The TTC 2018 Social Media Case

Georg Hinkel
FZI Research Center of Information Technologies
Haid-und-Neu-Straße 10-14, 76131 Karlsruhe, Germany
hinkel@fzi.de

Abstract

To cope with the increased complexity, models are used to capture what is considered the essence of a system. Models are often analyzed to obtain insights on the modeled system. Often, these analyses have a heuristical nature and need to be adjusted according to updated requirements and are therefore a subject of maintenance activities. It is therefore necessary to support writing such queries with adequate languages. However, in order to stay meaningful, the analysis results need to be refreshed as soon as the underlying models change. Therefore, a good execution speed is mandatory in order to cope with frequent model changes. In this paper, we propose a benchmark to assess model query technologies in the presence of model change sequences in the domain of social media.

1 Introduction

Models are a highly valuable asset in any engineering process as they capture the knowledge of a system formally and on a high level of abstraction. This abstraction allows to reason on properties in order to obtain insights on the underlying physical system through analysis.

These insights need to be refreshed as soon as the models of the system change in order to stay meaningful. However, for large systems it is often not viable to recalculate the entire model analysis for every change. Therefore, it is desirable to make use of prior computation in order to speed up later runs of the model analysis. On the other hand, information needs often change over time. This makes it very important to express such analyses in a maintainable, especially understandable form.

To assess how current modeling technologies are capable to offer a concise and understandable language for model analysis, yet still offer a good performance, we propose a benchmark. In this benchmark, two analyses should be formulated that analyze a model of a social media network. The analyses shall find the most influential posts and comments, according to selected criteria. A benchmark framework is provided that simplifies execution, correctness checks and measurements, including scalability measurements, of solutions.

The remainder of this paper is structured as follows: Section 2 introduces the metamodel that we use in this benchmark. Section 3 defines two tasks of the benchmark. Section 4 introduces the benchmark framework. Section 5 explains how solutions to the benchmark are to be evaluated.

2 Case Description

In this benchmark, we use the data from the 2016 DEBS Grand Challenge[1], adapted from the LDBC Social Network Benchmark[2]. For this version of the benchmark, we created an Ecore metamodel to describe the social network.

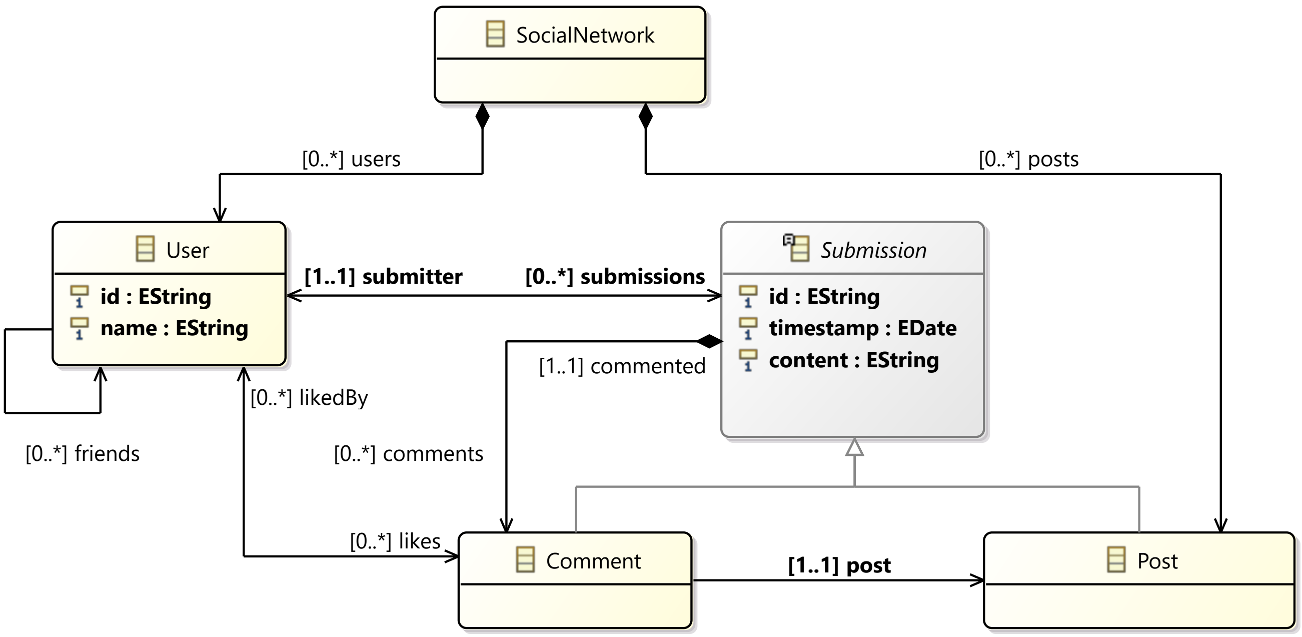

Figure 1: The metamodel of the social network

The metamodel of the social network is depicted in Figure 1. The social network consists of users, posts and comments. Users may have friends. However, due to a technical limitation of Ecore, the *friends* relationship is not an opposite of itself. Nevertheless, contestants may safely assume that it behaves as such, i.e. friendships are always bidirectional.

We provide several models and change sequences of different sizes. In each change sequence, new posts are added, new comments to existing posts are added, new users are added, users become friends or users like comments. The changes are always incremental, i.e. friendship relations do not break. Posts, comments or likes are also not withdrawn.

We are interested in the ability of different modeling languages to express queries and in their performance in batch or incremental evaluation of these queries.

3 Tasks

In the scope of the proposed benchmark, we focus on two model queries. The first query shall return the top three controversial posts, the second query shall return the most influential posts. Both queries have a result string for each timestep that is used to check the correctness of the solutions.

3.1 Task 1: Most controversial posts

We consider a post as controversial, if it starts a debate through its comments. For this, we assign a score of 10 for each comment that belongs to a post. Hereby, we consider a comment belonging to a post, if it comments either the post itself or another comment that already belongs to the post. In addition, we also value if users liked comments, so we additionally assign a score of 1 for each user that has liked a comment. The score of a post then is the sum of 10 plus the amount of users that liked a comment over all comments that belong to the given post.

[1]`http://debs.org/debs-2016-grand-challenge-social-networks/`
[2]`http://ldbcouncil.org/developer/snb`

The goal of the query is to find the three posts with the highest score. Ties are broken by timestamps, i.e. more recent posts should take precedence over older posts. The result string of this query is a concatenation of the posts ids, separated by the character `|`.

3.2 Task 2: Most influential comment

In this query, we aim to find comments that are commented by groups of users. We identify groups through the friendship relation. Hereby, users that liked a specific comment form a graph where two users are connected if they are friends (but still, only users who have liked the comment are considered). The goal of the second query is to find strongly connected components in that graph. We assign a score to each comment which is the sum of the squared component sizes.

Similar to the previous query, we aim to find the three comments with the highest score. Ties are broken by timestamps. The result string is again a concatenation of the comment ids, separated by the character `|`.

4 Benchmark Framework

The benchmark framework is based on the benchmark framework of the TTC 2017 Smart Grid case [1] and supports automated build and execution of solutions as well the visualization of the results using R. The source code and documentation of the benchmark as well as metamodels, reference solutions in NMF, example models and example change sequences are publicly available online at `https://github.com/TransformationToolContest/ttc2018liveContest`.

The benchmark consists of the following phases:

1. **Initialization:** In this phase, solutions may load metamodels and other infrastructure independent of the used models as required. Because time measurements are very hard to measure for this phase, the time measurement is optional.
2. **Loading:** The initial model instances are loaded.
3. **Initial:** An initial view is created.
4. **Updates:** A sequence of change sequences is applied to the model. Each change sequence consists of several atomic change operations. After each change sequence, the view must be consistent with the changed source models, either by entirely creating the view from scratch or by propagating changes to the view result.

In the following subsections, the change sequences, solution requirements and the benchmark configuration are explained in more detail.

4.1 Change Sequences

To measure the incremental performance of solutions, the benchmark uses generated change sequences. These change sequences are in the `changes` directory of the benchmark resources.

The changes are available in the form of models. An excerpt of the metamodel is depicted in Figure 2: There are classes for each elementary change operation that distinguish between simple assignments and collection interactions, such as adding or removing single elements or resetting, which means erasing the collection contents. The true metamodel contains concrete classes that distinguish further between the type of feature, whether it is an attribute, association or composition change. In these concrete classes, the added, deleted or assigned items are included[3]. The change metamodel also supports change transactions where a source change implies some other changes, for example setting opposite references.

Unfortunately, the implementation to apply these changes is only available in NMF. Incremental tools are therefore asked to transform the change sequences into their own change representation.

4.2 Solution requirements

The solutions are required to perform the steps of the benchmark in the order depicted above (cf. Section 4). Solutions must report the following metrics between these steps, in case of the update phase after every change sequence.

[3]In a composite insertion, the added element is contained, otherwise only referenced.

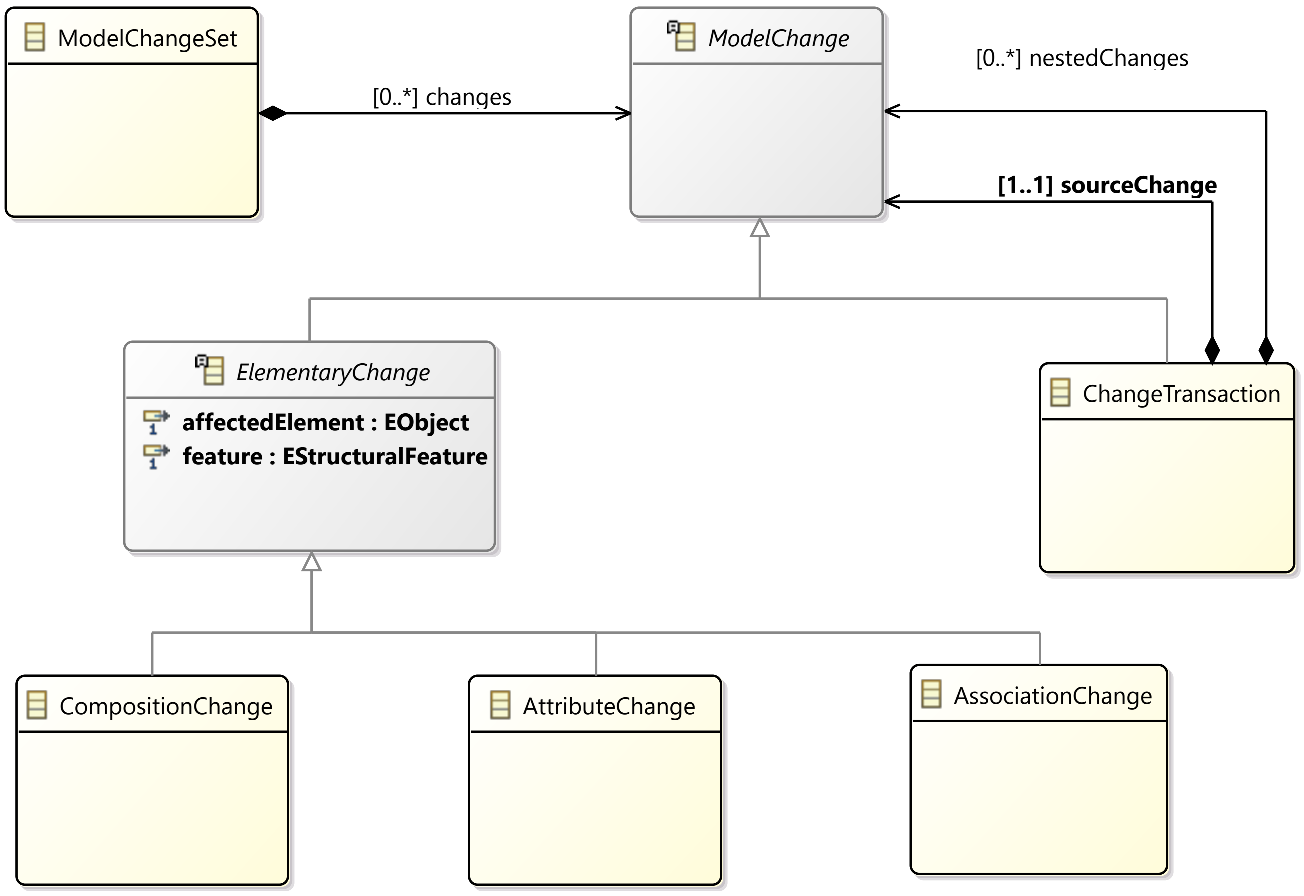

Figure 2: Metamodel of model changes (simplified)

- **Tool:** The name of the tool.
- **View:** The query that is currently computed
- **ChangeSet:** The name of the change set that is currently run
- **RunIndex:** The run index in case the benchmark is repeated
- **Iteration:** The iteration (only required for the Update phase)
- **PhaseName:** The phase of the benchmark
- **MetricName:** The name of the reported metric
- **MetricValue:** The value of the reported metric

Solutions should report on the runtime of the respective phase in integer nanoseconds (**Time**), the working set in bytes (**Memory**) and the query result string (**Elements**). The memory measurement is optional. If it is done, it should report on the used memory after the given phase (or iteration of the update phase) is completed. Solutions are allowed to perform a garbage collection before memory measurement that does not have to be taken into account into the times. In the update phase, we are not interested in the time to load models or changes or the perhaps required transformation of changes, but only the pure view update, i.e. either recomputation of the view or propagation of the change.

To enable automatic execution by the benchmark framework, solutions should add a subdirectory to the `solutions` folder of the benchmark with a `solution.ini` file stating how the solution should be built and how it should be run. An example configuration for the NMF solution is depicted in Listing 1. Because the solution contains the already compiled code, no action is required for build. However, solutions may want to run build tools like maven in this case to ensure the benchmark runs with the latest version.

```
[build]
default=MSBuild NMF.csproj /p:Configuration=Release
skipTests=MSBuild NMF.csproj /p:Configuration=Release

[run]
Q1=dotnet bin\Release\netcoreapp2.0\NMFSolution.dll
Q2=dotnet bin\Release\netcoreapp2.0\NMFSolution.dll
```

Listing 1: An example `solution.ini` file

Further, the NMF solution uses the same binaries for both queries. However, this is not mandatory. Solutions can choose to implement only one query or use multiple different executables for the different queries.

The repetition of executions as defined in the benchmark configuration is done by the benchmark. This means, for 5 runs, the specified command-line for a particular query will be called 5 times, passing any required information such as the query that should be computed, the run index, etc. in separate environment variables. All runs should all have the same prerequisites. In particular, solutions must not save intermediate data between different runs. Meanwhile, all iterations of the *Update* phase are executed in the same process and solutions are allowed (and encouraged) to save any intermediate computation results they like, as long as the results are correct after each change sequence.

The root path of the input models and changes, the run index, the number of iterations, the name of change sequences and the name of the query are passed using environment variables `ChangePath`, `RunIndex`, `Sequences`, `ChangeSet` and `Query`. To demonstrate the usage of these environment variables, the benchmark framework also contains a demo solution which does nothing but print out a time csv entry using the provided environment variables.

4.3 Running the benchmark

The benchmark framework only requires Python 2.7 or above and R to be installed. R is required to create diagrams for the benchmark results. Furthermore, the solutions may imply additional frameworks. We would ask solution authors to explicitly note dependencies to additional frameworks necessary to run their solutions.

If all prerequisits are fulfilled, the benchmark can be run using Python with the command `python scripts/run.py`. Additional command-line options can be queried using the option `-help`.

```
{
  "Queries": [
    "Q1",
    "Q2"
  ],
  "Tools": ["NMF"],
  "ChangeSets": [
    "1"
  ],
   "Sequences": 20
  "Runs": 5
}
```

Listing 2: The default benchmark configuration

The benchmark framework can be configured using JSON configuration files. The default configuration is depicted in Listing 2. When creating a new solution, we highly recommend to overwrite the contents of this configuration file locally so that by default the worked on solution is executed. In the configuration from Listing 2, both queries are computed, using only the solution in NMF, running the change sequence `1` contained in the `changes` directory 5 times each. The exact commands created by the benchmark framework are determined using the solution configuration files described below.

To execute the chosen configuration, the benchmark can be run using the command line depicted in Listing 3.

```
&> python scripts/run.py
```

Listing 3: Running the benchmark

Additional commandline parameters are available to only update the measurements, create visualizations or generate new change sequences.

5 Evaluation

Solutions of the proposed benchmark should be evaluated by their completeness, correctness, conciseness, understandability, batch performance and incremental performance.

For each evaluation, a solution can earn 5 points for Task 1 and 5 points for Task 2. In the latter, we explain how the points are awarded for Task 1. The points for Task 2 are awarded equivalently.

5.1 Completeness & Correctness

Assessing the completeness and correctness of model transformations is a difficult task. In the scope of this benchmark, solutions are checked for the correct result strings after each change.

Points are awarded according to the following rules:

- **0 points** The task is not solved.
- **1-4 points** The task is solved, but the number of elements in the result is either too high or too low.
- **5 points** The task is completely and correctly solved.

5.2 Conciseness

Detecting and especially forecasting an outage in a smart grid heavily relies on heuristics. Therefore, it is important to specify views in a concise manner.

- **0 points** The task is not solved.
- **1 point** The solution is the least concise.
- **5 points** The solution is the most concise.
- **1-5 points** All solutions in between are classified relative to the most and least concise solution.

To evaluate the conciseness, we ask every solution to note on the lines of code of their solution. This shall include the model views and glue code to actually run the benchmark. Code to convert the change sequence can be excluded. For any graphical part of the specification, we ask to count the lines of code in a HUTN[4] notation of the underlying model.

5.3 Understandability

Similarly to conciseness, it is important for maintainance tasks that the solution is understandable. However, as there is no appropriate metric for understandability available, the assessment of the understandability is done manually. For solutions participating in the contest, this score is collected using questionnaires at the workshop.

5.4 Performance

For the performance, we consider two scenarios: batch performance and incremental performance. For the batch performance, we measure the time the solution requires to create the view for existing models. For the incremental solution, we measure the time for the solution to propagate a given set of changes. Points are awarded according to the following rules:

- **0 points** The task is not solved.
- **1 point** The solution is the slowest.
- **5 points** The solution is the fastest.
- **1-5 points** All solutions in between are classified according to their speed.

The measurements for batch performance and for incremental performance are done separately. This means, solutions are allowed to run in a different configuration when competing for the batch performance than they use in the incremental setting. This is because in an application, usually only one of these aspects is particularly important.

[4]`http://www.omg.org/spec/HUTN/`

5.5 Overall Evaluation

Due to their importance, the points awarded in completeness & correctness and understandability are doubled in the overall evaluation. Furthermore, due to the importance of incremental updates, we give the incremental performance a double weight.

Thus, each solution may earn up to 80 points in total (40 for each task).

References

[1] G. Hinkel, “The TTC 2017 Outage System Case for Incremental Model Views,” in *Proceedings of the 10th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2017) federation of conferences*, ser. CEUR Workshop Proceedings, CEUR-WS.org, 2017.