The TTC 2018 Social Media Case

Georg Hinkel

Am Rathaus 4b, 65207 Wiesbaden, Germany
georg.hinkel@gmail.com

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 thus necessary to support writing model 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 queries of the benchmark. Section 4 presents the reference solution using NMF. Section 5 introduces the benchmark framework. Section 6 explains how solutions to the benchmark are to be evaluated.

In: A. Garcia-Dominguez, G. Hinkel and F. Krikava (eds.): Proceedings of the 11th Transformation Tool Contest, Toulouse, France, 29-06-2018, published at http://ceur-ws.org

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, translated series of events in the original data sources to models and change sequences and manually tuned the obtained change sequences[3].

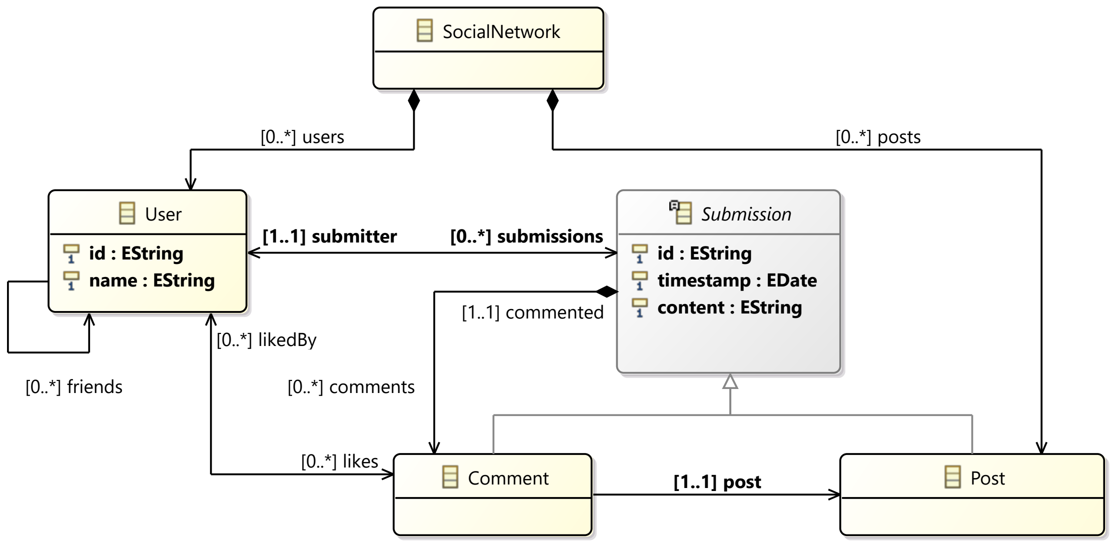

Figure 1: The metamodel of the social network

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 Queries

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. The results of both queries are concatenated to strings in order to use them for automated correctness-checks.

3.1 Query 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 per comment liked. The score of a post then is the sum of 10 plus the number of users that liked a comment `c`, summed up over all comments `c` rooted at the post.

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 `|`.

[1]`http://debs.org/debs-2016-grand-challenge-social-networks/`
[2]`http://ldbcouncil.org/developer/snb`
[3]We introduced changes that would actually change the results of the queries defined in Section 3.

3.2 Query 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 Reference Solution

As a reference, we provide a solution using NMF [1], [2] for the model representation and the standard .NET collection operators through the C# query syntax.

```
1  string.Join('|', (from post in SocialNetwork.Posts
2                 let score = post.Descendants().OfType<Comment>().Sum(c => 10 + c.LikedBy.Count)
3                 orderby (score, post.Timestamp) descending
4                 select post.Id).Take(3));
```

Listing 1: A solution for query 1 using the standard collection operators of .NET

The reference solution for query 1 is depicted in Listing 1. Line 2 of this listing computes the score of a given post by summing up the scores for the comments of a given post. To get all comments of a post, the solution simply uses the `Descendants` operation available to any NMF model element and combines it with a simple type filter. This implicitly utilizes the composition hierarchy to obtain a collection of all elements contained somewhere in the given post. Line 3 simply states that the collection of posts should be ordered by the score and the timestamp, before Line 4 specifies that we are only interested in the first three entries.

```
string.Join('|', (from comment in SocialNetwork.Descendants().OfType<Comment>()
               let layering = Layering<IUser>.CreateLayers(comment.LikedBy, u => u.Friends.Intersect(comment.LikedBy))
               let score = layering.Sum(l => Square(l.Count))
               orderby (score, comment.Timestamp) descending
               select comment.Id).Take(3));
```

Listing 2: A solution for query 2 using the standard collection operators of .NET and an implementation of Tarjan's algorithm

The reference solution for query 2 is depicted in Listing 2. It is very similar to Listing 1 in its overall structure except for the slightly more complex score calculation due to the required computation of strongly connected components. For this, we are using the class `Layering`, an implementation of Tarjan's algorithm. This implementation requires to specify the underying graph through its vertices (for each comment, this is the set of users that have liked the comment) and a function obtaining incident vertices of a given vertex (in this case the friends of a given user that have also liked the comment). The result of the scoring computation is then again ordered and the ids of the first three elements are used to return the result field.

5 Benchmark Framework

The benchmark framework is based on the benchmark framework of the TTC 2017 Smart Grid case [3] and supports automated build and execution of solutions as well as a correctness check and 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`.

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

5.1 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 the type of a feature, whether it is

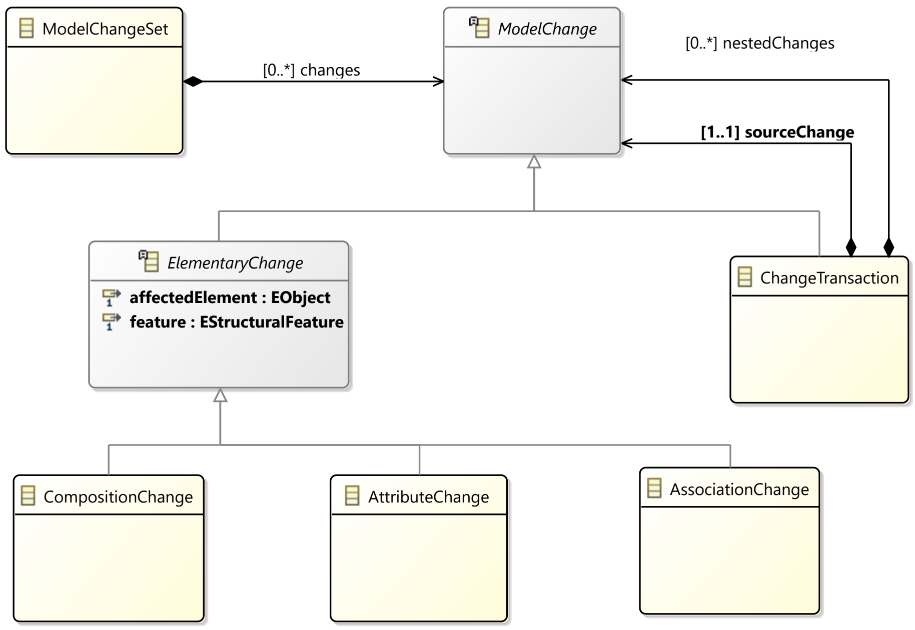

Figure 2: Metamodel of model changes (simplified)

an attribute, association or composition change. Subclasses further specify the kind of operation, i.e. whether elements are added, removed or changed. In these concrete classes, the added, deleted or assigned items are included[4]. The change metamodel also supports change transactions where a source change implies some other changes, for example setting opposite references.

Implementations to apply these changes are available in NMF and EMF. However, incremental tools are also allowed to transform the change models into their own, internal change representation. In such case, the transformation of the change representation can be excluded from the time measurements.

5.2 Solution requirements

The benchmark consists of four phases: Initialization, Load, Initial and Update where the latter is repeated 20 times. 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. In the update phase, we are only interested in the pure change propagation, i.e. loading change models does not have to be considered.

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 is provided with the benchmark resources. The query that is benchmarked, the run index as well as name and path of the change sequence is passed to the solution via environment variables. A template implementation of the benchmark is provided with the benchmark resources.

5.3 Running the benchmark

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 3: The default benchmark configuration

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

The benchmark framework can be configured using JSON configuration files. The default configuration is depicted in Listing 3. When creating a new solution, we highly recommend to overwrite the contents of this configuration file locally. In the configuration from Listing 3, both queries are computed, using only the solution in NMF, running the change sequence `1` contained in the `changes` directory 5 times each.

6 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 Query 1 and 5 points for Query 2. In the latter, we explain how the points are awarded for Query 1. The points for Query 2 are awarded equivalently.

6.1 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. A solution is awarded 5 points if the solution matches all expected results and 0 points otherwise.

6.2 Conciseness

Analytics of smart grid networks heavily relies on heuristics. Therefore, it is important to specify queries in a concise manner. The evaluation of conciseness is done by means of lines of code. This includes the queries 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[5] notation of the underlying model. The most concise solution gets 5 points, the least concise solution gets 0 points. All other solutions get points according to their lines of code.

6.3 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 submitted to the workshop and omitted otherwise.

6.4 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 calculate the query result for existing models. For the incremental performance, we measure the time for the solution to propagate a given set of changes. The fastest solution gets 5 points, the slowest solution gets 0 points. All other solutions get points according to the sum of the logarithmic benchmark results for all sizes.

6.5 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, "NMF: A Modeling Framework for the .NET Platform," Karlsruhe Institute of Technology, Tech. Rep., 2016.

[2] ——, "NMF: A multi-platform Modeling Framework," in *Theory and Practice of Model Transformations: 11th International Conference, ICMT 2018, Held as Part of STAF 2018, Toulouse, France, June 25-29, 2018. Proceedings*, accepted, to appear, Springer International Publishing, 2018.

[3] ——, "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.

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