Project Number 611125

TR – Train Benchmark

Version 0.1
29 January 2014
Draft

Public Distribution

BME

Project Partners: **ARMINES**, **Autonomous University of Madrid**, **BME**, **IKERLAN**, **Soft-Maint**, **SOFTEAM**, **The Open Group**, **UNINOVA**, **University of York**

Project Partner Contact Information

ARMINES Massimo Tisi Rue Alfred Kastler 4 44070 Nantes Cedex, France Tel: +33 2 51 85 82 09 E-mail: massimo.tisi@mines-nantes.fr	**Autonomous University of Madrid** Juan de Lara Calle Einstein 3 28049 Madrid, Spain Tel: +34 91 497 22 77 E-mail: juan.delara@uam.es
BME Daniel Varro Magyar Tudosok korutja 2 1117 Budapest, Hungary Tel: +36 146 33598 E-mail: varro@mit.bme.hu	**IKERLAN** Salvador Trujillo Paseo J.M. Arizmendiarrieta 2 20500 Mondragon, Spain Tel: +34 943 712 400 E-mail: strujillo@ikerlan.es
Soft-Maint Vincent Hanniet Rue du Chateau de L'Eraudiere 4 44300 Nantes, France Tel: +33 149 931 345 E-mail: vhanniet@sodifrance.fr	**SOFTEAM** Alessandra Bagnato Avenue Victor Hugo 21 75016 Paris, France Tel: +33 1 30 12 16 60 E-mail: alessandra.bagnato@softeam.fr
The Open Group Scott Hansen Avenue du Parc de Woluwe 56 1160 Brussels, Belgium Tel: +32 2 675 1136 E-mail: s.hansen@opengroup.org	**UNINOVA** Pedro Maló Campus da FCT/UNL, Monte de Caparica 2829-516 Caparica, Portugal Tel: +351 212 947883 E-mail: pmm@uninova.pt
University of York Dimitris Kolovos Deramore Lane York YO10 5GH, United Kingdom Tel: +44 1904 32516 E-mail: dimitris.kolovos@york.ac.uk	

Contents

Document Control

Version	Status	Date
0.1	Document outline	27 January 2014

Executive Summary

TODO

1 Overview

Scalability issues in model-driven engineering arise due to the increasing complexity of modeling workloads. This complexity comes from two main factors: (i) *instance model sizes* are exhibiting a tremendous growth as the complexity of systems-under-design is increasing, (ii) increasing *feature sophistication* in toolchains, such as complex model validation or transformations.

One of the the most computationally expensive tasks in modeling applications are *model queries*. While there are a number of existing benchmarks for queries over relational databases [6] or graph stores [3, 5], modeling tool workloads are significantly different. Specifically, modeling tools use much more complex queries than typical transactional systems, and the real world performance is affected by response time (i.e. execution time for a specific operation such as validation or transformation) than throughput (i.e. the amount of parallel transactions).

Overview To address this challenge, the Train Benchmark [7, 1] is a macro benchmark that aims to measure batch and incremental query evaluation performance, in a scenario that is specifically modeled after *model validation* in (domain-specific) modeling tools: at first, the entire model is validated, then after each model manipulation (e.g., the deletion of a reference) is followed by an immediate re-validation. The benchmark records execution times for four phases:

1. During the *read* phase, the instance model is loaded from hard drive to memory. This includes the parsing of the input as well as initializing data structures of the tool.

2. In the *check* phase, the instance model is queried to identify invalid elements. This can be as simple as reading the results from cache, or the model can be traversed based on some index. By the end of this phase, erroneous objects need to made available in a list.

3. In the *edit* phase, the model is modified to simulate effects of manual user edits. Here the size of the change set can be adjusted to correspond to small manual edits as well as large model transformations.

4. The re-validation of the model is carried out in the *re-check* phase similarly to the *check* phase.

The Train Benchmark computes two derived results based on the recorded data: (1) *batch validation time* (the sum of the *read* and check phases) represents the time that the user must wait to start to use the tool; (2) *incremental validation time* consists of the *edit* and *re-check* phases performed 100 times, representing the time that the user spent waiting for the tool validation.

Instance models The Train Benchmark uses a domain-specific model of a railway system that originates from the MOGENTES EU FP7 project, where both the metamodel and the well-formedness rules were defined by railway domain experts. This domain enables the definition of both simple and more complex model queries while it is uncomplicated enough to incorporate solutions from other technological spaces (e.g. ontologies, relational databases and RDF). This allows the comparison of the performance aspects of wider range of query tools from a constraint validation viewpoint.

The instance models are systematically generated based on the metamodel and the defined complex model queries: small instance model fragments are generated based on the queries, and then they are placed, randomized and connected to each other. The methodology takes care of controlling the number of matches of all defined model queries. To break symmetry, the exact number of elements and cardinalities are randomized.

This brings artificially generated models *closer to real world instances*, and *prevents query tools from efficiently storing* or caching of instance models. During the generation of the railway system model, errors are injected at random positions. These errors can be found in the check phase of the benchmark, which are reported, and can be corrected during the edit phase. The initial number of constraint violating elements are low (<1% of total elements).

Queries and transformations Queries are defined informally in plain text (in a tool independent way) and also formalized using the standard OCL language as a reference implementation (available on the benchmark website [1]). The queries range from simple attribute value checks to complex navigation operations consisting of several (4+) joins.

The functionally equivalent variants of these queries are formalized using the query language of different tools applying tool based optimizations. As a result, all query implementations must return (the same set of) invalid instance model elements.

In the *edit* phase, the model is modified to change the result set to be returned by the query in the recheck phase. For simulating manual modifications, the benchmark always performs hundred random edits (fixed low constant) which increases the number of erroneous elements. An edit operation only modify single model elements at once - more complex model manipulation is modelled as a series of edits.

Evaluation of measurements The Train Benchmark defines a Java-based framework and application programming interface that enables the integration of additional metamodels, instance models, query implementations and even new benchmark scenarios (that may be different from the original 4-phase concept). The default implementation contains a benchmark suite for queries implemented in Java, Eclipse OCL and EMF-IncQuery.

Measurements are recorded automatically in a machine-processable format (CSV) that is automatically processed by R [2] scripts. An extended version of the Train Benchmark [4] features several (instance model, query-specific and combined) *metrics* that can be used to characterize the "difficulty" of benchmark cases numerically, and – since they can be evaluated automatically for other domain/model/query combinations – allow to compare the benchmark cases with other real-world workloads.

2 Train Benchmark Queries

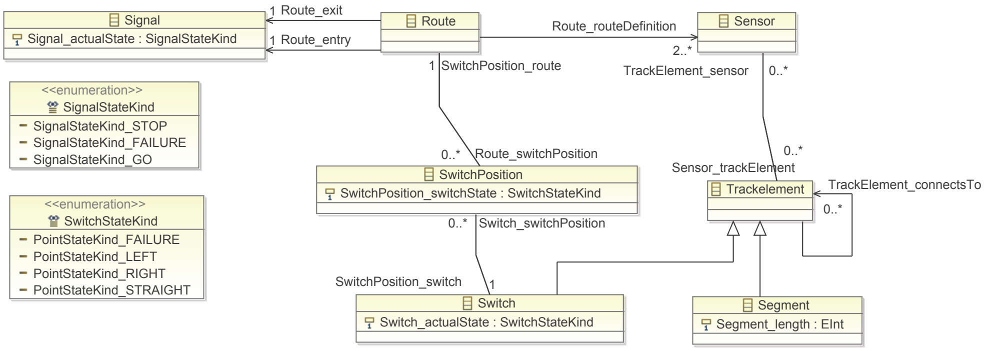

Figure 1: The metamodel of the Train Benchmark

In the following, we present the queries defined in the Train Benchmark.

We describe the semantics and the goal of each query. We also show the associated graph pattern and relational algebra query. The metamodel of the railroad system is shown in Figure 1.

2.1 Relational Schemas

For formulating the queries in relational algebra we define the following relational schemas for representing the vertices (objects) in the graph (instance model).

- $Route\,(id)$
- $Sensor\,(id, Segment\_length)$
- $Signal\,(id)$
- $Switch\,(id)$
- $SwitchPosition\,(id)$
- $TrackElement\,(id)$

The edges (relationships) are represented with the following relational schemas:

- $Route\_entry\,(Route, Signal)$
- $Route\_exit\,(Route, Signal)$
- $Route\_switchPosition\,(Route, SwitchPosition)$
- $Route\_routeDefinition\,(Route, Sensor)$
- $SwitchPosition\_switch\,(SwitchPosition, Switch)$
- $TrackElement\_sensor\,(Switch, Sensor)$
- $TrackElement\_connectsTo\,(TrackElement, TrackElement)$

2.2 Graph Patterns

Blue rectangles and arrows mark simple constraints, while red rectangles and arrows represent negative application conditions. The query return with the nodes in hollow blue rectangles. Additional constraints (e.g. arithmetic comparisons) are shown in the figure in text.

2.3 PosLength

2.3.1 Description

The *PosLength* well-formedness constraint requires that a segment must have positive length. Therefore, the query (Figure 2) checks for segments with a length less than or equal to zero. The SPARQL representation of the query is shown in Listing 1.

2.3.2 Goal

The query checks whether an object has an attribute. If it does, the value is checked. Checking attributes is a real world use case, although a very simple one. Note that simple string checking is also measured in the Berlin SPARQL Benchmark [?], and it concludes that the string comparison algorithm dominates the query time.

```
PREFIX base: <http://www.semanticweb.org/ontologies/2011/1/TrainRequirementOntology.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?xSegment
WHERE
{
    ?xSegment rdf:type base:Segment .
    ?xSegment base:Segment_length ?xSegment_length .

    FILTER (?xSegment_length <= 0)
}
```

Listing 1: The PosLength query in SPARQL

Figure 2: The PosLength query's pattern

2.3.3 Relational algebraic form

The *PosLength* query can be formalized in relational algebra as:

$$\pi_{Sensor\_id}\left(\sigma_{Segment\_length \leq 0}\left(Sensor\right)\right)$$

2.4 RouteSensor

The *RouteSensor* query is discussed in **??**.

2.5 SignalNeighbor

2.5.1 Description

The *SignalNeighbor* well-formedness constraint requires that routes that are connected through sensors and track elements have to belong to the same signal. Therefore, the query (Figure 3) checks for routes which have an exit signal and a sensor connected to another sensor (which is in a definition of another route) by two track elements, but there is no other route that connects the same signal and the other sensor. The SPARQL representation of the query is shown in Listing 2.

2.5.2 Goal

This pattern checks for the absence of circles, so the efficiency of the join operation is tested. One-way navigable references are also present in the constraint, so the efficient evaluation of these are also tested. Subsumption inference is required, as the two track elements can be switches or segments.

```
PREFIX base: <http://www.semanticweb.org/ontologies/2011/1/TrainRequirementOntology.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?xRoute1
WHERE
{
  ?xRoute1 rdf:type base:Route .
  ?xSensor1 rdf:type base:Sensor .
  ?xSensor2 rdf:type base:Sensor .
  ?xSignal rdf:type base:Signal .
  ?xTrackElement1 rdf:type base:Trackelement .
  ?xTrackElement2 rdf:type base:Trackelement .

  ?xRoute1 base:Route_exit ?xSignal .
  ?xRoute1 base:Route_routeDefinition ?xSensor1 .
  ?xTrackElement1 base:TrackElement_sensor ?xSensor1 .
  ?xTrackElement1 base:TrackElement_connectsTo ?xTrackElement2 .
  ?xTrackElement2 base:TrackElement_sensor ?xSensor2 .

  ?xRoute3 rdf:type base:Route .
  ?xRoute3 base:Route_routeDefinition ?xSensor2 .
  FILTER ( ?xRoute3 != ?xRoute1 )
```

```
  OPTIONAL {
          ?xRoute2 rdf:type base:Route .
          ?xRoute2 base:Route_entry ?xSignal .
          ?xRoute2 base:Route_routeDefinition ?xSensor2 .
          } .
  FILTER (!BOUND(?xRoute2))
}
```

Listing 2: The RouteSensor query in SPARQL

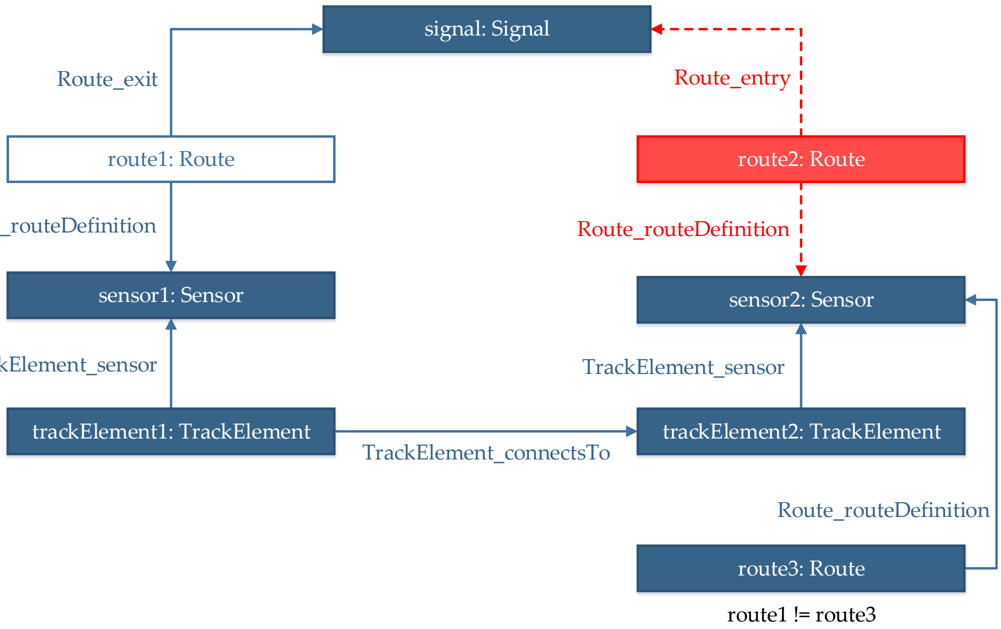

Figure 3: The SignalNeighbor query's pattern

2.5.3 Relational algebraic form

The *SignalNeighbor* query can be formalized in relational algebra as:

$$\pi_{Route\_entry.Route}\Big(\sigma_{Route\_entry.Route \neq Route\_routeDefinition_2.Route}\Big(\\ Route\_entry \bowtie Route\_routeDefinition_1 \bowtie TrackElement\_sensor_1 \bowtie \\ TrackElement\_connectsTo \bowtie TrackElement\_sensor_2 \bowtie Route\_routeDefinition_2 \triangleright \\ (Route\_exit \bowtie Route\_routeDefinition_3) \\ \Big)\Big)$$

2.6 SwitchSensor

2.6.1 Description

The *SwitchSensor* well-formedness constraint requires that every switch must have at least one sensor connected to it. Therefore, the query (Figure 4) checks for switches that have no sensors associated with them. The SPARQL representation of the query is shown in Listing 3.

2.6.2 Goal

This query checks whether an object is connected to a relation. This pattern is common in more complex queries, e.g. it is used the *RouteSensor* and the *SignalNeighbor* queries.

```
PREFIX base: <http://www.semanticweb.org/ontologies/2011/1/TrainRequirementOntology.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?xSwitch
WHERE
{
   ?xSwitch rdf:type base:Switch .

   OPTIONAL {
              ?xSensor rdf:type base:Sensor .
              ?xSwitch base:TrackElement_sensor ?xSensor .
              } .
   FILTER (!BOUND(?xSensor))
}
```

Listing 3: The RouteSensor query in SPARQL

Figure 4: The SwitchSensor query's pattern

2.6.3 Relational algebraic form

The *SwitchSensor* query can be formalized in relational algebra as:

$$\mathit{Switch} \triangleright \mathit{Sensor}$$

References

[1] The train benchmark website. `https://incquery.net/publications/trainbenchmark/full-results`, 2013.

[2] The R project for statistical computing. `http://www.r-project.org/`, 2014.

[3] Christian Bizer and Andreas Schultz. The Berlin SPARQL benchmark. *International Journal on Semantic Web & Information Systems*, 5(2):1–24, 2009.

[4] Benedek Izsó, Zoltán Szatmári, Gábor Bergmann, Ákos Horváth, and István Ráth. Towards precise metrics for predicting graph query performance. In *2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 412–431, Silicon Valley, CA, USA, 11/2013 2013. IEEE, IEEE. Acceptance Rate: 23%.

[5] Michael Schmidt, Thomas Hornung, Georg Lausen, and Christoph Pinkel. SP2Bench: A SPARQL performance benchmark. In *Proc. of the 25th International Conference on Data Engineering*, pages 222–233, Shanghai, China, 2009. IEEE.

[6] Transaction Processing Performance Council (TPC). TPC-C Benchmark, 2010.

[7] Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, Zoltán Szatmári, and Varro Daniel. An integrated development environment for live model queries. *Science of Computer Programming*, Submitted. Accepted.