.35s</td>
<td>58.33s</td>
</tr>
</tbody>
</table>
Related work
(comparison with OCL2SQL-DresdenOCL)

OCL pattern
context: Class
inv: OCL boolean expression

MySQL pattern
select *
from Class
where not OCL2SQL(OCL boolean expression)

OCL2SQL mapping is based on patterns and it does not support iterators.
Mapping OCL to MSFOL
Mapping OCL to MSFOL

C. Dania, M. Clavel: OCL2FOL+: Coping with Undefinedness. OCL@MoDELS 2013: 53-62
C. Dania, M. Clavel. OCL2MSFOL: a mapping to many-sorted first-order logic for efficiently checking the satisfiability of OCL constraints. MoDELS 2016: 65-75
From OCL to MSFOL
Mapping data models

• sorts: \textit{Int}, \textit{String} and \textit{Classifier}.
  (null and invalid for each sort)

• a predicate for each class.
  \textit{Timeline} : \textit{Classifier} → \textit{Bool}

• a function for each attribute.
  \textit{age} : \textit{Classifier} → \textit{Int}

• one/two function(s)/predicate(s) for each association.
  \textit{friends} : \textit{Classifier} × \textit{Classifier} → \textit{Bool}

+ Set of axioms:
\[ \forall (x : \textit{Classifier})(\textit{Profile}(x) \Rightarrow \neg (\textit{Timeline}(x) \lor \ldots \lor \textit{Post}(x))) \]
\[ \neg (\textit{Profile}(\text{nullClassifier}) \lor \textit{Profile}(\text{ invalClassifier})) \]
From OCL to MSFOL
Mapping OCL expressions

- (Sub-)expressions of type **Boolean (Integer)** are translated into formulas (terms)
  - not, and, or, implies, =, >, <, forall, exists, one, isEmpty, notEmpty, includes, excludes, +, -, ….

\[
\text{Profile.allInstances()} \rightarrow \text{forall}(p | p.age > 18)
\]

\[
\forall(x : \text{Classifier})(\text{Profile}(x) \land (age(x) > 18 \land \neg (\text{nullInt} = age(x) \lor \text{invalInt} = age(x))))
\]

Axiom: \(\neg (\text{nullInt} = 18 \lor \text{invalInt} = 18)\)
From OCL to MSFOL
Mapping OCL expressions

- (Sub-)expressions of type **Set** (or **Primitive types** that require definition) are translated into predicates formulas (functions), whose (fresh) predicate (function) symbols satisfy the corresponding axioms (also generated by the mapping)
  - select, reject, including, excluding, collect (follow by asSet),
  - any, max, min

[Profile.allInstances() \rightarrow select(p | p.age > 18) \rightarrow isEmpty()]

**Select**

**Select : Classifier \rightarrow Bool**

\(\forall (x : \text{Classifier})(\text{Select}(x) \Leftrightarrow (\text{Profile}(x) \land (\text{age}(x) > 18 \land \neg (\text{nullInt} = \text{age}(x) \lor \text{invalInt} = \text{age}(x))))))\)

\(\forall (x : \text{Classifier})(\neg \text{Select}(x))\)
Checking unsatisfiability

Data model $\mathcal{D}$. Set of $\mathcal{D}$-constraints $\mathcal{I}$. A Boolean OCL expression $\text{expr}$

Then, $\text{expr}$ evaluates to true in every valid instance of $\mathcal{D}$ if and only if:

\[
\text{o2f}_{\text{data}}(\mathcal{D}) \cup \left( \bigcup_{\text{inv} \in \mathcal{I}} \text{o2f}_{\text{def}}(\text{inv}) \right) \cup \left( \bigcup_{\text{inv} \in \mathcal{I}} \{\text{o2f}_{\text{true}}(\text{inv})\} \right) \\
\cup \text{o2f}_{\text{def}}(\text{expr}) \cup \{\text{o2f}_{\text{false}}(\text{expr})\}.
\]

is unsatisfiable.

Satisfiability Module theories (SMT) solvers

We can expect: $\text{sat}$ (there exists at least one valid instance of the model), $\text{unsat}$ (no valid instance of the model exists), $\text{unknown}$ (check is inconclusive).

SMT solvers cannot be complete when dealing with quantifiers (undecidability).
OCL2MSFOL

tool component architecture

Diagram:
- Data model flows to DM validator.
- OCL content flows to DB model generator.
- Invariants, expression, type expression, and vendor id flow to OCL validator.
- OCL validator flows to SMT solver selector and MSFOL generator.
- MSFOL theory flows to MSFOL constraints.
- MSFOL constraints flow to MSFOL theory.
# OCL2MSFOL Benchmark

<table>
<thead>
<tr>
<th>{1,2}</th>
<th>CVC4</th>
<th>Z3</th>
<th>CVC4fm</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsat</td>
<td>161</td>
<td>24</td>
<td>48</td>
</tr>
<tr>
<td>{1,3}</td>
<td>unsat</td>
<td>173</td>
<td>13</td>
</tr>
<tr>
<td>{2,3}</td>
<td>sat</td>
<td>—</td>
<td>16</td>
</tr>
<tr>
<td>{4}</td>
<td>unsat</td>
<td>138</td>
<td>15</td>
</tr>
<tr>
<td>{5}</td>
<td>sat</td>
<td>—</td>
<td>17</td>
</tr>
<tr>
<td>{5,6}</td>
<td>unsat</td>
<td>172</td>
<td>13</td>
</tr>
<tr>
<td>{1,7}</td>
<td>unsat</td>
<td>237</td>
<td>14</td>
</tr>
<tr>
<td>{1,8}</td>
<td>sat</td>
<td>—</td>
<td>18</td>
</tr>
<tr>
<td>{1,6,8}</td>
<td>unsat</td>
<td>198</td>
<td>16</td>
</tr>
<tr>
<td>{1,9}</td>
<td>sat</td>
<td>—</td>
<td>18</td>
</tr>
<tr>
<td>{1,6,9}</td>
<td>unsat</td>
<td>200</td>
<td>19</td>
</tr>
<tr>
<td>{1,10}</td>
<td>unsat</td>
<td>203</td>
<td>18</td>
</tr>
<tr>
<td>{12}</td>
<td>sat</td>
<td>—</td>
<td>169</td>
</tr>
<tr>
<td>{11,12,13}</td>
<td>sat</td>
<td>—</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>{14,20}</th>
<th>CVC4</th>
<th>Z3</th>
<th>CVC4fm</th>
</tr>
</thead>
<tbody>
<tr>
<td>sat</td>
<td>—</td>
<td>105</td>
<td>28</td>
</tr>
<tr>
<td>{16,20}</td>
<td>sat</td>
<td>—</td>
<td>466</td>
</tr>
<tr>
<td>{17,20}</td>
<td>sat</td>
<td>—</td>
<td>14</td>
</tr>
<tr>
<td>{14,17,20}</td>
<td>unsat</td>
<td>239</td>
<td>13</td>
</tr>
<tr>
<td>{16,19}</td>
<td>unsat</td>
<td>168</td>
<td>16</td>
</tr>
<tr>
<td>{21}</td>
<td>sat</td>
<td>—</td>
<td>17</td>
</tr>
<tr>
<td>{22}</td>
<td>sat</td>
<td>—</td>
<td>199</td>
</tr>
<tr>
<td>{16,22}</td>
<td>unsat</td>
<td>149</td>
<td>18</td>
</tr>
<tr>
<td>{16,23}</td>
<td>unsat</td>
<td>148</td>
<td>16</td>
</tr>
<tr>
<td>{15,17,18,24}</td>
<td>unsat</td>
<td>250</td>
<td>15</td>
</tr>
<tr>
<td>{25}</td>
<td>unsat</td>
<td>63</td>
<td>58</td>
</tr>
<tr>
<td>{11,12,13,18}</td>
<td>sat</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>{6,27}</td>
<td>sat</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>{11,12,13,18,26}</td>
<td>unsat</td>
<td>352</td>
<td>13</td>
</tr>
</tbody>
</table>

- Undefinedness-related (times in ms)
- Generalization-related (times in ms)
### Related work

#### Other mappings from UML/OCL to other formalisms

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Target formalism</th>
</tr>
</thead>
<tbody>
<tr>
<td>FiniteSAT</td>
<td>System of Linear Inequalities</td>
</tr>
<tr>
<td>DL</td>
<td>Description Logics, CSP</td>
</tr>
<tr>
<td>MathForm</td>
<td>Mathematical Notation</td>
</tr>
<tr>
<td>UMLtoCSP</td>
<td>CSP</td>
</tr>
<tr>
<td>EMFtoCSP</td>
<td>CSP</td>
</tr>
<tr>
<td>AuRUS</td>
<td>FOL</td>
</tr>
<tr>
<td>OCL2FOL</td>
<td>FOL</td>
</tr>
<tr>
<td>OCL-Lite</td>
<td>Description Logics</td>
</tr>
<tr>
<td>BV-SAT</td>
<td>Relation Logic</td>
</tr>
<tr>
<td>PVS</td>
<td>HOL</td>
</tr>
<tr>
<td>CDOCL-HOL</td>
<td>HOL</td>
</tr>
<tr>
<td>KeY</td>
<td>Dynamic Logic</td>
</tr>
<tr>
<td>Object-Z</td>
<td>Object-Z</td>
</tr>
<tr>
<td>UML-B</td>
<td>B</td>
</tr>
<tr>
<td>UML2Alloy</td>
<td>Relation Logic</td>
</tr>
<tr>
<td>USE</td>
<td>Relation Logic</td>
</tr>
<tr>
<td>HOL-OCL</td>
<td>HOL</td>
</tr>
<tr>
<td>OCL2FOL+</td>
<td>FOL</td>
</tr>
</tbody>
</table>

**G1**
(Do not support OCL constraints)

**G2**
(Support OCL constraints)

**G3**
(Support OCL constraints and OCL null)

**G4**
(Support OCL constraints and OCL null and invalid)
Application domains
Checking model satisfiability
Case study: eHealth Record Management System

Data models
- 9 classes
- 3 generalisations
- 24 attributes
- 10 associations

Checking model satisfiability
Case study: eHealth Record Management System

- 38 invariants

There must be at least one medical center
MedicalCenter.allInstances() \rightarrow notEmpty()

Every medical center should have at least one employee.
MedicalCenter.allInstances() \rightarrow \forall m (m.employees \rightarrow notEmpty())

Each patient is treated by a doctor who works in the department where the patient is treated.
Patient.allInstances() \rightarrow \forall p (\exists d (d=p.department \wedge p.doctor.departments \rightarrow exists(d | d=p.department)))

1. CVC4 Finite Model returns \textbf{sat} in 7 seconds.
2. If we add 1 more constraint.
   CVC4 Finite Model returns \textbf{unsat} in 4 seconds.
Validating and instantiating models
A Security Metamodel

Data models
• 24 classes
• 3 generalisations
• 47 attributes
• 22 associations
33 invariants

M. Arjona, C. Dania, M. Egea, A. Maña, Validation of a Security metamodel for Development of Cloud Applications. OCL@MoDELS 2014: 33-42
Validating and instantiating metamodels

A Security Metamodel

CVC4 Finite Model returns `sat` + one instance.
Analysing security models

- SecureUML is a modeling language for specifying fine-grained access control policies for actions on protected resources.

Auth(Worker, update(salary)) = false

Auth(Supervisor, update(salary)) = self.supervisedBy = caller or false

Auth(Worker, read(salary)) = caller = self

Auth(Supervisor, read(salary)) = caller = self or true

Analysing security models

Auth(Worker, update(salary)) = false
Auth(Supervisor, update(salary) = self.supervisedBy = caller or false
Auth(Worker, read(salary)) = caller = self
Auth(Supervisor, read(salary) = caller = self or true

Can Bob read Alice’s salary?

Data model \( D \). SecureUML model \( S \).
A role \( r \). An action \( act \).

\[
\begin{align*}
\forall x \in \text{data}(D) \cup \{ \exists \text{caller} \exists \text{self} \\
(\forall f_{\text{true}}(\text{caller}.\text{role} = r) \\
\wedge \forall f_{\text{true}}(\text{Auth}(S, r, act)))\}
\end{align*}
\]
Analysing security models

Auth(Worker, update(salary) = false
Auth(Supervisor, update(salary) = self.supervisedBy = caller or false
Auth(Worker, read(salary)) = caller = self
Auth(Supervisor, read(salary) = caller = self or true

Can Bob read Alice’s salary? ✓

Data model $\mathcal{D}$. SecureUML model $\mathcal{S}$.
A role $r$. An action $act$.

$$\operatorname{o2f_{data}}(\mathcal{D}) \cup \{\exists(caller)\exists(self)$$

$$\left(\operatorname{o2f_{true}}(caller.\text{role} = r) \right.$$  

$$\left.\wedge \operatorname{o2f_{true}}(\text{Auth}(\mathcal{S}, r, act)) \right)\}$$
Analysing security models

Auth(Worker, update(salary) = false
Auth(Supervisor, update(salary) =
  self.supervisedBy = caller or false
Auth(Worker, read(salary)) = caller = self
Auth(Supervisor, read(salary) = caller = self or true

Can Bob read Alice’s salary? ✓
Can Alice update Bob’s salary? ×

Data model $\mathcal{D}$. SecureUML model $\mathcal{S}$.
A role $r$. An action $act$.

$$o2f_{data}(\mathcal{D}) \cup \{\exists(caller)\exists(self)
  \quad (o2f_{true}(caller\cdot role = r)
  \quad \land o2f_{true}(Auth(\mathcal{S}, r, act))))\}$$
Related work

Security models

Many proposals exist for reasoning about RBAC policies, each one using a different logic or formalism.

**Lithium**: framework for specifying and reasoning about FGAC policies. It is based on a decidable fragment of (multi-sorted) first-order logic. In contrast to OCL, this logic does not consider undefined values.

**Kuhlmann et al**: Employing UML and OCL for designing and analysing role-based access control models.
Analysing privacy models
Facebook: posting and tagging

- Who owns the timeline where the post is posted?
  - Who are his/her friends?
- Who posted the post?
  - Who is tagged in the post? Who are his/her friends?
  - Who are his/her friends’ friends?
- Audience selected by the timeline's owner for a post that is posted in his/her timeline.

C Dania, M Clavel: Modeling Social Networking Privacy. TASE 2014: 50-57
Analysing privacy models

Facebook: posting and tagging

Alice posts a photo of herself, Bob and Ted in her timeline, and sets its audience to Friends. Then, Alice tags Bob in this photo.

Can Bob see the photo in Alice’s timeline? ✓

Alice has set her default audience to Friends.

\[
\text{post.audience} = \text{Friends}
\]

Bob is a friend of Alice.

\[
\text{self.profile.friends} \rightarrow \text{includes}(\text{caller})
\]

\[
(\text{post.audience} = '\text{Friends'} \text{ and post.creator} = \text{self.profile} \text{ and post.tags.profiling.friends} \rightarrow \text{includes}(\text{caller}) \text{ and self.profile.blocks} \rightarrow \text{excludes}(\text{caller}))
\]

Method:

\[
\text{readPost(post)}
\]

anybody can read any post that has its audience selected to ‘Friends’ and was created by the owner of the timeline, if he or she is a friend of somebody tagged on the post, unless he or she is blocked by the owner of the timeline.
Checking data invariants preservation

Steps

Preservation of the application’s data invariants.

It consists in 3 steps:

Step 1: Modelling sequences of states (Film, Project).
A filmstrip is a way of encoding a sequence of snapshots of a syst
It consists in 3 steps:

Step 1: Modelling sequences of states (Film, Project).
A filmstrip is a way of encoding a sequence of snapshots of a system.

Step 2: Modelling sequences of data actions (Execute)

\[ \text{Update}(\text{doctor, o1, i+1, ‘Bob’}) \]

\[ o1.\text{doctor}(i+1) = ‘Bob’ \]

Step 3: Proving invariants preservation.

C. Dania, M. Clavel: Model-Based Formal Reasoning about Data-Management Applications. FASE 2015: 218-232
Checking data invariants preservation

Data model $\mathcal{D}$ with invariants $\Phi$

A sequence of actions $\mathcal{A} = \langle act_1, act_2, \ldots, act_n \rangle$

We say that $\phi$ preserves an invariant $\gamma$ if and only if:

$$
o2f_{\text{data}}(\text{Film}(\mathcal{D}, n)) \cup \{o2f_{\text{true}}(\gamma) \mid \gamma \in \bigcup_{i=1}^{n-1} \text{Execute}(\mathcal{D}, act_i, i)\}$$

$$\cup \ o2f_{\text{true}}(\text{not(Project}(\mathcal{D}, \bigwedge_{\psi \in \Phi} (\psi), 1) \implies \text{Project}(\mathcal{D}, \phi, n)))) \text{ is unsatisfiable.}$$
## Checking data invariants preservation

### Case study: eHealth Record Management System

The data model contains 18 entities, 40 attributes, and 48 association-ends.

<table>
<thead>
<tr>
<th>Action</th>
<th>Acts.</th>
<th>Conds.</th>
<th>Invariants</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>affected</td>
<td>preserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>min.</td>
<td>max.</td>
</tr>
<tr>
<td>Create an administrative</td>
<td>8</td>
<td>9</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Create a nurse</td>
<td>10</td>
<td>11</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Create a doctor</td>
<td>11</td>
<td>12</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>Reassing a doctor</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Reassing a nurse</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Register patient</td>
<td>30</td>
<td>6</td>
<td>28</td>
<td>26</td>
</tr>
<tr>
<td>Move a patient</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### Related work

Gogolla et al. From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics.
Conclusions

• Code-generator from OCL queries to the procedural language extensions of SQL (SQL-PL)
  • each OCL expression is mapped to a single stored procedure
  • temporary tables are used
  • the three-valued evaluation semantics of OCL is considered

• Mapping from OCL to many-sorted FOL
  • our results depend of our formalization of UML/OCL in MSFOL and the heuristics implemented in the SMT solver (finite model finder),
  • the four-valued evaluation semantics of OCL is considered.

• Application domain:
  • checking consistency, analysing security and privacy properties, and checking data invariants preservation across states

Future work

• Look for the integration of developed tools into CASE tools
• Empirical validation of the usefulness of the approach for a software engineering team.
Questions?

http://software.imdea.org/~dania/
publications + tools + case studies