A DSL for creating Q&A active games on Twitter

Benedikt Langer[1], David Kleindiek[2], and Lukas Schäfer[2]

[1] Universidad Complutense de Madrid, Avenida Séneca 2, 28040 Madrid, Spain
[2] Universidad Autónoma de Madrid, Ciudad Universitaria de Cantoblanco, 28049 Madrid, Spain

Abstract. The goal of this project was to design a Domain-Specific-Language (DSL), which could be used, to create Twitter bots, for playing question and answer (Q&A) games on the social media platform Twitter. Besides the DSL and the final executable program for the game, a Petri net should be designed to be able to analyze the game design.

Keywords: Twitter · DSL · Q&A.

1 Introduction

Over the course of this semester, we learned about the basics which are used in a model-driven software development. This includes what OCL-constraints are, how meta-modelling works, for what DSLs are used, how working code is generated from a meta-model and other topics.
To conclude the course, each student had to work on a project in groups of 2-3. The topic could either be selected from a already given list or a group could come up with its own topic.
Our group decided to work on the following subject:

A DSL for creating Q&A active games on Twitter

The idea is as follows:
A Twitter user starts to follow the bot. The bot then sends a welcome message with the game instructions to the said user as a direct message. As a next step, the user receives the information about the initial test of the Q&A game, such as location, maximum time and an allowed number of attempts from the bot.
To answer a test, the user must send a tweet with an "@bot", so that the bot can extract the given information from the tweet. A test is successful, when the given answer is correct, the tweet was send within the given time limit, within the coordinates of the location and at least one attempt remains.
A game is finished, when the user answers the final test of a game correctly. After this, the user can unfollow and follow the bot again, to start a new game.

2 Approach

At the beginning of the project, we drew a simple model, that represented the game, with the following elements (see Fig. 1):

- Game
- Tests
- Locations

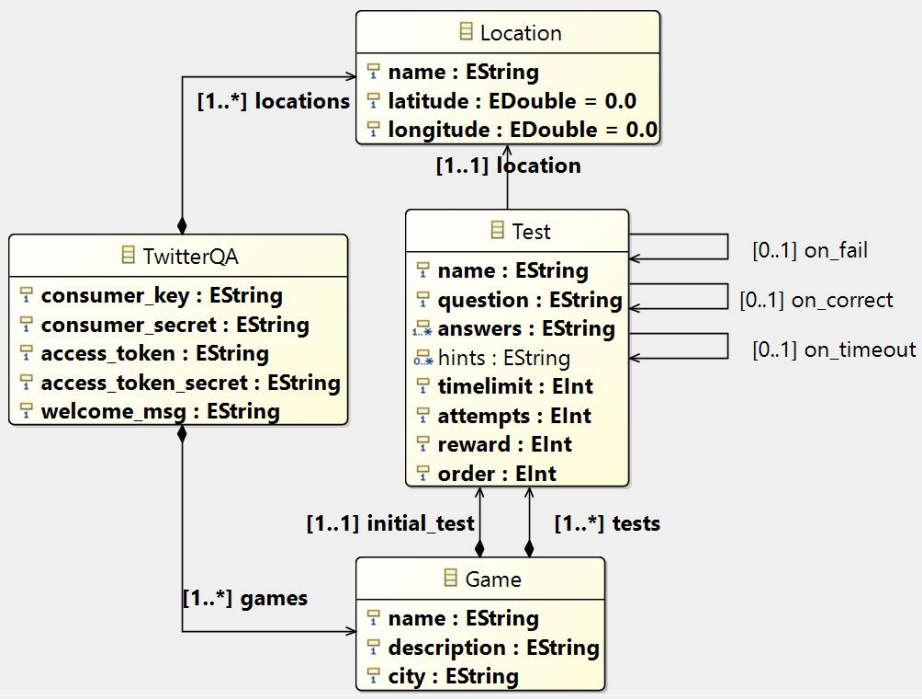

Fig. 1. Implemented Meta-Model

Furthermore we have looked into the documentation of the Twitter4j-API [1], which was suggested to us, to find suitable methods to use.
After some time, we discovered that our initial model was not correct, as the players should not be a part of the model at all. The model is only supposed to represent the game-dynamics and its required elements. The players are not a part of this model, since they already exist as users on the platform outside the model. Except from participating in the game, they do not have any influence on the elements and the dynamics of the model. Furthermore all the information and interactions between the game and the user are supported by the Twitter4j-API.

As soon as we corrected our meta-model, we started working on the ecore-file, to represent the model as accuratly as possible in the program. The ecore-file contains several OCL-constraints, which ensure the correct structure of a game. For example a test always points to an upcoming test and not to a previous one. So it doesn't matter, if the user sends the correct message, fails the test or the answer arrives after the time-limit, he always gets send to a new test.
With the final form of this ecore-file, we followed the given steps from the lecture, to generate a xtext-file, based on the ecore. We then used this generated file to write our DSL "Main.twiqa" for the Twitter Q&A. Through conversion of the Main.twiqa into a xmi-file, we could later/next use Acceleo to create the final java code for the model.
We simultaneously worked on the development of the DSL, the according trans-

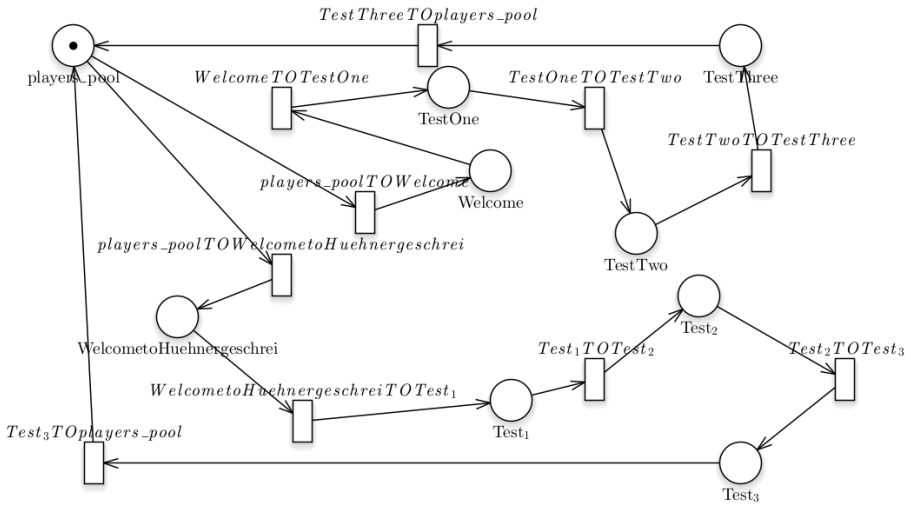

Fig. 2. Visualized Game in Petri-Net with GreatSPN

formation into working java code and the actual game-enginge, which merges the Twtter4j-API and the model code.
The game-engine works as follows:
There are several queues, which are used for the flow of a game. These are "sendWelcomeMessage", "getGameSelection", "sendTest" and "getAnswer".
As soon as a user follows the bot, the bot creates a player instance with the username as the player ID and puts this player into the "sendWelcomeMessage" queue. After the message was sent, the player gets put into the "getGameSelection" queue, where the bot waits for the player to choose a game per direct message. When the player selected a game, the program shifts the player into the "sendTest" queue, where the bot sends the information about the Initial

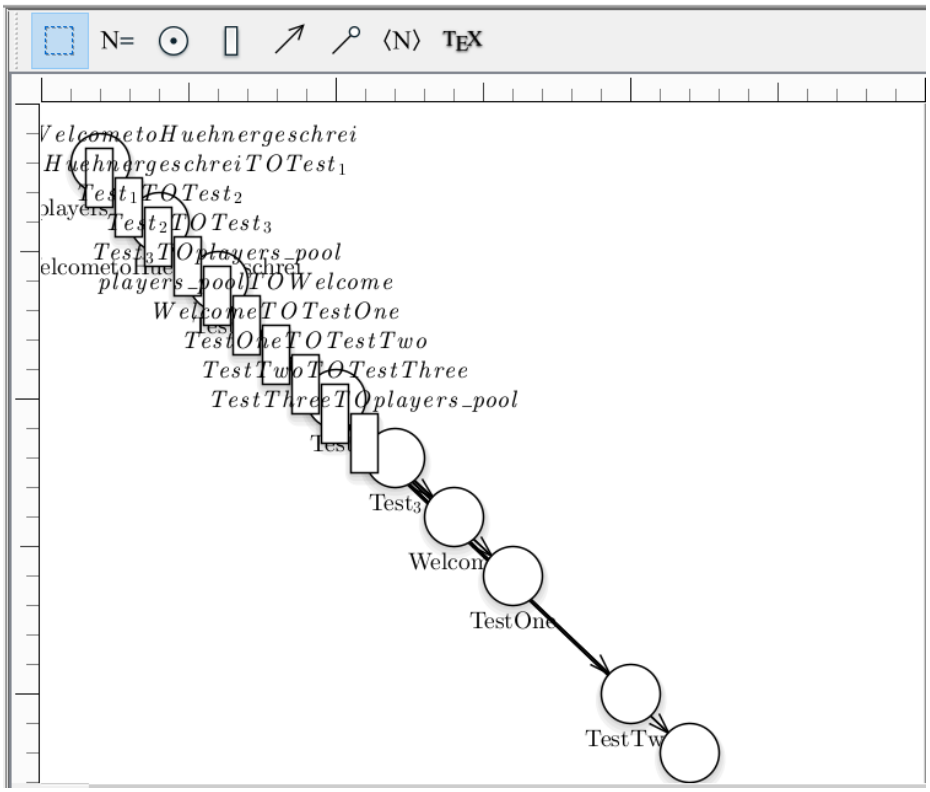

Fig. 3. Drawbacks of transformation with a python script

Test as a direct message to the user, as the program is working on this queue. In the following the player is moved into the "getAnswer" queue, where the bots awaits a tweet from the user, with the given answer. Depending on the time the tweet was send, the correctness of the answer and the number of remaining attempts, the player gets either put into the 1. "sendTest" queue (answer was correct, player has no more attempts or time limit was violated) or 2. "getAnswer" queue (answer was wrong, at least one attempt remaining and time limit was not exceeded). Once the player answers the Final Test correctly, the bot sends him a congratulations message and the according player instance gets deleted. If the player wants to play another game, he as to unfollow and follow the bot again.
In the last part we wrote a simple python script, which converts a new game into a Petri-net, so that it can be verified with GreatSPN, that every test in the game is reachable. The script takes as a input the xmi-file of the game and produces as an output a xml-file and a PNPRO, which both can be read-in and graphically represented by GreatSPN.

3 Tool support

The tools we used for our project are listed below:

- Twitter-4j API
- Twitter
- Eclipse
- Acceleo
- Jupyter Notebook
- GreatSPN

4 Evaluation

As it can be seen in the provided screenshots, the produced java code is working and the bot sends the instructions, as soon as a user follows the bot. Also the game selection and "playing" the game by sending tweets with an "@bot" works quite well (Figure 4 (L) and Figure 5 (L)).
Taking a furhter look at the screenshot in Figure 4 (R), the bot also realizes, if a wrong answer was send and sends three messages containing a text that says: Your answer was incorrect", how many attempts the user still has and also a hint. Figure 5 (R) on the other hand contains messages for two different kinds of cases:
1. The tweet of the user contained the correct answer and the bot answers with the amount of points the user now has and the next test
2. The user answer the final test correctly, receives his final points and also what the highscore for this game overall is.
Furthermore the transformation of the xmi-file for a new game into a xml-file

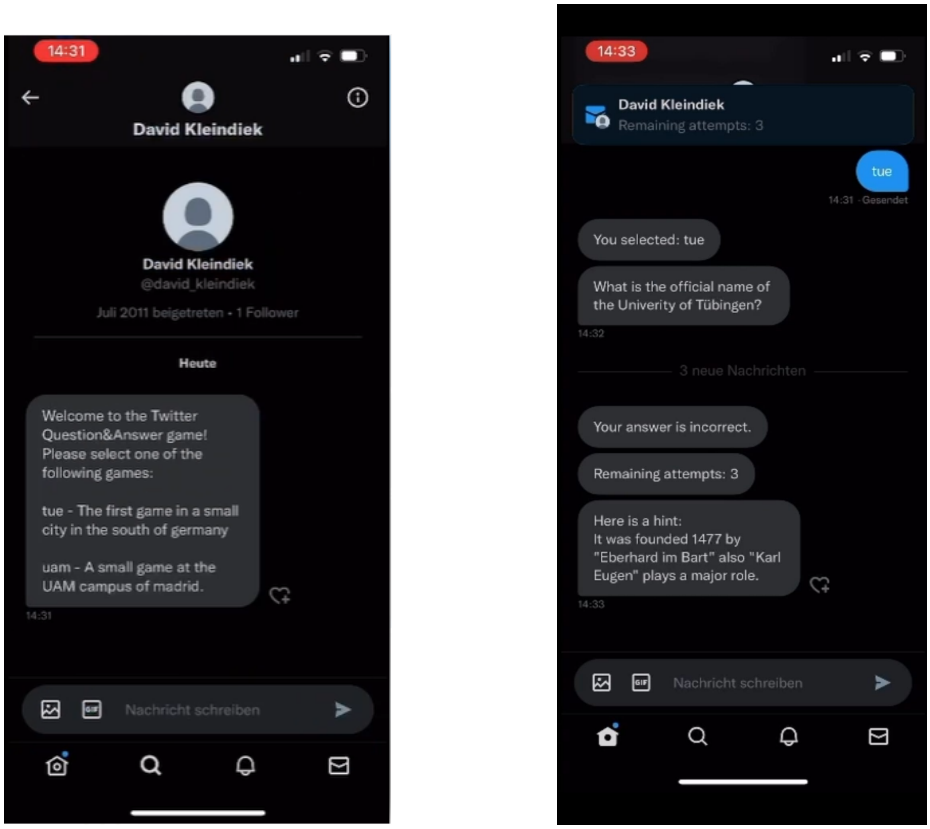

Fig. 4. Gameselection per direct message (L) and direct message, if send tweet has the wrong answer (R)

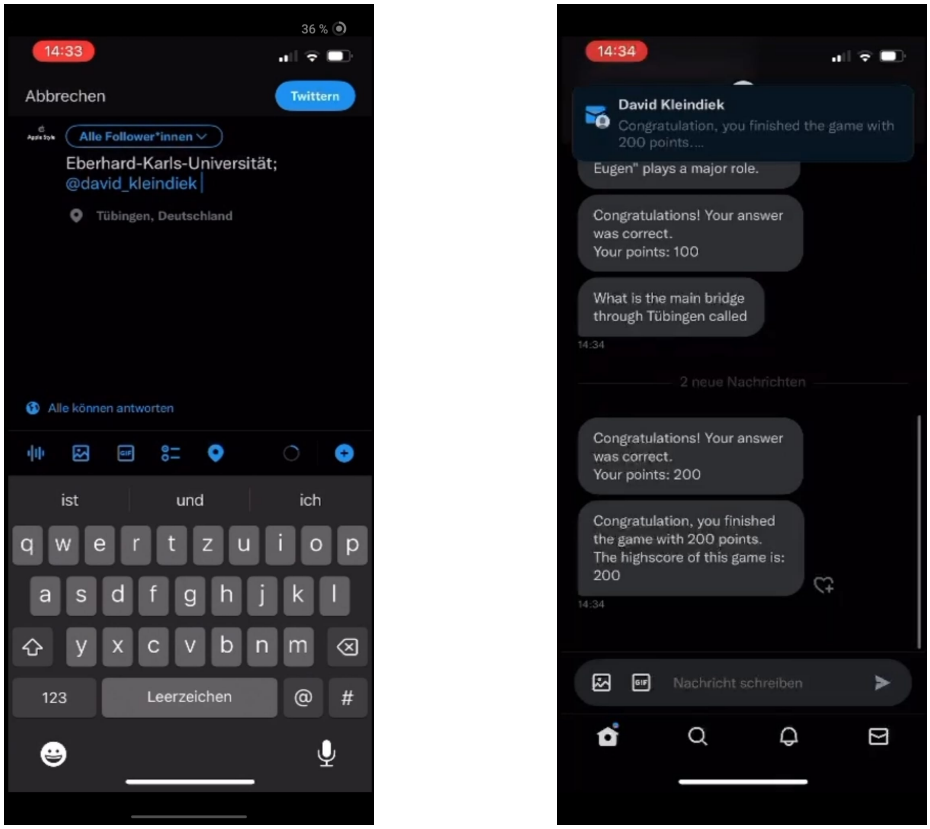

Fig. 5. Hint, if a given answer is incorrect (L) and messages, if the answer is correct and the game is finished (R)

and a PNPRO-file, which both can be read in with GreatSPN as a Petri-Nets, works quite well, as it can be seen in Figure 2.
However, a minor drawback of the transformation through a python script is that the placement of the places and transitions is quite difficult. This leads to a graph such as in Figure 3.
Altough the code and the bot work quite well, the performance is not optimal and probably will remain this way. This is because the Twitter-4j API is allowing only a few requests per minute. This leads to the fact, that the bot sends a query every 30 seconds, if he has received any new messages or tweets.

5 Conclusion

In this article we have shown, how the contents of this lecture are applied in real-world implementations of real-world models.
We used the given tools from the lecture and applied the knowledge that we gathered from the lecture, to receive a working application in the end. Even though we used tools like acceleo and in the end got a working Twitter Q&A game, we still did not use every tool, which we had at hand.
Intended is the transformation of the xmi-file into a Petri-Net would have been possible to be completed with an Atlas Transformation Language (ATL). Using this, a bad positioning of the Petri-Net elements, such as in Figure 3, could have been avoided.

References

1. Twitter4j