TracQL: A Domain-Specific Language for Traceability Analysis

Norbert Tausch¹, Michael Philippson¹, and Josef Adersberger²

¹University of Erlangen-Nuremberg, Germany, [norbert.tausch|philippson]@cs.fau.de
²QAware GmbH, Munich, Germany, adersberger@qaware.de
Overview

- Introduction
  - Motivation
  - Problem

- The Traceability Query Language
  - Goals
  - Characteristics

- Evaluation
  - Architecture-to-code consistency

- Conclusion
Motivation

- Traceability helps to improve and maintain software quality in the software development process.
- Project: A Software Project Control Center
Problem

- Implementation of traceability analysis is complex.
  - Value chain: Extraction -> Representation -> Analysis

- Current approaches do not provide a suitable framework.
  - They use different languages for analysis:
    - Model-based: ATL, OCL, QVT
    - Graph-based: Gremlin, GreQL
    - Database-based: SQL
    - XML-based: XPath, XPointer, XQuery
    - Traceability-related: TQL, VTML
  - Disadvantages:
    - External DSLs are difficult to extend.
    - Cumbersome to work with multiple data sources and to create (inter-model) links between them.
The Traceability Query Language – Goals

- Idea:
  - Provide a language for the whole traceability value chain.

- Goals/Requirements:
  - Representation-Independence:
    - To work with multiple data sources including inter-model links.
  - Extensibility:
    - To add new analysis and to adjust old ones to the current project.
  - Expressiveness:
    - To provide clear and concise traceability analyses.
  - Performance:
    - No performance penalty that breaks the workflow.
The Traceability Query Language – Characteristics

- **TracQL** is graph-based.
  - Property graph model
  - Adapter concept (e.g., Neo4j Graph-DB, EMF model)

- **TracQL** is **statically typed**.
  - Provides typed graphs.
  - Works with concrete artifact and link types (e.g. EMF classes).

- **TracQL** is an **internal DSL** which is:
  - Based on Scala (object-oriented and functional language).
  - Directly extensible with new functions and operators.
Evaluation

- We focus on anomaly analysis.
  Detection of divergences between architecture and code.

```python
def findDivergences(graph: ArchitectureGraph) =
  for { source <- graph.vertices
       target <- findRelated(source) -- findAdjacent(source)
  } handleDivergence(source, target)
```
Evaluation – Results

- Example: Find related artifacts (details in the paper).

```scala
def findRelated(artifact: QVertex) = artifact.successors(Arch2CodeLink).
  during(_.successors(NestingLink), Every[Qvertex]).
  successors(Link).
  during(_.predecessors(NestingLink), Code.Types).
  toSeq.predecessors(Arch2CodeLink)(!Identity(artifact)).
  foldLeft(HashMap[QVertex, Int]())((map, a) => increaseCount(map, a))
```

- Evaluation: Industrial project (11k vertices, 38k edges)

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler Tokens</th>
<th>Factor</th>
<th>In Memory</th>
<th>Factor</th>
<th>Neo4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>TracQL</td>
<td>85</td>
<td>1.0</td>
<td>58</td>
<td>1.0</td>
<td>136</td>
</tr>
<tr>
<td>Gremlin Groovy</td>
<td>144</td>
<td>1.7</td>
<td>184</td>
<td>3.2</td>
<td>223</td>
</tr>
<tr>
<td>Gremlin Java</td>
<td>232</td>
<td>2.7</td>
<td>126</td>
<td>2.2</td>
<td>347</td>
</tr>
<tr>
<td>Cypher</td>
<td>301</td>
<td>3.5</td>
<td>-</td>
<td>44.0</td>
<td>5,982</td>
</tr>
</tbody>
</table>
Conclusion

- *TracQL* is an internal DSL focused on implementing traceability analysis.
  - It aims at supporting the whole traceability value chain: Extraction -> Representation -> Analysis

- *TracQL* fulfills our main goals/requirements:
  - Representation-Independence
  - Extensibility
  - Expressiveness
  - Performance

- We evaluated *TracQL* on a non-trivial architecture-to-code traceability problem.