TracQL: A Domain-Specific Language for Traceability Analysis

Norbert Tausch[1], Michael Philippsen[1], and Josef Adersberger[2]
[1]University of Erlangen-Nuremberg, Germany, [norbert.tausch|philippsen]@cs.fau.de
[2]QAware GmbH, Munich, Germany, adersberger@qaware.de

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

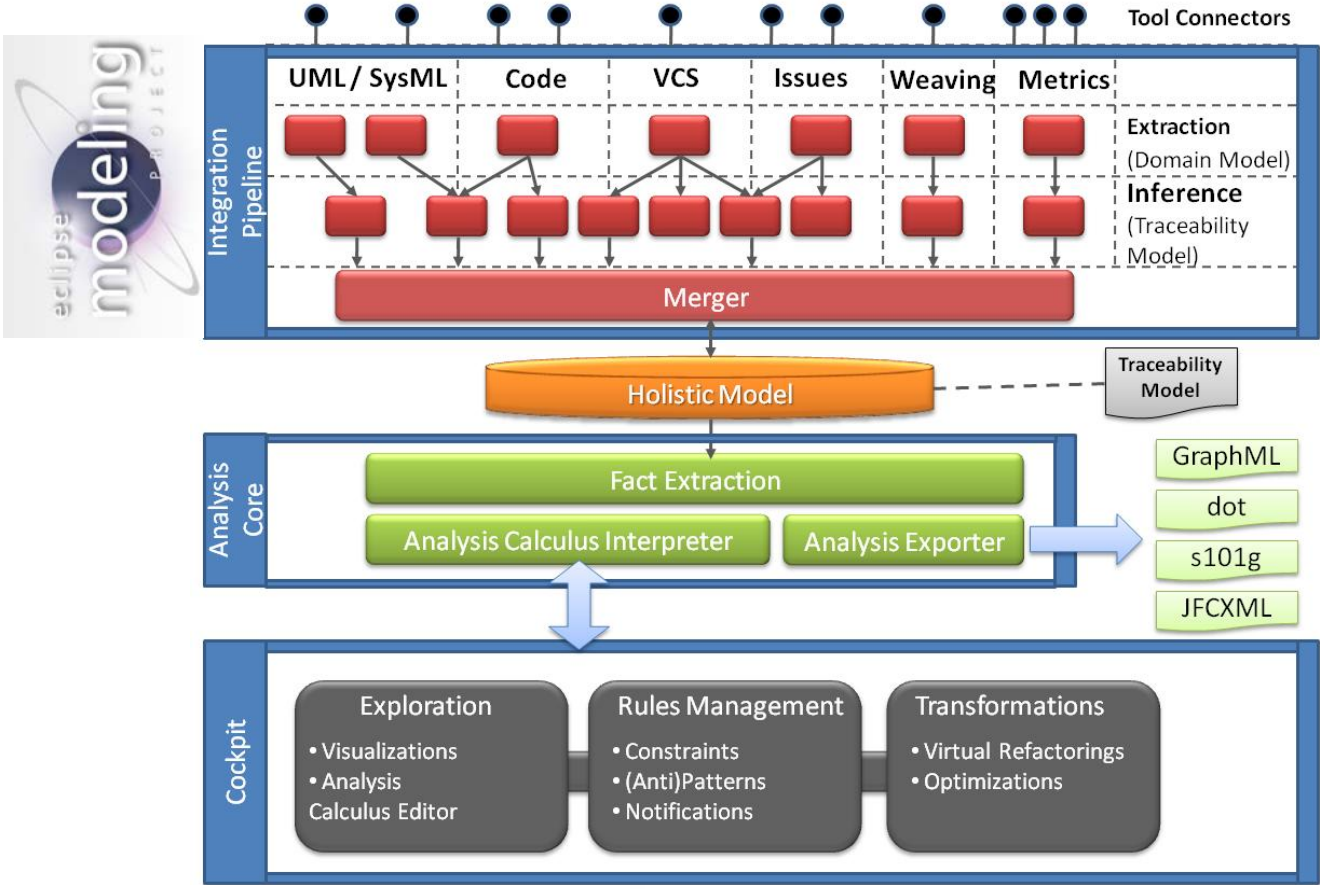
COMPUTER SCIENCE DEPARTMENT
PROGRAMMING SYSTEMS GROUP
ERLANGEN • GERMANY

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.

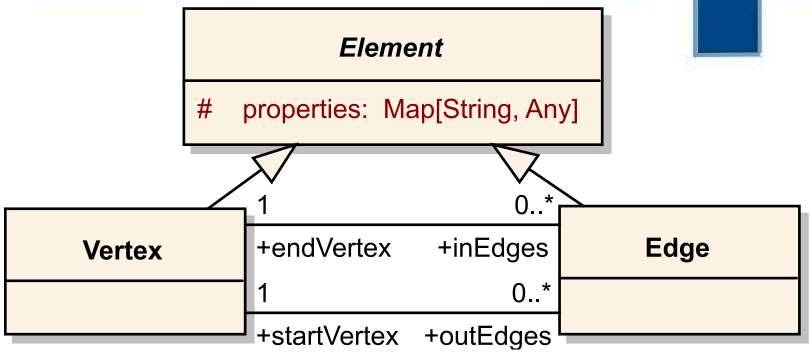
The Traceability Query Language – Goals

- 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)



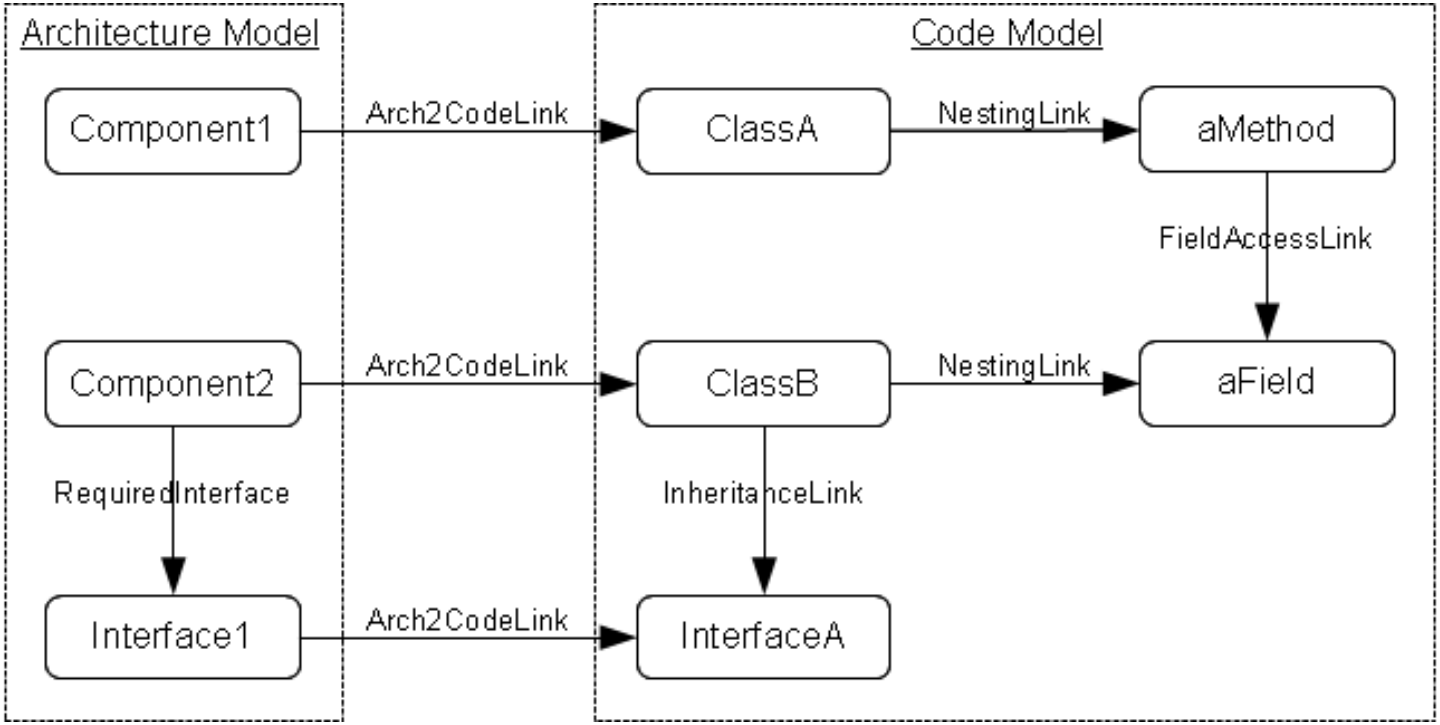
- *TracQL* is statically typed.
  - Provides typed graphs.
  - Works with concrete artifact and link types (e.g. *EMF* classes).

- *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.




```
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).

```
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)

Expressiveness

Language	Compiler Tokens	Factor
TracQL	85	1.0
Gremlin Groovy	144	1.7
Gremlin Java	232	2.7
Cypher	301	3.5

Performance [ms]

In Memory	Factor	Neo4j
58	1.0	136
184	3.2	223
126	2.2	347
-	44.0	5,982

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.