Querying UML Models using OCL and Prolog: A Performance Study

Joanna Chimiak-Opoka
Michael Felderer, Chris Lenz
Institute of Computer Science
University of Innsbruck, Austria

TU/e

Christian Lange
Dep. of Math. and Computer Science
Eindhoven University of Technology, Netherlands

9^{th} April 2008, MoDeVVa 2008, Lillehammer, Norway

Overview

1. Introduction
2. Experiment
3. Experiment results
4. Conclusion

1. Introduction

Idea of the study

Authors: Joanna Chimiak–Opoka Chris Lenz Michael Felderer Christian Lange

Topic:	Analysis of UML models		
Method:	Exploratory research		Confirmatory research
Language:	Object Constraint Language	Prolog	other

Question:	Which language is better for UML model querying?
Aspect:	Which language is **faster** for UML model querying?
Method:	Laboratory experiment

Paper: Querying UML Models using OCL and Prolog: A Performance Study

Relevance of the study

- UML inherited complexity caused by
  - multi–diagram view
  - cross-diagram relationships

⇒ detection of model defects

- model size outstripping human perception

⇒ information filtering and aggregation

⇒ a tool supported model querying

- instant feedback

⇒ a tool supported model querying

⇒ fast interpretation of the queries

⇒ selection of a proper querying language

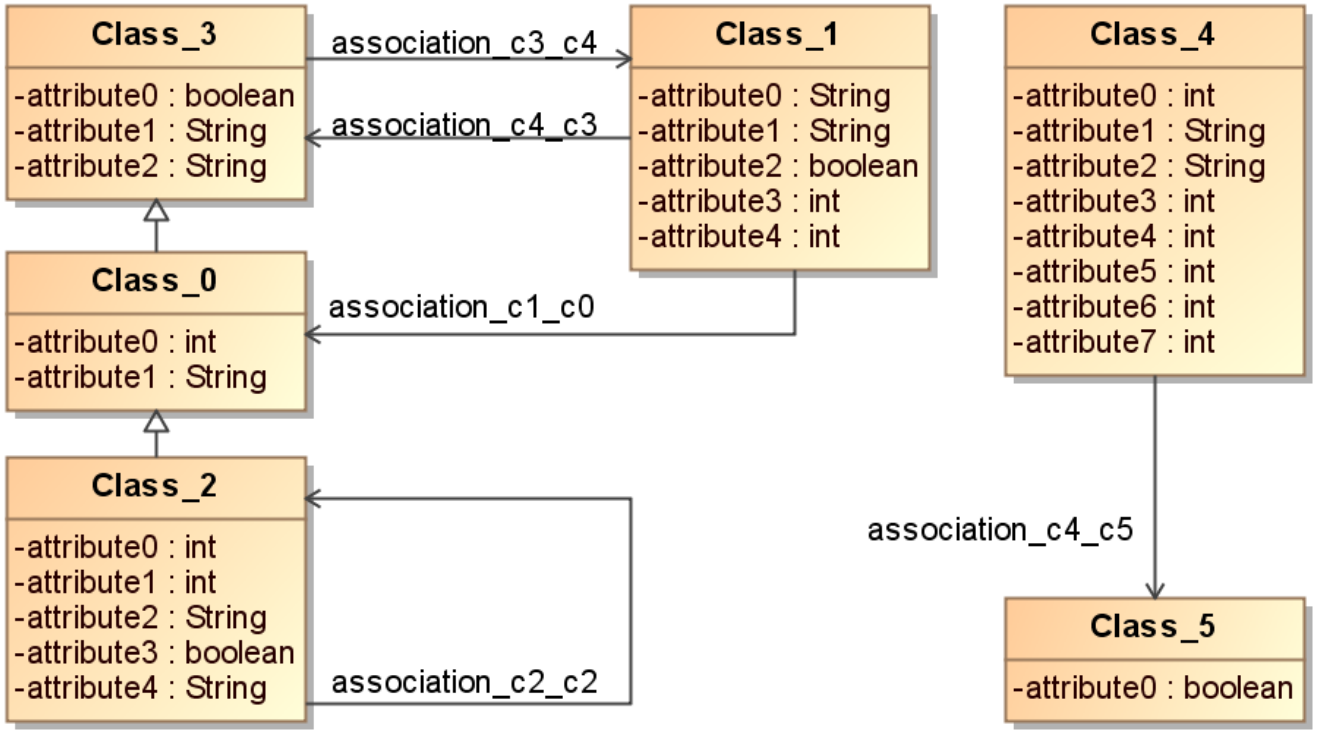
Selection of models

industrial models (i) vs. generated models (g)

- relevancy of models (i)
- control over model size (g)
- control over model structure (g)
- availability of models (g)

For the performance study we use **generated models**.

An exemplary generated model



Model elements:
PrimitiveTypes, Classes, Generalizations, Attributes, Associations
+ Instances, Slots and Links.

Selection of the queries

- used in an industrial case study on quality assessment of models
- general purpose queries (not domain–specific)
- different level of expression complexity

$\Rightarrow$ 8 queries for the experiment

- results with different interpretation

$\Rightarrow$ 3 queries for the paper

Selection of the languages

- querying based on model original representation
- $\Rightarrow$ Object Constraint Language (OCL)
- $\Rightarrow$ EMFT OCL interpreter
- mapping models to another representation
  + querying with an appropriate query language
- $\Rightarrow$ Prolog (list representation)
- $\Rightarrow$ SWI Prolog

2. Experiment

The goal and the method

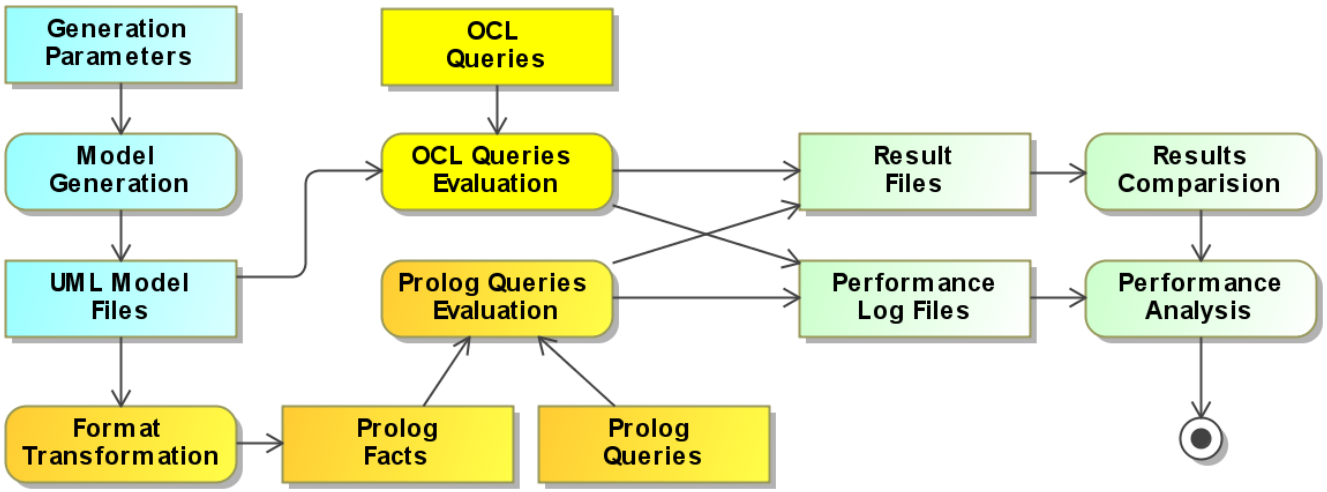
Goal: Performance analysis of model querying with two different query evaluation frameworks.

Additionally: observation of aspects typical for individual query languages.

Method: a laboratory experiment with

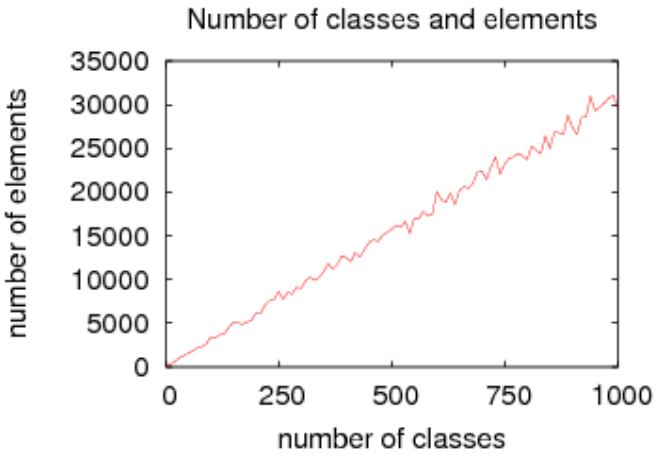
- one dependent variable:
  - the evaluation time
- three independent variables:
  - model (118 models of different size 1...10000)
  - query (8/3 different queries)
  - evaluation frameworks (OCL, Prolog)

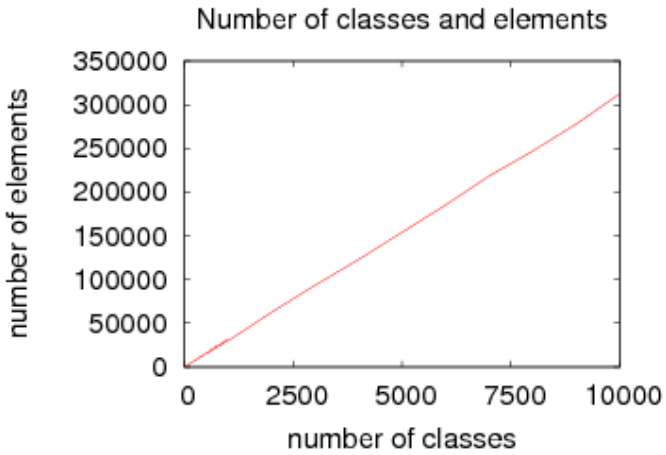
Experiment environment



3. Experiment results

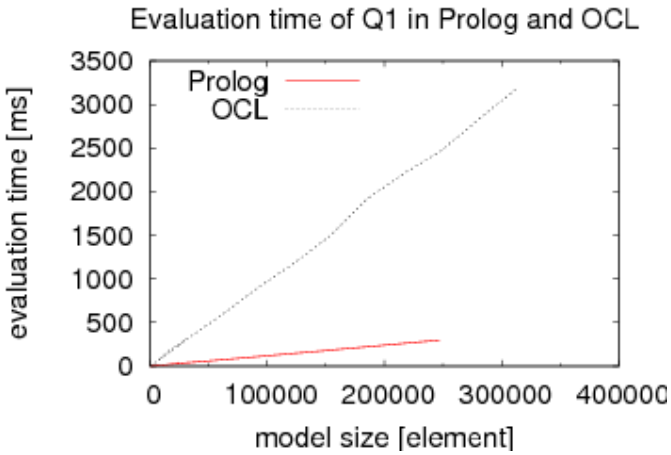
Number of classes and number of elements

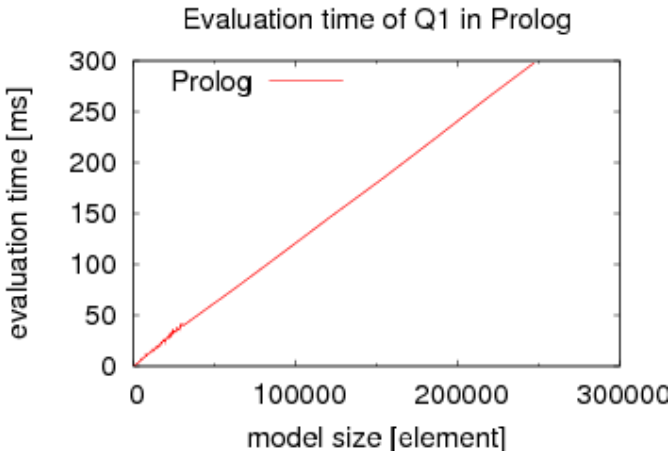




- Generation algorithm $\Rightarrow$ a **linear dependency** up to a randomisation factor
- Generator parameters $\Rightarrow$ a slop of the line
- In the following results **number of elements** is used

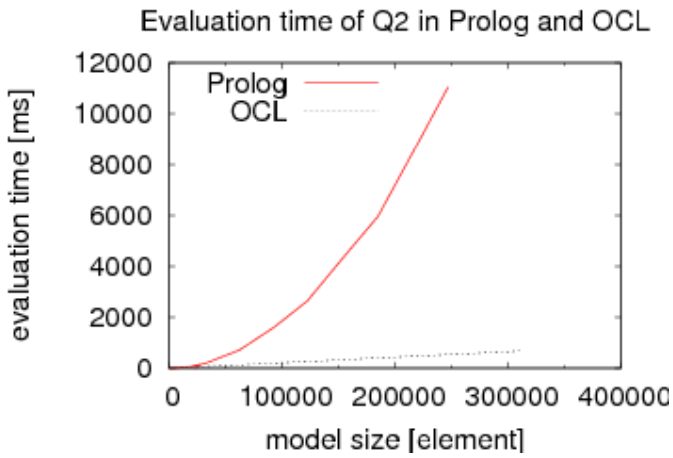
Q1: Overall number of model elements

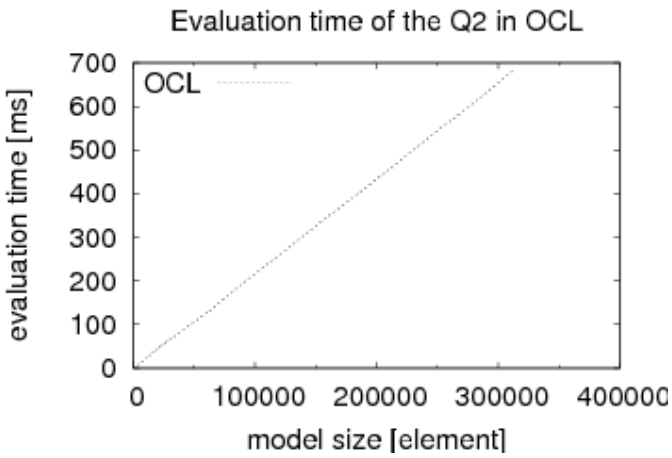




- **linear** execution time for both languages
- **Prolog faster** ⇐ differences in query formulation (much simpler in Prolog)
- In **Prolog model size was limited** ⇐ restrictions on memory size (of the used interpreter)

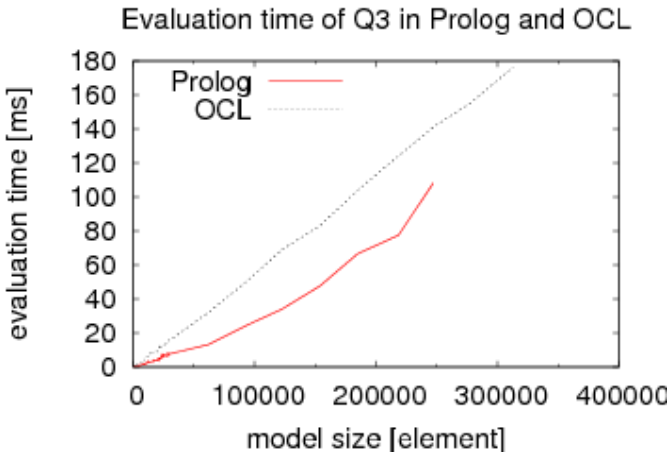
Q2: List of all classes with a given number of associations

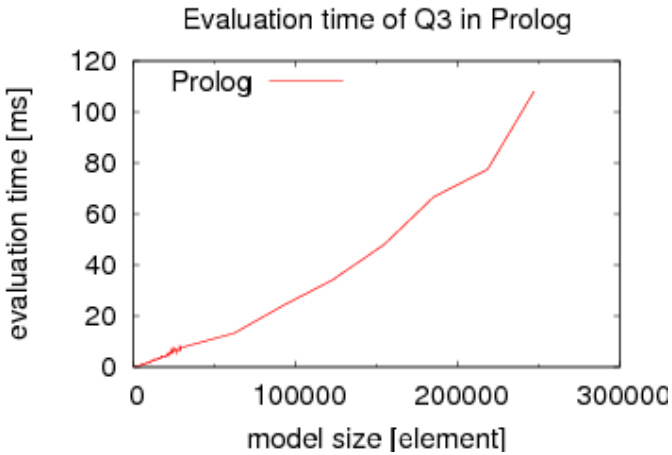




- **OCL faster** with **linear** evaluation time
- **Prolog** with **quadratic** evaluation time ⇐ inter–element dependencies

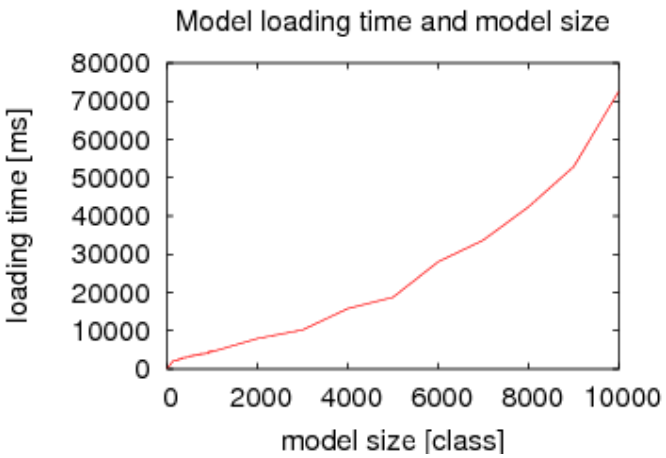
Q3: Maximal depth of the inheritance tree

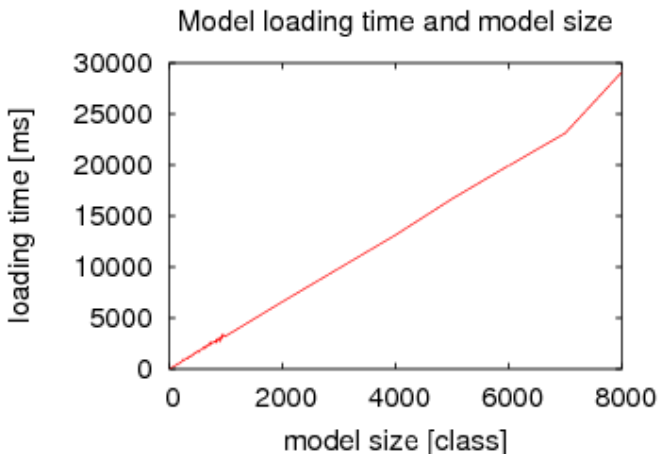




- **Prolog** if **faster** although with **non-linear** execution time
- **OCL** is relatively fast and with **linear** execution time despite the fact of recursive calls of a method

Model loading time





- **OCL non–linear** time (up to 70s) ⇐ a text to object hierarchy transformation
- **Prolog linear** time (up to 30s) ⇐ a text to text transformation
- Influence of model size:
  - OCL — using swap memory ⇒ slower loading
  - Prolog — restrictions on memory size ⇒ limitation on model sizes

4. Conclusion

Summary

- **Fast querying** of models is important for *comprehension, analysis and improvement of models*.
- **Fast feedback** is desired for modeling activities performed by *humans* and within *automated* methods.
- To study the *performance characteristics* of two different querying techniques, OCL and Prolog, we conducted a **laboratory experiment**.
- We have developed an **experiment environment** to generate UML models, evaluate queries, collect, parse and analyse the results.

Discussion

There are two important issues:

- the **choice of queries** was not representative
- the level of **query optimisation** was determined by authors command of the query languages

  The evaluation time of queries can be shortened by improvements in their implementation or by their optimisation through an interpreter. We tried to implement queries to achieve good performance and we assumed that this level of optimisation is realistic.

Results

- A **list representation** of models, as used for Prolog, is more effective for queries collecting or selecting elements based on their **direct properties** (like Q1).
- A **hierarchical representation** (reflecting original structure) of models together with navigation abilities of a query language, as in case of OCL, enables faster evaluation of queries based on properties of **relationships between elements** (like Q2).

Future work

- Studies on *performance and expressiveness* of both languages could provide **application guidelines** for OCL and Prolog queries in model analysis.
- Such guidelines could help tool vendors to decide which representation and evaluation of models and queries is better suited for **types of analysis** to be implemented.

Thank you for your attention

Questions...

Queries in the experiment

1. overall number of model elements (measure for model size)
2. list of all classes that have exactly two associations
3. maximal depth of the inheritance tree
4. ratio of abstract classes (measure for abstractness of the model)
5. number of associations (measure for associativity of the model)
6. class with a specific name
7. list of instances(objects) of a specific class
8. list of all classes that have association to a specific class