Fast, Faster and Super-Fast Queries

István Ráth
@istvanrath
EMF-IncQuery lead
VIATRA2 committer
Budapest University of Technology and Economics

Edward Willink
Eclipse MMT co-lead
Eclipse OCL lead
Eclipse QVTd lead
Eclipse QVTo committer
OMG OCL 2.4 RTF chair
OMG QVT 1.2 RTF rep
OMG UML 2.5 FTF rep

EclipseCon Europe, Ludwigsburg
23rd October 2012
Overview

- Queries and Technologies
- Fast OCL queries
  - OCL to Java code generation
- Faster OCL queries
  - OCL Virtual Machine
- Super-fast queries
  - faster than Java
  - optimized re-evaluation

Summary
**Example Query**

- Ecore metamodel
- Ecore model
- Query

  - result derived by computation on model

```java
class Library {
    List<Pair<Book, Member>> overdueReport(Date today) {
        List<Pair<Book, Member>> results = new ArrayList<Pair<Book, Member>>();
        for (Loan loan : library.getLoans()) {
            if (loan.getDate() < today) {
                results.add(new Pair<Book, Member>(loan.getBook(), loan.getMember()));
            }
        }
        return results;
    }
}
```

**EMF Java to produce overdue Book+Member report**
# Query Technologies

## Increased abstraction/portability

<table>
<thead>
<tr>
<th>Technology</th>
<th>Data model</th>
<th>Query language</th>
<th>Expressive power</th>
<th>Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>RDBMS</td>
<td>SQL</td>
<td>Tuple relational calculus</td>
<td>Compiler</td>
</tr>
<tr>
<td>XML/XPath</td>
<td>XML documents</td>
<td>Xpath expression language</td>
<td>Tree path expressions</td>
<td>Interpreter</td>
</tr>
<tr>
<td>SPARQL</td>
<td>RDF</td>
<td>Graph pattern based logical queries (SPARQL)</td>
<td>Tuple relational calculus</td>
<td>Interpreter, compiler</td>
</tr>
<tr>
<td><strong>OCL (active)</strong></td>
<td>EMF,...</td>
<td>OCL Expression</td>
<td>Powerful functional language</td>
<td>Interpreter, compiler</td>
</tr>
<tr>
<td>EMF Query (mature)</td>
<td>EMF</td>
<td>SQL-like</td>
<td>Simple queries (subset of SQL)</td>
<td>Interpreter</td>
</tr>
<tr>
<td>EMF Query2 (incubation)</td>
<td>EMF</td>
<td>SQL-like</td>
<td>Simple queries (subset of SQL)</td>
<td>Interpreter with index</td>
</tr>
<tr>
<td><strong>EMF-IncQuery (incubation)</strong></td>
<td>EMF,...</td>
<td>Graph pattern based logical queries (IQPL)</td>
<td>Tuple relational calculus + extras (comparable to OCL)</td>
<td>Interpreter, compiler</td>
</tr>
<tr>
<td>Xtend (active)</td>
<td>EMF</td>
<td>Xbase Expression</td>
<td>Extended Java</td>
<td>Compiler</td>
</tr>
</tbody>
</table>
Batch Evaluation Time vs Model Size

Model query performance evaluation by model validation

In order of increasing performance:
- SPARQL tools
- hand-optimized in-memory MySQL
- top Eclipse tools (OCL, EMF-IncQuery)
- hand-optimized EMF/Java

One order of magnitude (10x)
Complete OCL

```
context Library
def: overdueReport(today : ecore::EDate) : Bag(Tuple(book : Book, member : Member)) =
  loans->select(date < today)->collect(loan | Tuple{book = loan.book, member = loan.member})
```

- Complementary document
  - extra operations/properties/invariants
  - existing meta-model unaware of the complements

- Pre-Indigo: no OCL editing support
  - Java API for many years
  - only useable in custom applications

- Indigo: Xtext editor

- Juno: Load Complete OCL Resource...
  - Xtext/Ecore/UML can be complemented
Embedded OCL

```java
class OverdueTuple {
    property book : Book;
    property member : Member;
}

class Library {
    operation overdueReport(today : ecore::EDate) : OverdueTuple[*] { !unique } {
        body: loans->select(date < today)->collect(loan : Loan | OverdueTuple{book = loan.book, member = loan.member});
    }
    attribute name : String;
    property loans : Loan[*] { composes };
}
```

- Pre-Helios: no OCL editing support
  - manual XML editing of CDATA and EAnnotations
- Helios: Xtext OCLinEcore editor
  - automates EMF Delegate EAnnotations entry
  - OCL source embedded in EMF’s Java
    - automatically executed by OCL interpreter
- Juno: OCL to Java code generator
  - automatically executed by EMF genmodel
Fast OCL execution

@Generated NOT manual Java (pre-Helios)

EMF Delegates support EAnnotations (Helios)

Direct OCL 2 Java (Juno)

Ecore Model \rightarrow genmodel \rightarrow Java File/Class

@generated NOT

Ecore Model

EAnnotation

some_ocl

genmodel

Java File/Class

some_ocl

run

parse

interpret

Ecore Model

EAnnotation

some_ocl

genmodel

Java File/Class

@generated NOT

run

Ecore Model

EAnnotation

some_ocl

genmodel

Java File/Class

(Impl)
call

Java File/Class

(OCL bodies)

run
Useable OCL

Step 1, Add OCL to Ecore

Step 2, optionally enable Code Generation

Step 3, Generate Models
Fast OCL Execution Support

- Fast Control
  - Acceleo templates tree-walk at compile-time
- Fast Values with OCL semantics
  - 1 is always equal to 1.0 (Java: only sometimes equal)
  - unlimited integers (Java: unchecked 32/64 bit wraparound)
- Polymorphic Values
- Fast Operations and Properties
  - Polymorphic feature implementation API
- Fast Reflection, Metamodel access
  - auto-generated dispatch table view of metamodel
Is the OCL VM Faster?

- Fast Control, Values, Operations, Reflection
- Juno Code Generator performance
  - no run-time parsing costs - good
  - subsequent speed similar - really disappointing

Investigation

- eGet("XXX") costly
  - code generate as getXXX saves 30%!
- fully polymorphic values are very costly
  - every collection is copied
  - every collection element is wrapped by a 'box' object
Smart Code Generation => Faster

Kepler M3 is 3 to 29 times faster than Juno

- Partial value polymorphism
  - EMF collections used directly
  - only Integer, Real and EClass objects wrapped/boxed
  - static analysis eliminates `instanceof` costs
  - redundant polymorphic dispatch eliminated

Still To Do

- deeper analysis of invalid/null
- operation inlining, CSE, ...
Faster, Extensible OCL Evaluation

- OCL evaluation is just a tree walk over the AST
  - interpreter visits tree nodes directly at run time
  - compiler pre-visits tree nodes at compile time
- Extended languages add extra AST classes
  - extra visitor methods
  - same evaluation process
  - same debugging requirements
- OMG extended languages
  - QVTc, QVTo, QVTr, MOFM2T (Acceleo)
Real systems

Simple systems - full evaluate once
  - just need efficient evaluation

Real systems
  - evaluate [small-change re-evaluate]*

Naive Java full re-evaluation costly
  - e.g. Xtext substantially reconstructs intermediates
  - reliable hand-coded incremental update error-prone

Intelligent re-evaluation is selective
  - "Only recalculate what is necessary, reuse the rest"
Benchmarking

Goal: to assess query performance of EMF-based tools

Methodology
- Use small (<1000 EObjects), medium (10-100k EObjects) and very large instance models (up to 2.8M EObjects)
- Queries of varying complexity

Benchmarking loop (measure memory usage throughout)
- Load model (measure model initialization time)
- Evaluate queries (measure evaluation time)
- Modify model (measure modification time)
- Re-evaluate queries (measure re-evaluation time)

Batch validation
(Incremental) revalidation
Batch Validation Reprise

- OCL performance significantly impacted by query specification (.allInstances)
- Same effect for „naive“ Java code

Top Eclipse tools (OCL, EMF-IncQuery) within close range of hand-optimized EMF/Java
**Surprise (?)**: Re-validation

RouteSensor Edit Time + Check2 Time, UserScenario

Result set change / Number of modifications [#elem]

- OCL+IA and EMF-IncQuery are characteristically faster than hand-optimized EMF/Java
- Both provide nearly constant re-evaluation performance

- IA can improve OCL performance for revalidation
- EMF-IncQuery 10x-100x faster than IA

Model size / Batch result set size [#elem]

Time [ms]

1e+06
100000
10000
1000
100

- Eclipse OCL+IA (Ecore) Refactored
- EMF-IncQuery
- Eclipse OCL (CG M3)
- Eclipse OCL (Ecore) Refactored
- Java
- Eclipse OCL (Ecore)
- Eclipse OCL+IA (Ecore)
The reason: the power of declarative approaches

- Java: imperative expressions with side effects
  - Code difficult/impossible to analyze
- Declarative languages: side-effect free expressions
  - Programs can be analyzed to optimize change processing
  - Supports efficient re-evaluation: only re-evaluate those areas affected by changed input

Approaches
- OCL Impact Analyzer
- EMF-IncQuery
EMF-IncQuery: Pattern queries

OCL expressions: Model-based evaluations
- Constraint - returns true/false
- Query - returns one or more values/objects

IncQuery Patterns: Additional Abstraction
- Query - returns a tuple of matching objects
  - (Constraint – an annotated query)

objects in source model
- constrained by instance type
- constrained by inter-relationships
- constrained by arbitrary subqueries
  - attribute constraints as restricted/side-effect free Xbase expressions
Pattern example

pattern OverdueBooks(P: Member, B:Book, Today: Date) {
  Loan(L); // declaration optional due to type inference
  Loan.book(L,B);
  Loan.member(L,P);
  Loan.date(L,D);
  check(Today.after(D))
}

Incremental example
- a book is renewed
- one due date changes
- one OverdueBooks match changes
- <= full-re-evaluation

Incremental example 2
- today becomes tomorrow
- all comparisons previously returning true re-evaluate
- some OverdueBooks matches change
- < full-revaluation
Comparing rule "difficulty"

EMF-IncQuery provides top performance regardless of query complexity.
OCL(+IA) performance prone to degradation for complex queries and/or use of .allInstances().

![Graph comparing rule "difficulty" showing EMF-IncQuery's superior performance compared to OCL(+IA).]
EMF-IncQuery

Coming very soon to Eclipse Modeling, get it from the Marketplace today

Features available today

- Fully featured Xtext2-based developer tools
- Integration options
  - Generic/generated API: both query results and incremental deltas
  - Query-based derived EReferences/EAttributes with notification support
  - Databinding support
- Interactive Query Explorer to provide live views of models, works with any EMF-based tool

Near future

- Zest visualization support
- Full UML support (Dynamic EMF)
Super-fast queries

Today in Juno

- OCL+IA can reduce re-evaluation times
- EMF-IncQuery
  - Provides top performance regardless of model size and query complexity (much faster than OCL+IA and handwritten Java)
  - At the cost of somewhat increased memory consumption

Future

- Synergies between EMF-IncQuery and OCL(+IA)
- Common framework for query-based features
Super-fast queries

Today in Juno

- OCL+IA can reduce re-evaluation times
- EMF-IncQuery
  - Provides top performance regardless of model size and query complexity (much faster than OCL+IA and handwritten Java)
  - At the cost of increased memory consumption

Future

- Synergies between EMF-IncQuery and OCL(+IA)

To see EMF-IncQuery in live action doing realtime gesture recognition, check out Jonas Helming's Jnect Session on Thursday:

Jnect – Get Your Eclipse Moving
(Thursday 2.30PM-3PM, Theater)
Summary

- **Fast**
  - direct OCL to Java code generation

- **Faster**
  - efficient OCL Virtual Machine dispatch tables
  - ongoing optimizations

- **Super-fast**
  - declarative (re-)evaluation can be optimized with EMF-IncQuery and OCL+IA
  - much faster than (naive/typical) Java!
Unused slides
Xcore and friends / OCL

- If all you want is Java
  - use Xcore/Xbase not OCL

- OCL is a Specification language -
  - platform neutral

- Sharper syntax
  - much of it adopted by Xbase

- Declarative
  - side effect free, no assignment

- Foundation for MOFM2T(Acclelo), QVT, ...
OCL Code Generator

- Acceleo templates
  - flexible, extensible
  - eliminates run-time parsing costs
- Run-time speed not significantly faster
  - polymorphic, accurate, UML-aligned
- Juno 2012, pivot-based code generator
  - Acceleo OCL to Java templates
  - eliminates run-time compilation
    - not significantly faster
OCL Evaluation Principles

- AST Tree walk
  - root to if to ...
- AST node per usage
  - IfExp
  - VariableExp
  - IntegerLiteralExp
- Extensible
  - Extensible AST nodes
  - auto-generated Visitor (faster than Ecore Switch)
'Ecore' Operation Call : \texttt{a.b(c,d)}

- Tree search over type and supertypes \--- a
  - Linear search for operation name \--- b
    - Linear search to match argument types \--- (c,d)
      - Tree search for conformant type/supertype \--- c then d

- Select best unique match
'OCL VM' Operation Call

- Fragment provides derived view of base
  - may have overloaded entries

- Linear search of fragments at required depth
  - Direct index to operation
Example OCL VM dispatch

Problem
- fa1() for a C

Compile-time
- A::fa1
- depth 0, index 1

Run-time
- C
- A is depth 0
  - A for C
- A::fa1 is index 1
  - C::fa1

let c : C in c.fa1()
OCL Virtual Machine => Faster

- Non-polymorphic values bad
  [org.eclipse.ocl.ecore since at least Europa]
  - Boolean, String, Integer ...
  - difficult/impossible to have precise OCL semantics

- Fully polymorphic values bad
  [org.eclipse.ocl.examples.pivot since Indigo]
  - BooleanValue, StringValue, IntegerValue ...
  - too many boxing/wrapping objects

- Partially polymorphic values also bad
  - Boolean, String but IntegerValue ...
  - too many instanceof tests
Simple Interpreted OCL VM

Program is an Abstract Syntax Graph (AST)

- VariableExp
  - references variable to read as a value
- PropertyCallExp
  - references object and property to read as a value
- OperationCallExp
  - references operation to apply to some values

Run-time Interpretation

- tree-walking evaluation visitor

Extensible with new AST node classes
Code Generated OCL VM

- Program is an Abstract Syntax Graph (AST)
- Compile-Time Code Generation
  - tree-walking code generating visitor
- Run-time Execution
  - direct Java, direct model accesses
- Extensible with new AST node classes
- Optimisable
  - direct model access getXX() rather than eGet('XX')
  - inlining of non-polymorphic (final) operations
(Imperative) OCL VM for QVT

- Simple OCL VM
  - AST walker
  - QVT richer AST

- Code generated VM
  - dispatch tables
  - flattened code
  - inlined operations

- Debugging tools