Efficient OCL Impact Analysis
Motivation

- The Object Constraint Language (OCL) is used for different purposes in modelling:
  - Constraints
  - Queries
  - Model Transformation
- Especially when used in model transformations OCL expressions form a kind of implicit dependency

OCL query: elements→select(e | e.isRelevant)

source scope  \hspace{1cm} \text{result}
Motivation (2)

- What happens if the source model changes?
- Result may be invalidated.
- OCL expressions needs to be re-evaluated.
- But what happens if things grow large?
  - Huge models.
  - Many, complex constraints.
- Re-evaluating all constraints becomes infeasible.
  - Naive approach:
    \[ O(|\text{expressions}| * |\text{modelElements}|) \]
More Generally

- Given …
  - a set of OCL expressions
  - a set of model elements
  - a model change notification

- Which of the OCL expressions may have changed its value on which context elements?
Agenda

- Running example
- The `traceback` function
- Partial evaluation and delta propagation
- Changes in unused subexpressions
- Evaluation
Running Example

```
context Department::sumBudget():Integer
    if self.subDepartments->size() >= 1 then
        self.subDepartments->iterate(department;
            result : Integer = 0 | result + department.sumBudget())
        + self.budget
    else
        self.budget
    endif

context Department
    inv: self.sumBudget() < 10000
```
The *traceback* function

- Goal: Compute all context objects „*self*“, that for which a given expression \( e \) evaluates to a different result than before the change.

- Example:
  - Expression: `self.subDepartments.budget`
  - Change: \( c_1 = (\text{dep2}.\text{budget} \text{ from } 5000 \text{ to } 5001) \)

- *traceback*: Given a model change determine all expressions which navigate to that property.
  - *traceback* `self.subDepartments.budget(dep2) = \{\text{dep1}\}`
The *traceback* function (2)

- Defined for each type of OCL expression.
- For example: PropertyCallExp (simplified):

\[ \text{traceback}_n(s) := \text{let } t' = \]
\[ \text{`n.source.type` . allInstances() ->select(} \]
\[ \text{`n.referredProperty` = s) ->select(x |} \]
\[ x.oclIsKindOf( `n.source.type`) ) in \]
\[ \text{sourceObjects ->collect(so | traceback}_n.soce(so; t'))')) \]

Example: `self.subDepartments.budget`
Delta Propagation

- In some cases changes cannot have caused an expression to change its result.

- **Example:** `self.subDepartments->select(d | d.x)`
  - Where `x` is of type `Boolean`
  - Adding an element to reference `b` which has `x` set to `false` will not change the result of the expression.

- For complex expressions early determination of empty change sets could reduce computation effort.
Delta Propagation (2)

- An expression $e$ is *monotonic* iff:
  - It is a CallExp expression with upper multiplicity $> 1$
  - It's source expression $s$ has an upper multiplicity $> 1$
  - Adding an element to $s$‘s result either
    - Leaves $e$‘s result collection unchanged or
    - Adds one or more elements to $e$‘s result
  - For each *monotonic* expression (such as select, collect, if-then-else, etc.) we define a function $\delta_e$ which indicates the change in the $e$‘s set of elements.
- Recursively applied this allows for early determination of changes that do not affect the result of an expression.
Partial Evaluation

- For some expressions a „look to the right“ during change analysis can also avoid unnecessary computations:
  - **Example:** `self.name = 'abc'`
    - Not affected by a name change from 'x' to 'y'

- **Combination of Delta Propagation and Partial Evaluation:**
  - Use partial evaluation to determine old and new values of call expressions
  - If delta propagates to an empty set using delta propagation the expression isn‘t affected by the change.
Changes in Unused Subexpressions

- If subexpressions are **not used** to determine the result of the overall expressions changes to them do not impact the overall result.

- Example:

```plaintext
context Department inv: if self.name = 'Boss'
then self.budget < 20 000
else self.sumBudget() < 10 000
endif
```

- If a department’s name is „Boss“, a change to the subDepartments property will not affect the result of the constraint
Total re-evaluation time meaned with a 90% CI

- All optimizations activated, TracebackSteps
- Without unused checks, TracebackSteps
- Without OperationCallExp selection, TracebackSteps
- Without unused checks, without OperationCallExp selection, TracebackSteps
- With delta propagation, NavigationSteps

Scaled models from small to large
Evaluation (2)

- Results at a glance:
  - Traceback implementation significantly faster than naive approach.
  - Partial Evaluation and Delta Propagation have a great positive impact on performance.
  - Unused Subexpressions Check requires a lot of additional partial evaluations which do not amortise.
Application Scenario

- FURCAS language workbench ([www.furcas.org](http://www.furcas.org)) [1]
- OCL-based attribute grammar
- OCL used for queries and attribute computation
- Changes in one part of the model need to be propagated to dependent parts using Impact Analysis

- Other features of FURCAS:
  - View-based modelling for textual languages
  - Decorator model for textual views allows for separation of model content and textual representation
Conclusions & Future Work

- OCL Impact Analysis allows to efficiently re-evaluate OCL expressions over larger models.
- Optimisations additionally positively influence IA performance.
- Impact Analysis implementation available at:
  - Git repository: https://github.com/FURCAS-dev/FURCAS

- Future Work
  - Submit to Eclipse MDT project
  - Extract OCL attribute grammar component from FURCAS
References

Power and productivity for a better world™