Efficient Persistence, Query, and Transformation of Large Models

Gwendal DANIEL
Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya (UOC)
gdaniel@uoc.edu

Supervisor: Jordi Cabot
Co-Supervisor: Gerson Sunyé
Co-Supervisor: Massimo Tisi
Context

“A model of a system or process is a theoretical description that can help you understand how the system or process works, or how it might work.” (Collins 2017)

Credit: https://www.autodesk.com/solutions/bim
Context

• Software Modeling

Forward Engineering

Reverse Engineering
Limited adoption in the industry
• Scalability of existing solutions
• Large generated models

[J. Whittle et. al., 2014]
Limited adoption in the industry
• Scalability of existing solutions
• Large generated models

[J. Whittle et. al., 2014]
Research Challenges

1. Storing large models
   - Efficient data structures
   - Efficient persistence solutions
   - Efficient memory management

2. Accessing stored models efficiently
   - Data representation
   - Caching and prefetching

3. Querying large models
   - Complex model navigations
   - Efficient constraint checking

4. Transforming large models
   - Refactoring operations
   - Code generation
Contributions

• A novel scalable modeling ecosystem

1. A multi-database model persistence solution

2. A model prefetching and caching component

3. An OCL to database query language approach

4. A scalable model transformation engine
Efficient Model Persistence

Efficient Transformations
- Gremlin-ATL

Efficient Queries
- Mogwaï

NeoEMF

PrefetchML

Scalable Model Persistence
Model Persistence

• Default serialization mechanism: XMI

• Scalable model persistence frameworks: CDO, Morsa
  • Use databases to store models
  • Lazy loading
  • Low memory footprint
Problematic

• Performance & memory issues
  • [Pagan et.al., 2011; Gomez et. al., 2015]

• Mostly relational solutions

• Generic scalability improvements
  • Single storage solutions
  • Not aware of the modeling scenario

• External (non-standard) APIs
NeoEMF

• A rich infrastructure that allows to choose the database and data representation to use.
• Compliant with existing modeling solutions
• Extensible architecture
• Memory efficiency
NeoEMF

• NoSQL persistence
  • Specific purpose
  • Efficient to process large amount of data
  • Built-in scaling capabilities
  • Advanced query language
NeoEMF

Model-Based Tools

Modeling Framework

NeoEMF Advanced API

Persistence Framework

NeoEMF Core

Persistence API

Backend Connector

Data-store Connector

Caching

Importers

Backend API

XML

Blueprints

MapDB

BerkeleyDB

HBase + Zookeeper

Model Access API
Evaluation

Invisible Methods Results (ms)

EMF-XMI  CDO  NeoEMF/Graph  NeoEMF/Map

3 models (6000 to 1.5M elements)
Constrained memory environment
Model Caching and Prefetching

Efficient Transformations
- Gremlin-ATL

Efficient Queries
- OCL
- Mogwaï

NeoEMF

PrefetchML

Scalable Model Persistence
Prefetching & Caching

• Prefetching
  • Bring objects in memory before they are requested

• Caching
  • Retain objects in memory to speed-up their access

• Integrated in databases and file systems
  • Speeds-up I/O intensive applications
Problematic

• Database prefetchers & caches
  • Lack of fine-grained configuration
  • Tightly coupled to data representation
  • Not common in NoSQL stores

• Prefetching components in model persistence frameworks
  • Not expressive enough
  • Lack of fine-grained strategies
Hypothesis

- We need to prefetch and cache models
  - Scalable persistence frameworks
    - Databases to store large models (relational, NoSQL)
    - Latencies to bring elements from the database (lazy-loading)
  - Metamodel information
    - High-level prefetching and caching rules
    - Decoupled from the data representation
PrefetchML

• A prefetching and caching framework at the model level

• PrefetchML DSL (rule definition)
  • Metamodel and model level
  • Use case dependent
  • Readable

• PrefetchML Engine (rule execution)
  • Datastore independent
  • Transparent to the persistence solution
PrefetchML

Import ‘http://www.example.com/Java’

plan samplePlan {
  use cache LRU[size=100, chunk=10]

  rule r1: on starting
    fetch Package.allInstances()

  rule r2: on access type Class (self.name = ‘c2’)
    fetch self.methods.modifier

  remove type Package
}
Evaluation

- Up to 18% improvement on NeoEMF/Graph
- Up to 95% improvement on NeoEMF/Map
- Invalid plans detected at runtime
- New rule suggestions
Efficient Model Queries
State of the Art

Under the Hood

- **Low-level modeling API** → Not aligned with the database capabilities
- **Fragmented queries** → Not efficient → Remote database
- **Intermediate objects** → Memory consumption → Execution time overhead

Modeler Point of View

- **OCL Query**
- **OCL Interpreter**
- **EMF API**
- **Database**

- get(p1)
- get(p1,name)
- ...
- get(pn)
- get(pn,name)

---

Mogwaï
Problematic

• Not efficient to compute model queries

• Database queries are more efficient but
  • Modern persistence frameworks typically rely on NoSQL databases
    • Multiple query languages
    • Multiple data representations
    • Low-level queries are hard to understand and maintain
    • Modeling expertise vs. Database expertise

• Solution: generate them!
Mogwaï

Modeler

OCL Query

Model Transformation

Gremlin Traversal

Blueprints Database

- Generate graph database queries from OCL expressions
- Bypass modeling framework API
- Gremlin Query Language
- Single execution of the queries
- Compatible with existing modeling frameworks
Mogwai

- Gremlin metamodel
  - ~ 100 classes
- ATL Transformation
  - OCL to Gremlin mapping
  - Query composition
  - 70 rules and helpers

<table>
<thead>
<tr>
<th>OCL Expression</th>
<th>Gremlin Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>g.idx(&quot;&quot;metaclasses&quot;)[[name:'Type'']]</td>
</tr>
<tr>
<td>AllInstances()</td>
<td>inE(‘instanceof’).outV</td>
</tr>
<tr>
<td>Collect(att)</td>
<td>Att</td>
</tr>
<tr>
<td>Collect(ref)</td>
<td>outE(‘ref’).inV</td>
</tr>
<tr>
<td>Select(cond)</td>
<td>Filter{cond}</td>
</tr>
<tr>
<td>oclIsTypeOf(Type)</td>
<td>outE(‘instanceof’).inV.transform{it.next() == Type}</td>
</tr>
</tbody>
</table>

Package.allInstances().classes
- select(c | c.name = ‘class1’)

g.idx(""metaclasses")[[name:""Package'""]]
inE(""instanceof")\.outV.
outE(""classes")\.inV.
filter{it.name='class1'}
Evaluation

Invisible Methods Results (ms)

2 models (80000 to 1.5M elements)
Constrained memory environment
NeoEMF/Graph
Scalable Model Transformations

Efficient Transformations

Efficient Queries

Scalable Model Persistence
Gremlin-ATL

- Extension of Mogwaï for model transformations
  - ATL to Gremlin transformation
  - Multiple database support
  - Advanced configuration

- Evaluation: reference transformation
  - Comparison with the standard ATL engine
  - Reduces the execution time by a factor of 2 to 134
  - Reduces the memory consumption by a factor of 10
Conclusion
Conclusion

• Novel modeling ecosystem
  • Scalable persistence approach
    • Multi-databases: adapted to the expected modeling scenario
    • Compliant with existing modeling solutions
    • On-demand loading
  • Prefetching and caching component
    • Improve large model access efficiency
    • Define precise plans adapted to a given model access pattern

Scalable Model Persistence
Conclusion

• Novel modeling ecosystem
  • Translation-based approach for query and transformation computation
    • Standard language support
    • Generate database-specific operations
    • Bypasses modeling framework API
    • Positive performances on large models
Now what?

• Apply modeling techniques to improve big data processes
  • Define database structure
  • Understand existing data
  • Handle variety
UmlToGraphDB (ER 2016)

• Conceptual schema to Graph database mapping
  • MDA methodology
  • Input UML/OCL
  • Output Neo4j/Gremlin
  • (Optional) Code generation
Ongoing work

• Apply modeling techniques to improve big data processes
  • UML to NoSQL
    • Multi-database model persistence
    • Cross-datastore constraint checking
  • Model transformations to express data migration operations
    • Promising results on the Neo4j panama paper database
Publications

• Journals
  • G. Daniel, G. Sunyé, A. Benelallam, M. Tisi, Y. Vernageau, A. Gomez & J. Cabot
  NeoEMF, a Multi-Database Model Persistence Framework for Very Large Models. SCP 2017
  • G. Daniel, G. Sunyé, J. Cabot
  Advanced Prefetching and Caching of Models with PrefetchML. SoSym 2018

• International Conferences
  • G. Daniel, G. Sunyé, J. Cabot
  Mogwai, a Framework to Handle Complex Queries on Large Models. RCIS 2016
  • G. Daniel, G. Sunyé, J. Cabot
  • G. Daniel, G. Sunyé, J. Cabot
  UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases. ER 2016
  • G. Daniel, F. Jouault, G. Sunyé, J. Cabot
  Gremlin-ATL: a Scalable Model Transformation Framework. ASE 2017

• 3 Workshops
Websites & Software Repositories

NeoEMF: neoemf.com

PrefetchML: github.com/atlanmod/Prefetching_Caching_DSL

Mogwai/Gremlin-ATL: github.com/atlanmod/mogwai

UmltoGraphDB: github.com/atlanmod/uml2nosql
Question?

Thank you for your attention!