YARR!

Yet Another Rewriting Reasoner

Jörg Schönfisch

ORE Workshop 2013

Ulm, Germany, 22.07.2013

Agenda

- Introduction
- Query Rewriting
- Architecture & Implementation
- Performance
- Current development

Introduction

Motivation for implementing YARR

- We needed a reasoner for closed source customer projects
- Requirements
  - Stable implementation
  - Liberal licensing
  - Reuse of existing architecture
  - Suitable for a collaborative and concurrent deployment scenario

Introduction

YARR's Features

- Query answering through query rewriting (Presto algorithm)
- Support of the OWL 2 QL profile
- Persistence in a relational database system (RDBMS)
- SPARQL endpoint
- Developed as part of a larger platform

Introduction

YARR's Limitations

- SPARQL 1.1 support limited to SELECT and ASK queries
  - UPDATE, CONSTRUCT and DESCRIBE are not supported
- Property paths (especially negation of properties) not implemented
- Some built-in functions are not implemented, yet
  - abs, ceil, floor, …
  - Hashing functions
  - Some String operations
  - …

Introduction

OWL 2 QL Profile

- Enables conjunctive queries to be answered in LogSpace using standard relational database technology
- Suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to access the data directly via relational queries (i.e. SQL)
- Limitations:
  - No cardinality restrictions
  - No (disjoint) unions
  - No transitive properties
  - …
- Algorithms and prototypes for OWL 2 QL reasoning exist, but they are not easily available for commercial use in closed source products and customer projects

Query Rewriting

- Getting complete and sound answers from the database requires rewriting of the query to also fetch knowledge only implicitly defined by the TBox and the ABox
- Query Rewriting expands the query so that implicit facts are also retrieved from the ABox
- Features of Query Rewriting
  - The ABox is not involved in the reasoning step during query answering
    - Changes in the knowledge base are instantly reflected in query results
    - Changes create no overhead (e.g. removal of old inferences)
    - Only read access to the data is needed
  - No preprocessing of the data required
  - Trades less storage space for the knowledge base for more complex queries

Preliminaries

Rewriting Example

- SPARQL Query:

```
SELECT     ?athlete ?games
WHERE {    ?event hasGoldWinner ?athlete ;
                  occurredInGames ?games .
           ?athlete isRepresentativeOf Germany . }
```

- (Relevant) TBox axioms:

```
InverseProperties(hasGoldWinner,wonGold)
InverseProperties(isRepresentativeOf,hasRepresentative)
```

- Translation to Datalog:

```
q(?games,?event) :- hasGoldWinner(?event,?athlete),
                    occuredInGames(?event,?games),
                    isRepresentativeOf(?athlete,Germany)
```

Preliminaries

Rewriting Example

- Rewritten Datalog after applying the Tbox axioms:

```
q(?games,?event) :- occuredInGames(?event,?games),
                    view_1(?event,?athlete),
                    view_2(?athlete)

view_1(?event,?athlete) :- hasGoldWinner(?event,?athlete)
view_1(?event,?athlete) :- wonGold(?athlete,?event)

view_2(?athlete) :- isRepresentativeOf(?athlete,Germany)
view_2(?athlete) :- hasRepresentative(Germany,?athlete)
```

- The translation to SQL results in a query with 24 subselects and altogether 50 join operations.

Preliminaries

Query Rewriting vs Materialization

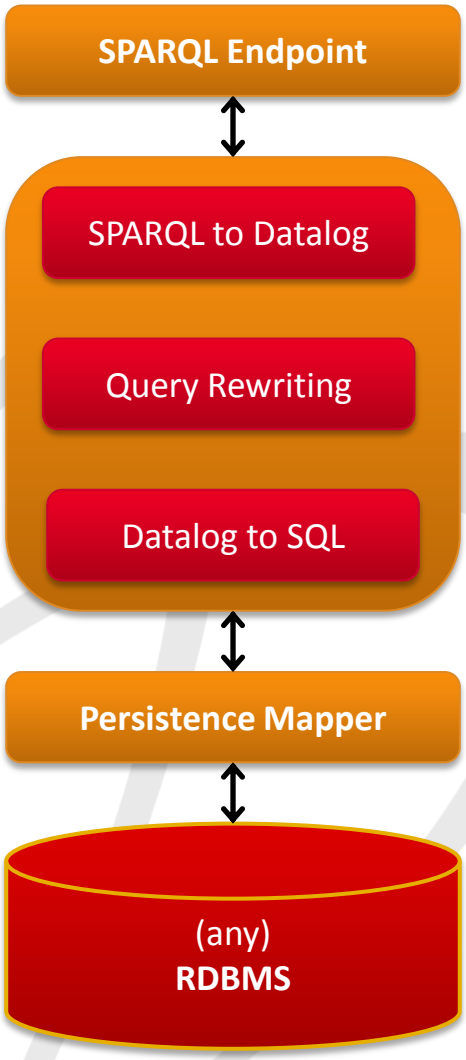
- Alternative approach to rewriting: Materialization
  - Everything that can be inferred about the data is stored explicitly beforehand
  - Queries can be executed as-is
  - Changes in the data are more complex as they influence the inferences

- Advantages of Query Rewriting
  - Changes to instances have no impact on the internal state of the reasoner
  - Highly concurrent editing and reasoning
  - Requires potentially less storage space
  - Size of the ABox is irrelevant for the reasoning step
  - Requires no additional steps during load time of the data

- Disadvantages
  - Potentially more complex queries

Architecture

Reasoner Architecture



Sesame

jOOQ

PostgreSQL

ORACLE

HyperSQL

SOFTPLANT
Living Enterprise Architecture

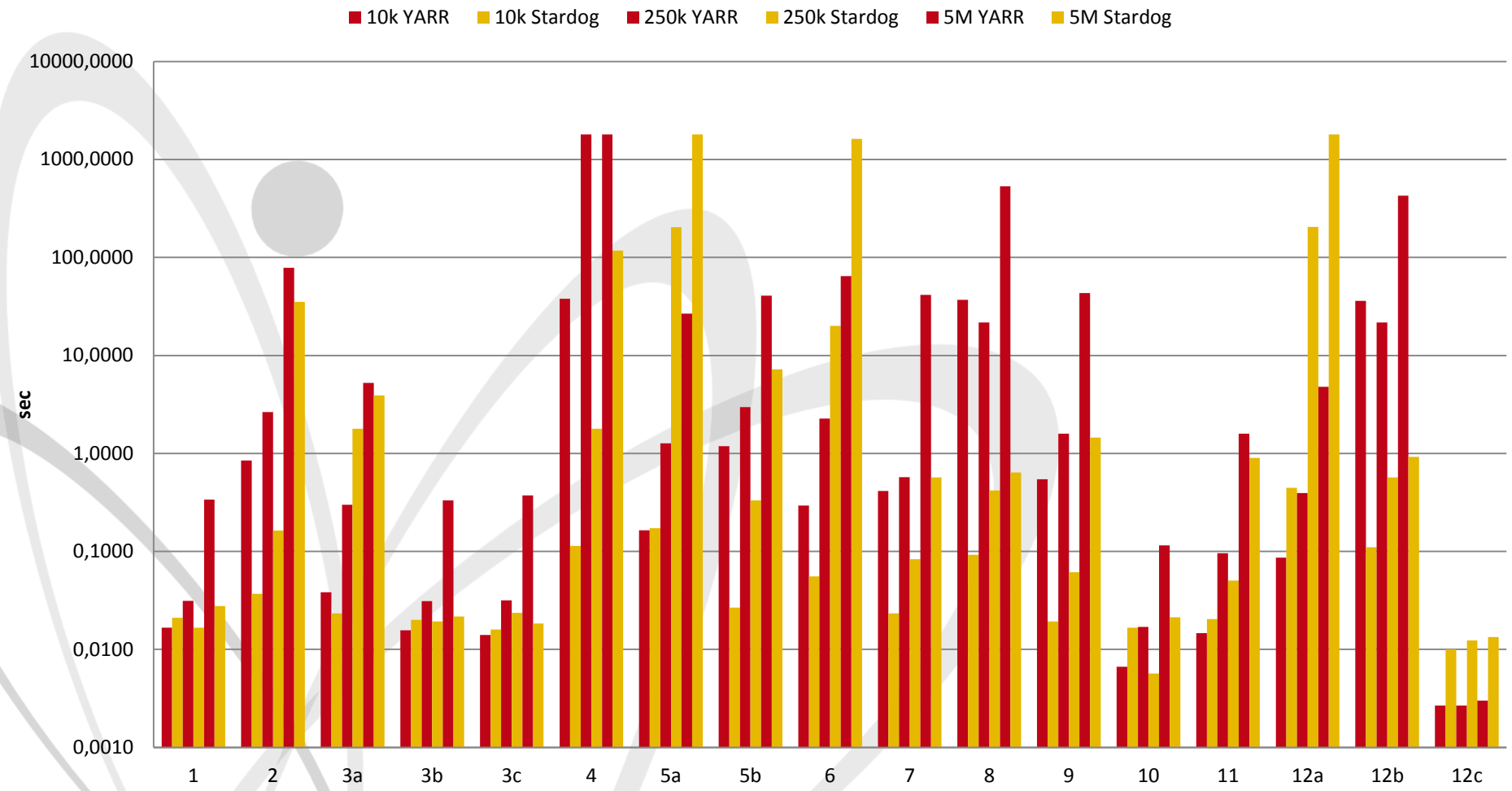
Architecture

Database Schema

- Based on the structure of the OWL 2 Metamodel
- Every part of the metamodel is stored in a separate table
  - Facilitates editing of the knowledgebase
- Consists of 43 tables
  - 6 tables for each type of entity
  - 4 tables for each type of assertion
  - 2 tables for literals
  - 21 tables for different axioms (domain, range, equivalency, …)
  - 10 auxiliary tables

Performance

Sp2Bench SPARQL Benchmark



SOFTPLANT
Living Enterprise Architecture

Current development

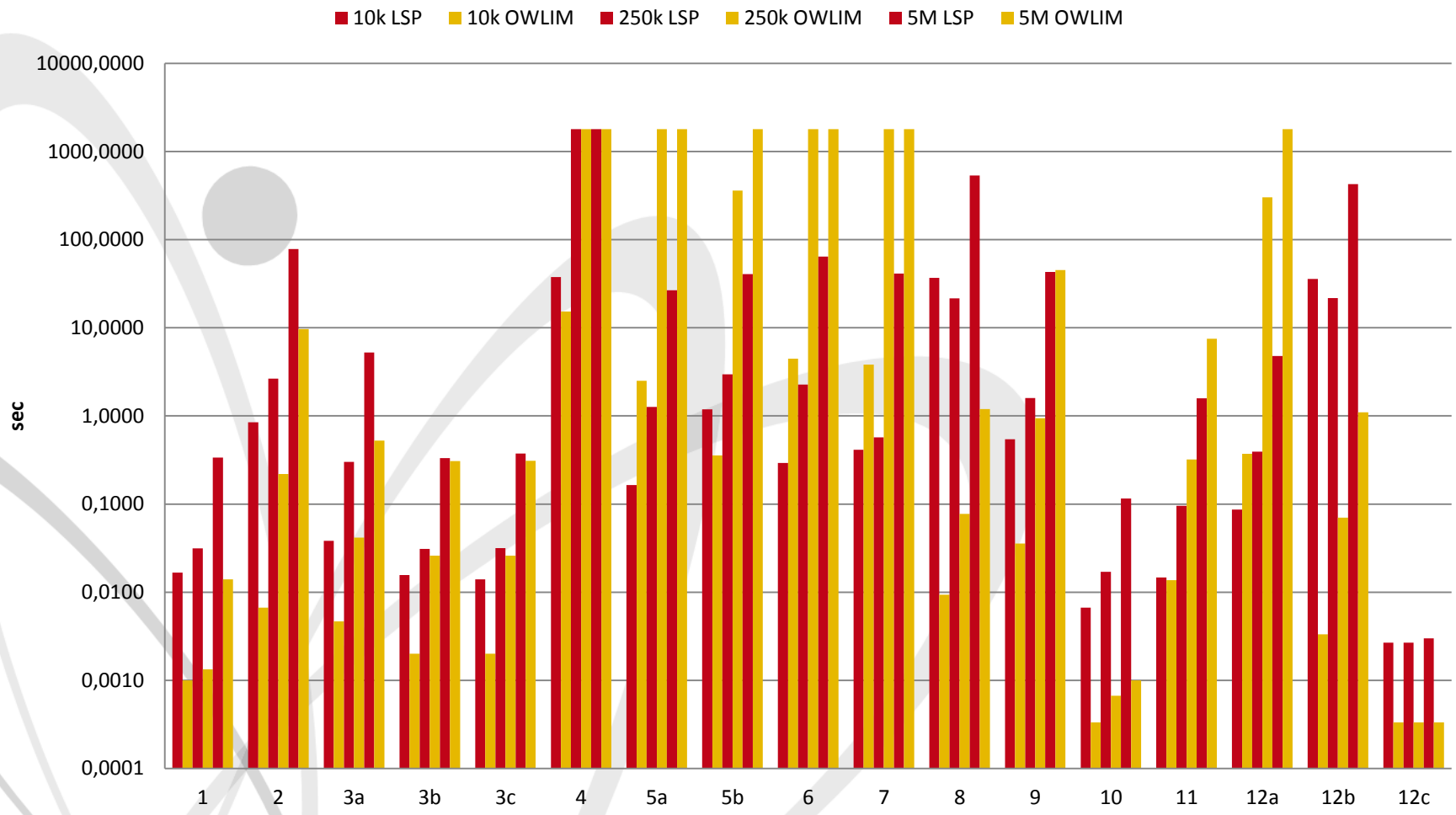
- Further support for SPARQL 1.1 Queries
  - Negation in Property Paths
  - Implement more built-in functions
  - Support for CONSTRUCT
- Performance improvement
  - Caching
  - Optimizations in the SQL translator
  - Analysis of slow queries
- Implementing adapters for ontology-based data access (OBDA)
  - Possibly supporting R2RML (W3C mapping language from RDF to relational data)

Thank You!

SOFTPLANT
Living Enterprise Architecture

Performance

Sp2Bench SPARQL Benchmark



SOFTPLANT
Living Enterprise Architecture

Architecture

Pure Java Application
UI built with Vaadin