Evaluating Software Architecture using Ontologies for Storing and Versioning of Engineering Data in Heterogeneous Systems Engineering Environments

Richard Mordinyi **Estefania Serral** **Dietmar Winkler** **Stefan Biffl**

Vienna University of Technology
Institute of Software Technology, CDL-Flex, Austria

http://cdl.ifs.tuwien.ac.at

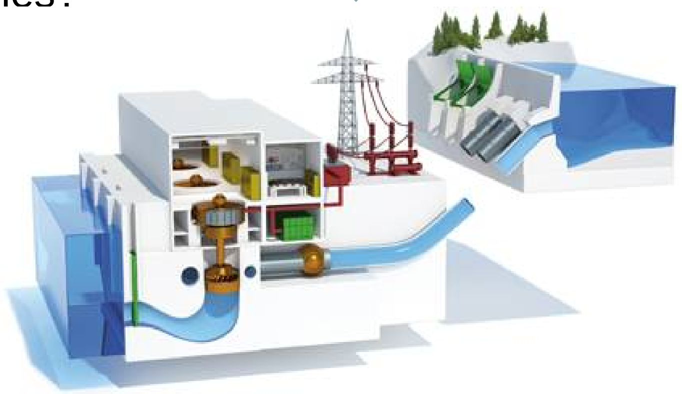
Motivation & Goals

Motivation:

- Large-Scale Engineering Projects, e.g., hydro power plants, car manufacturing plants, steal mills.
- Cooperation of different engineering disciplines.
- Disciplines have specific engineering tools.
- Manual effort required for data exchange and synchronization (high risks).

Key **research questions** focus on:

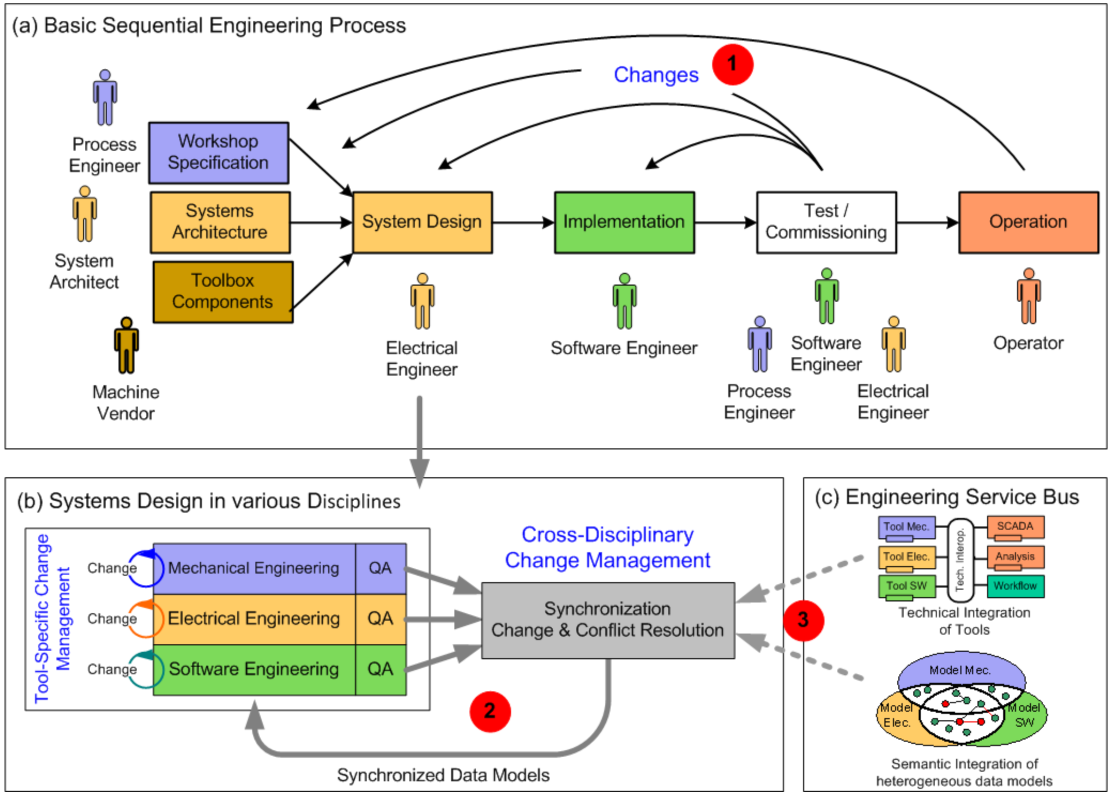
- How to enable efficient data exchange across disciplines?
- How to provide storage mechanisms to support efficient data access?



Goals of the paper:

- Overcoming technical and semantic gaps in large-scale engineering projects.
- Evaluation of storage mechanism for efficient data access.

Engineering Process and Changes

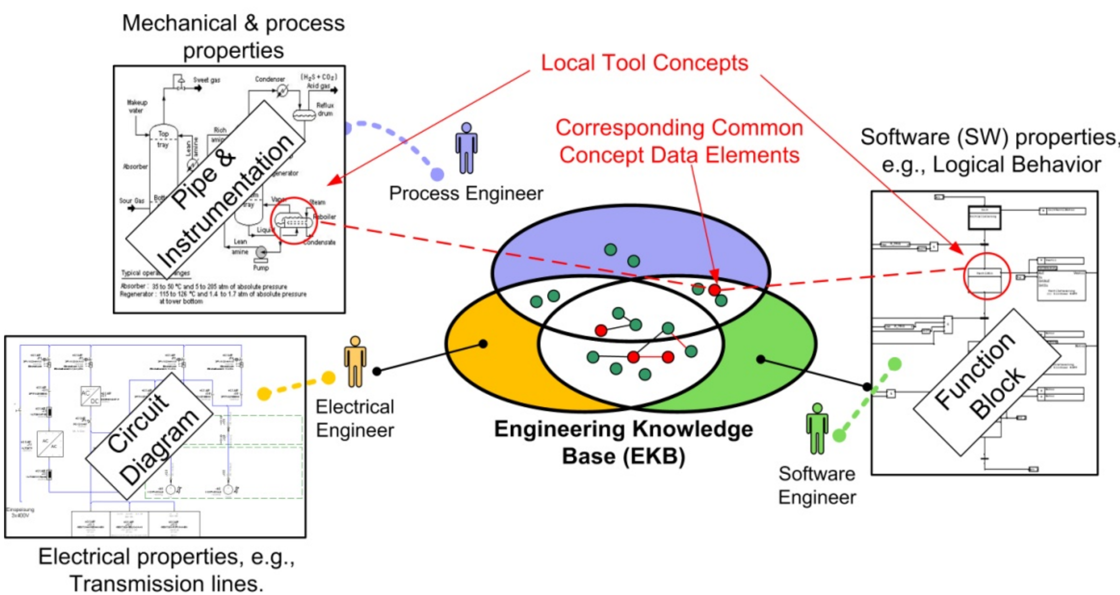


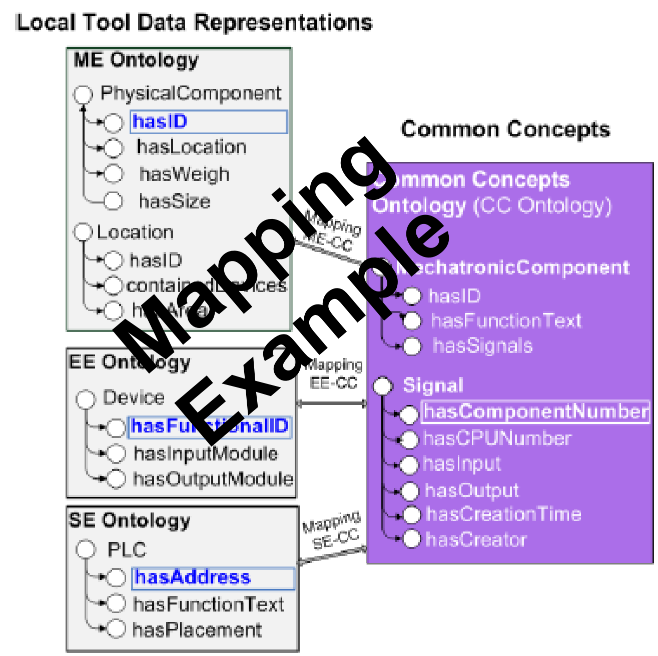
1. Sequential Engineering Processes and Changes.
2. Frequent Synchronization of heterogeneous and distributed disciplines.
3. Overcoming technical gaps of tools and semantic gaps of data models.

→ Common Concepts and the Engineering Knowledge-Base (EKB) are the foundation effective and efficient data exchange between disciplines.

EKB Concepts for Data Mapping

- Common data elements to link distributed and heterogeneous (local) data models.
- Local tool concepts vs. common data elements between two or more disciplines.
- Engineering Knowledge Base (EKB) holds common concepts and enables data integration based on semantic technologies.





→ Question: How could (versions of) data elements be stored efficiently and effectively?

Related Work and Research Issues

- **Ontology file storage**
  - XML-based semantic files, the ontology and its data instances are stored together and loaded into memory.
  - Examples: Jena, Sesame, Oracle 11g.
- **Triple storage**
  - Subject-predicate-object expressions in ontology languages, stored in specific data bases. Examples: Jena TDB, Bigdata.
- **Relational data bases**
  - Ontology storage manages concepts but individuals are stored a data base. Transformation of SPARQL queries to database queries; Examples D2RQ, Quest.
- Current evaluation studies do not include data integration scenarios.

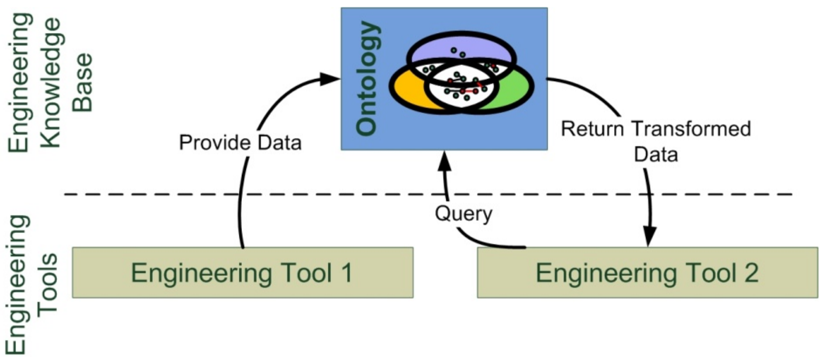
Research Issues:

- **RI-1: Data Management**: Performance of storages according to insert, update, and delete operations of EKB data elements in an integration environment.
- **RI-2: Data Analysis:** Performance of storages according to querying operations to analyze historical data (and versions), e.g., the number of changes over time or the change history of one component.

Candidate Architectures (1)

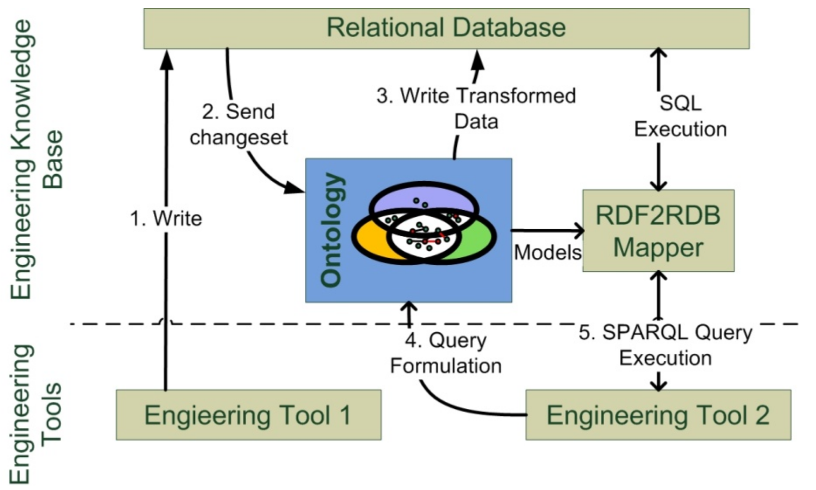
Variant A: Ontology-Based Storage

- Single ontology storage
  - Holds ontology concepts and instances in one single ontology.
  - Ontology storage: SESAME API.
  - SPARQL for transformations and queries.
- Versioning
  - Change set vocabulary.



Variant B: RDB2RDF Mapper

- Ontology Component
  - Stores and manages concepts.
- Relational Database
  - Stores and manages versioned individuals.
  - Reflects ontology models.
- Versioning modelled in the ontology

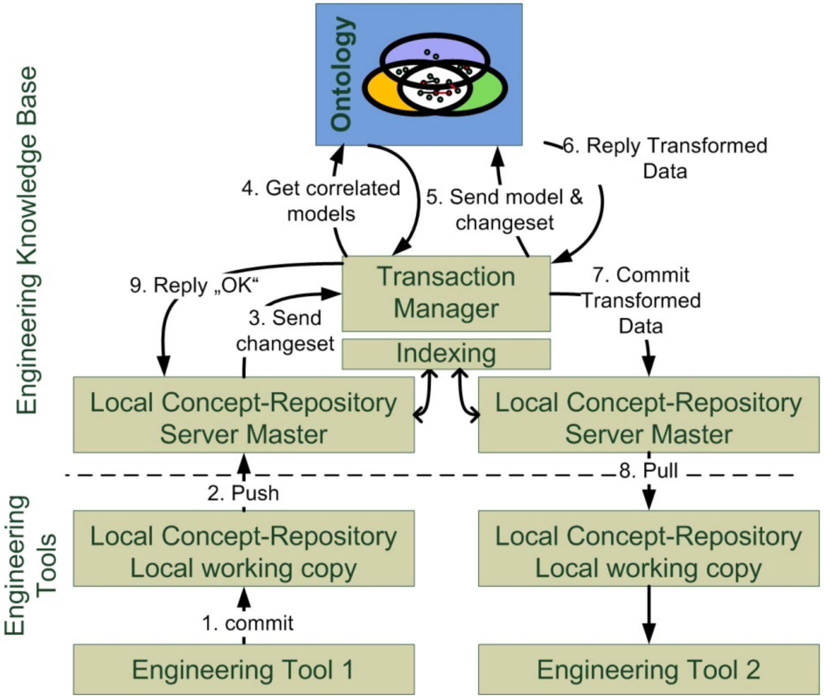


Candidate Architectures (2)

Variant C: Versioning System

- Ontology component
  - stores and manages concepts.
- Versioning System, e.g., GIT
  - Full versioning capabilities.
  - Stores and manages versioned individuals.

- Local Concept Repositories
  - Each concept in one repository.
  - Each individual one turtle file.
- Querying RDF with Apache Jena ARQ



→ Question: How do the different architecture variants perform?

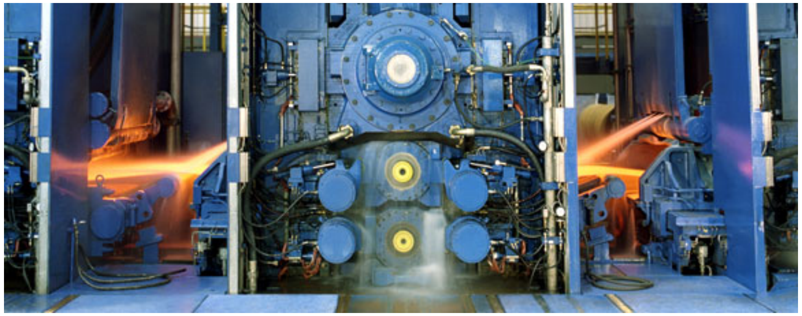
Evaluation Use Case

Goal and Context

- Investigate the performance of different architecture variants.
- Application Context: Control of a Steel Mill – 6 million data points with PLC engineering tools

Involved Tools / Data Models

- Electrical plan
- Mechanical plan
- PLC Code

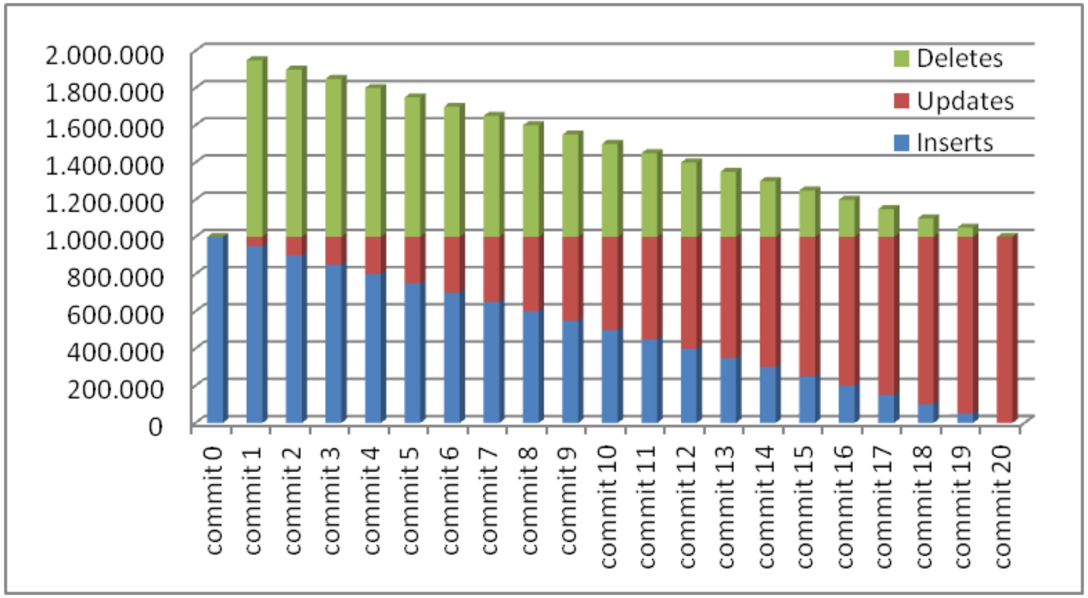


Evaluation Scenarios

- Scenario 1: Data Insertion in the Local Tool Ontologies
  → constant number of data elements.
- Scenario 2: Data transformation with increasing project size
  → increasing number of data elements, comparable to real world settings.
- Scenario 3: Historical Data Analysis Capabilities (for Scenario 1 and 2).

Data Insertion in the Local Tool Ontologies
Evaluation Scenario 1

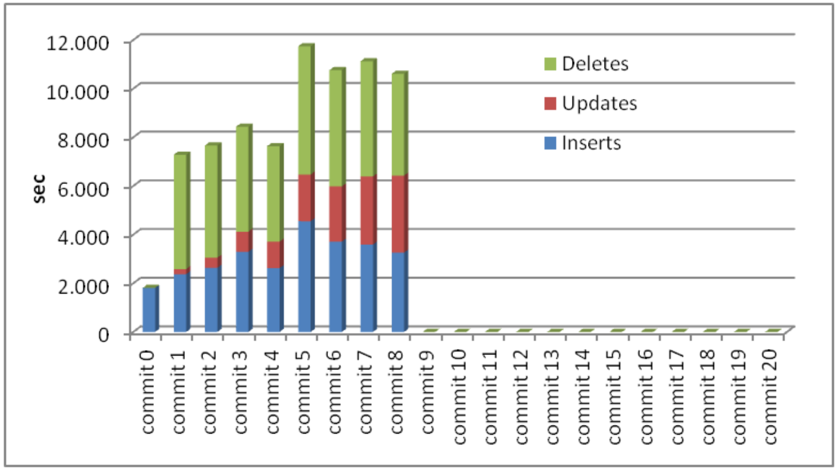
- Focus on Data Management Performance.
- Behavior of the architecture with respect to the operation types, i.e., insert, update, and delete.
- Fixed amount of data records in the system (i.e., 1 Mio data sets).
- Changing the share of add/delete/remove over time, i.e., per commit.



Data Insertion in the Local Tool Ontologies Evaluation Scenario 1 – Results (1/2)

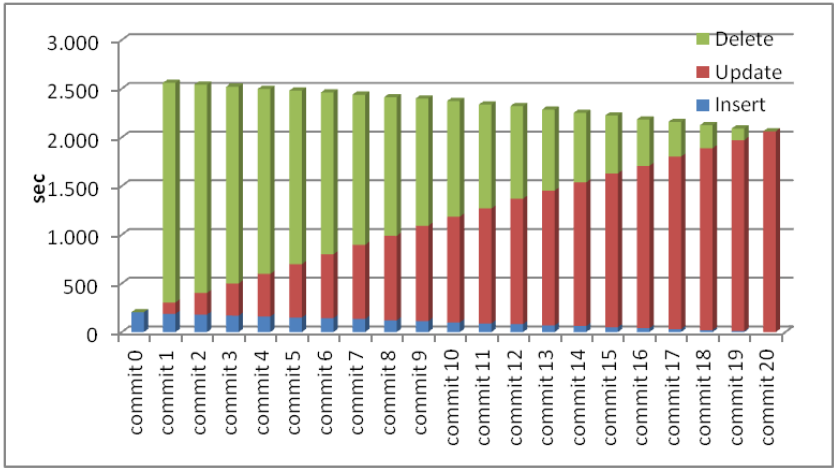
Variant A: Ontology-Based Approach

- Fluctuations in time independent of the executed operations.
- Commit 9 ended in a fatal error caused by the JRE.
- Tools & Storage Approaches: Bigdata version 1.2.3, Sesame Native store v2.6.3



Variant B: Mapper-Based Approach

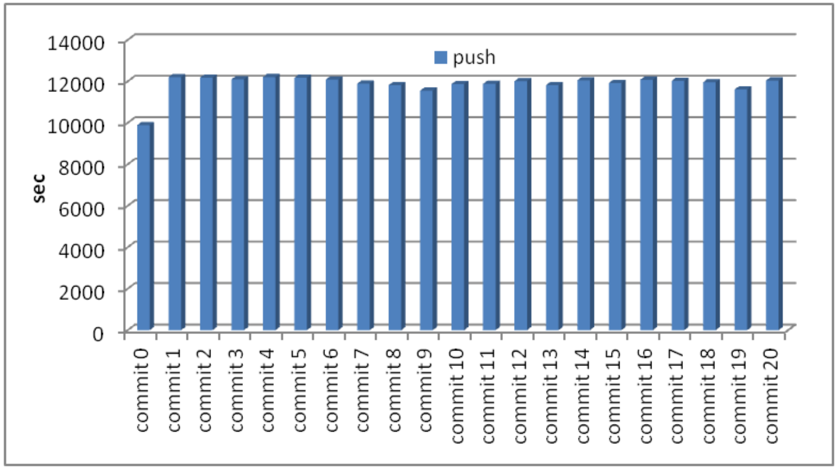
- Continuously linear behavior with respect to the changing amount of operations.
- Deleting/updating requires notable more time than inserting.
- Overall execution time 4 to 5 times less.
- Tools and Storage Approaches: D2RQ version 0.8.1, mysql 5.5



Data Insertion in the Local Tool Ontologies Evaluation Scenario 1 – Results (2/2)

Variant C: GIT-Based Approach

- Measured push-command effort.
- Almost continuously constant execution time.
- Each commit always has to cope with one million files (including versioning information).
- Large amount of files stresses file system.
- Tools and Storage Approaches: Git, Apache Jena ARQ 2.11.0

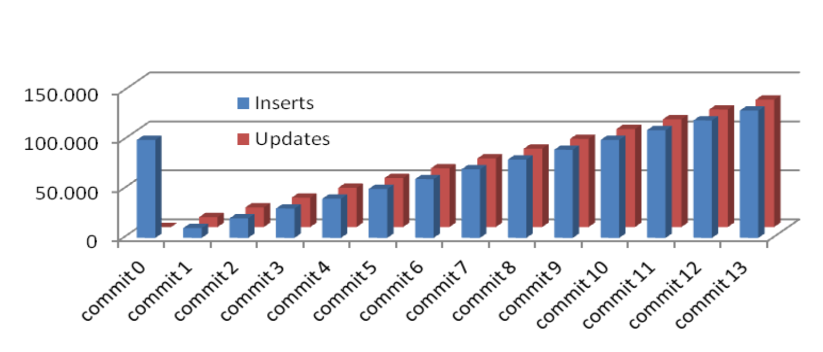


Summary of Evaluation Scenario 1

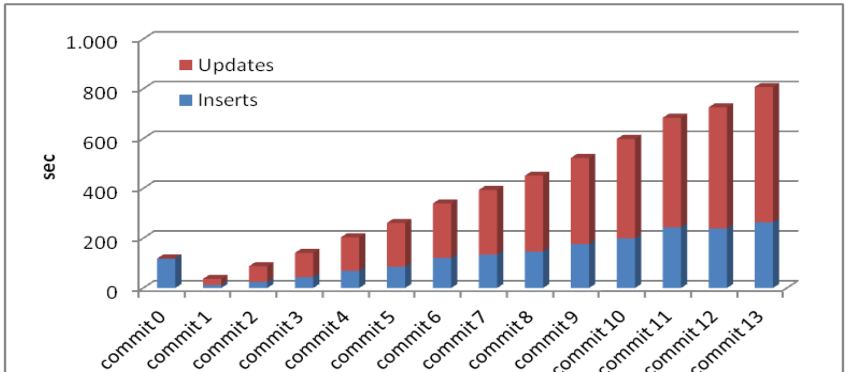
- Ontology-Based approach crashed due to resource limits.
- Mapper-Based approach required a constant execution time for handling a constant amount of 1 Mio Data Sets.
- GIT-Based approach require more time because of file system handling and version control, no separation of different operations.

Evaluation Scenario 2:
Data Transformation with Increasing Project Size

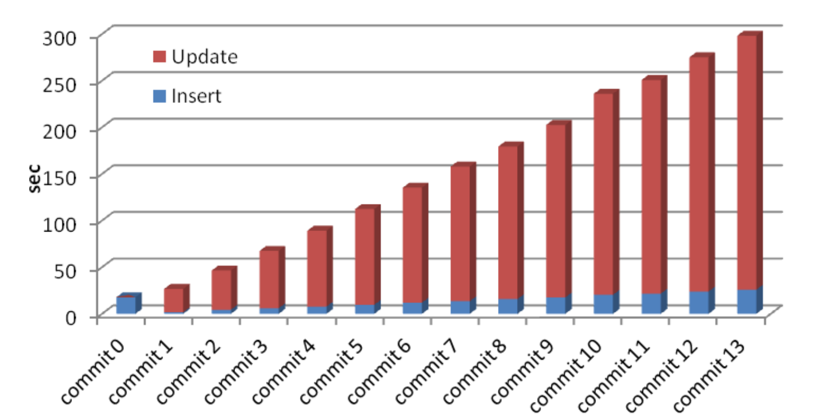
- Focus on increasing number of elements comparable to real-world projects (add and update operations).
- Each commit adds new data; Starts with 100.000; Ends at commit 13 with ~1 million data records



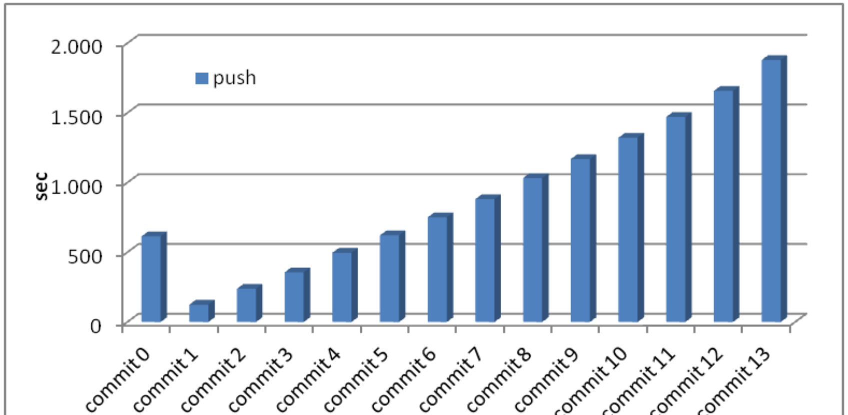
Sequence of Commits



Variant A: Ontology-Based Approach



Variant B: Mapper-Based Approach



Variant C: GIT-Based Approach

Evaluation Scenarios 1&2
Resource Consumption

Hardware Constraints

- Intel® Core™ i7-3537U Processor 2 GHz, 10 GB RAM, and 256 GB SSD harddisk
- Non-distributed environment
- Ubuntu 12.04 64bit, OpenJDK 64bit JRE 7.0_25
- java heap size of 8 GB RAM

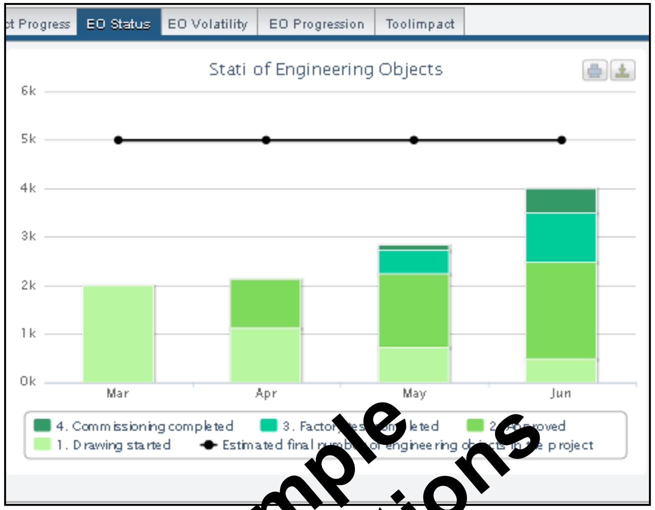
	Ontology-Based	Mapper-Based	Git-Based
Memory Consumption			
Scenario 1	5 700 MB	150 MB	< 290 MB
Scenario 2	4 600 MB	150 MB	< 148 MB
Disk usage (# Files)			
Scenario 1	700 Mio. Triples ~ 50 GB	5 GB	40 GB
Scenario 2	103 Mio. Triples ~ 6 GB	352 MB	4 GB

Evaluation Scenario 3: Querying of Data Elements

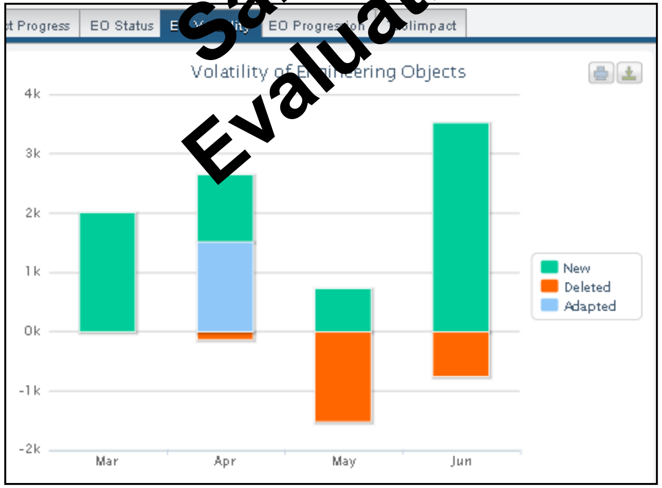
- Important issue to get (aggregated) information out of the system.

Queries for Evaluation:

- **Query 1: “What is the number of changes, deletions, and insertions during the project?”**
- Query 2: “What is the number of changes, deletions, and insertions when comparing two specific weeks?”
- Query 3: “Which components have been added, changed, or deleted on a weekly basis during the project?”
- Query 4: “Which sub-components of a specific component have been added, changed, or deleted on a week basis during the project?”
- Query 5: “How often has a specific common concept been updated during the project?”
- Query 6: “How often has a specific component been changed on a week basis during the project?”



Sample Evaluations



Evaluation Scenario 3: Storage Performance for Querying

- Performance measurement after every commit for scenario 1 with a constant number of data elements (similar observations for scenario 2).
- Depending on the **query complexity** and the **amount of involved data sets** the effort or information could increase.

Ontology-Based Approach (Search):

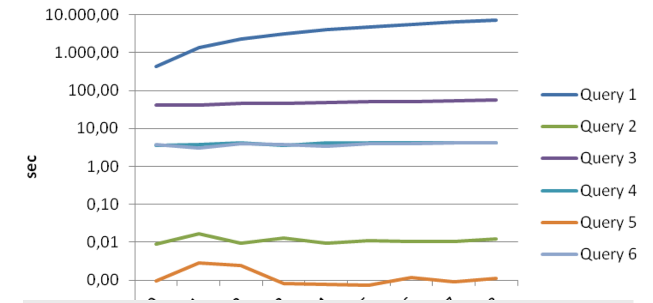
- Query 2 and 5: specific focus that allows to limit the data to be analyzed beforehand.
- Query 1 is the slowest because the entire data set has to be analyzed (logarithmic y-axis scale!)

Mapper-Based Approach (Search):

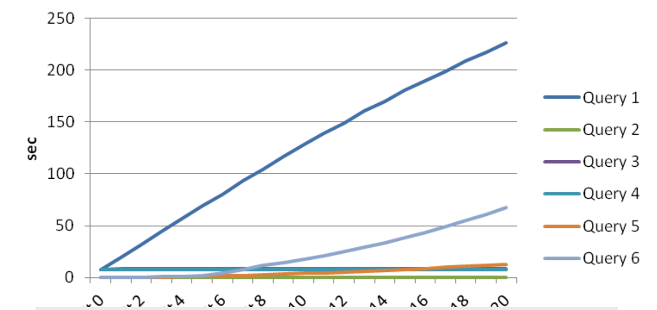
- Query 1 needs most of the time to analyze the entire data set and execute string comparison operations.

GIT-Based Approach (Search):

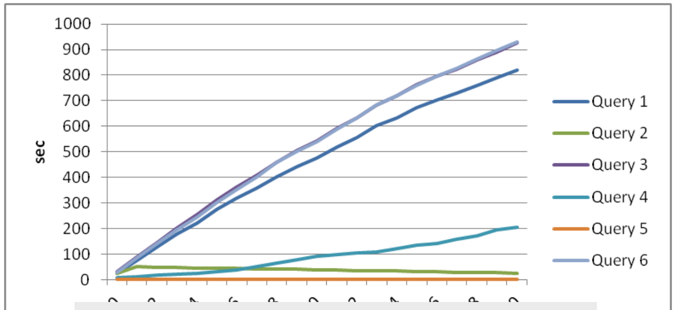
- Query 1 and 6 require more time → focus on the entire data set (and file system)



Variant A: Ontology-Based Approach



Variant B: Mapper-Based Approach



Variant C: GIT-Based Approach

Summary and Future Work

- Integrating heterogeneous disciplines and data models require efficient semantic approaches for data management and querying.
- **Evaluation of three software architectures using ontologies**
  - Ontology-based for small data models / projects.
  - RDB2RDF Mapper-based for large data models and sets.
  - Versioning System-based for supporting complex engineering processes.

	Advantages	**Disadvantages**
Ontology-based	+ Simple architecture	+ slow query execution times
Mapper-based	+ Relational databases are well researched and widely used	+ Higher complexity in application + mapping configuration requires manual adaptations + model adaptations
Git-based	+ well-established versioning system + track of changes	+ High architectural complexity + Performance strongly depends on the file system

- **Future work**
  - Add more complex data models and relationships.
  - Involvement of commercial implementations.
  - Investigation of Non-SQL storage systems.

Thank you ...

Evaluating Software Architectures using Ontologies for Storing and Versioning of Engineering Data in Heterogeneous Systems Engineering Environments

Richard Mordinyi, Estefania Serral, Dietmar Winkler, Stefan Biffl

Vienna University of Technology
Institute of Software Technology, CDL-Flex, Austria

Dietmar.Winkler@tuwien.ac.at