RPG modelling langage engineering with Web GME

Raha Naseri

Department of Computer science, University of Antwerp

Abstract

The design of complex systems by using visual formalism is always a good idea. By creating and using domain-specific modeling language(DSML), we can build the systems in modeling tools. These tools facilitate the analyzing, simulating and code generation for the system. In this report I describe the WebGME tool and discuss its suitability for modeling role playing games(RPG).

Keywords: WebGME, metamodel, domain specific modeling language

1. Introduction

WebGME is an open source cloud-based collaborative modeling tool which allows the user to design a Domain-specific modeling language and create domain-specific models for it. This software is hosted on Amazon cloud and is used for modeling complex science and engineering systems. The most outstanding characteristics of webGME are extensibility, scalability, model versioning and prototypical inheritance. WebGME is a platform independent and browser-based tool which does not require installation. The predecessor of the tool is GME which was released by the same research group but some of its limitations led them to create WebGME. Some of GME limitations were that the tool is not scalable and capable to support only small systems. Moreover, since it is not web-based, it only supports limited platforms and hence it is not possible for the users to collaborate simultaneously. The rest of the paper is structured as follows: in section 2 a brief overview of Web GME is provided, in section 3 different features of the user interface of the tool are described, in section 4 I mention some technical issues of tool and

Email address: `raha.naseri@student.uantwerpen.be` (Raha Naseri)

finally I discuss my future plans related to modeling of the RPG game by using this tool.

2. WebGME overview

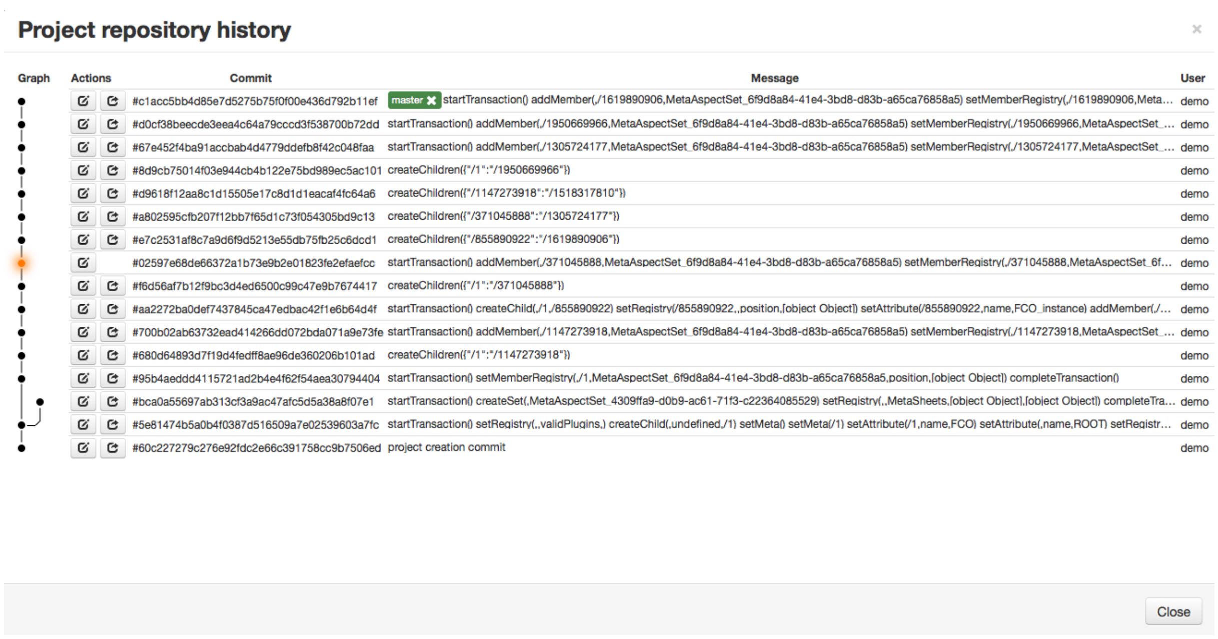
In this section I describe some characteristics of the tool. There are some interesting characteristics in webGME like version control, crosscuts and multi paradigm modeling that are briefly discussed in this section.

. *Version control:*
In webGME the models version is controlled according to Maroti Et al. In webGME "Version control is implemented by treating almost all objects in the database read only, and creating new versions of models (and updating references) whenever they are modified.the client does not need to download all models from the database, only a tiny fraction that are currently visible to the user, so it can work with large databases that would not even fit in the memory of the client."[2]
With version control users can recall and recover any previous version of the project. This can be done in the GUI of webGME by using the option project repository in the main toolbar. This option will pop up a window where shows the previous versions of the project. The tool allows to load any of these versions or create a new branch in the repository for them. Finally, Version control is mostly useful in the case of teamworking. Figure 1 demonstrates a snapshot of the project's repository history.

. *Crosscut and multi-paradigm modeling:*
By creating a crosscut user can see multiple models in a shared window and add or edit the relations between them. Suppose that we want to intermix multiple DSMLs and make a new DSML out of them which also supports the relations between them. This is possible in webGME and for this purpose we have to import the desired DSMLs or a part of them in a new project, create the new model from all the imported DSMLs and specify the relations between them in the new project. In order to visualize and edit these relations we have to create a crosscut.

3. User Interface

In order to get familiar with the GUI environment of the tool i went through the documentation of the tool [4] and some tutorials [5],[6]. Webgme



Project repository history

Graph	Actions	Commit	Message	User
●		#c1acc5bb4d85e7d5275b75f0f00e436d792b11ef	master startTransaction() addMember(,/1619890906,MetaAspectSet_6f9d8a84-41e4-3bd8-d83b-a65ca76858a5) setMemberRegistry(,/1619890906,Meta...	demo
●		#d0cf38beecde3eea4c64a79cccd3f538700b72dd	startTransaction() addMember(,/1950669966,MetaAspectSet_6f9d8a84-41e4-3bd8-d83b-a65ca76858a5) setMemberRegistry(,/1950669966,MetaAspectSet_...	demo
●		#67e452f4ba91accbab4d4779ddefb8f42c048faa	startTransaction() addMember(,/1305724177,MetaAspectSet_6f9d8a84-41e4-3bd8-d83b-a65ca76858a5) setMemberRegistry(,/1305724177,MetaAspectSet_...	demo
●		#8d9cb75014f03e944cb4b122e75bd989ec5ac101	createChildren({"/1":"/1950669966"})	demo
●		#d9618f12aa8c1d15505e17c8d1d1eacaf4fc64a6	createChildren({"/1147273918":"/1518317810"})	demo
●		#a802595cfb207f12bb7f65d1c73f054305bd9c13	createChildren({"/371045888":"/1305724177"})	demo
●		#e7c2531af8c7a9d6f9d5213e55db75fb25c6dcd1	createChildren({"/855890922":"/1619890906"})	demo
●		#02597e68de66372a1b73e9b2e01823fe2efaefcc	startTransaction() addMember(,/371045888,MetaAspectSet_6f9d8a84-41e4-3bd8-d83b-a65ca76858a5) setMemberRegistry(,/371045888,MetaAspectSet_6f...	demo
●		#f6d56af7b12f9bc3d4ed6500c99c47e9b7674417	createChildren({"/1":"/371045888"})	demo
●		#aa2272ba0def7437845ca47edbac42f1e6b64d4f	startTransaction() createChild(,/1,/855890922) setRegistry(/855890922,,position,[object Object]) setAttribute(/855890922,name,FCO_instance) addMember(,/...	demo
●		#700b02ab63732ead414266dd072bda071a9e73fe	startTransaction() addMember(,/1147273918,MetaAspectSet_6f9d8a84-41e4-3bd8-d83b-a65ca76858a5) setMemberRegistry(,/1147273918,MetaAspectSet_...	demo
●		#680d64893d7f19d4fedff8ae96de360206b101ad	createChildren({"/1":"/1147273918"})	demo
●		#95b4aeddd4115721ad2b4e4f62f54aea30794404	startTransaction() setMemberRegistry(,/1,MetaAspectSet_6f9d8a84-41e4-3bd8-d83b-a65ca76858a5,position,[object Object]) completeTransaction()	demo
●		#bca0a55697ab313cf3a9ac47afc5d5a38a8f07e1	startTransaction() createSet(,MetaAspectSet_4309ffa9-d0b9-ac61-71f3-c22364085529) setRegistry(,,MetaSheets,[object Object],[object Object]) completeTra...	demo
●		#5e81474b5a0b4f0387d516509a7e02539603a7fc	startTransaction() setRegistry(,,validPlugins,) createChild(,undefined,/1) setMeta() setMeta(/1) setAttribute(/1,name,FCO) setAttribute(,name,ROOT) setRegistr...	demo
●		#60c227279c276e92fdc2e66c391758cc9b7506ed	project creation commit	demo

Figure 1: Repository histrory

modeling has 2 sections, the model and the metamodel which are connected to each other by the protypical inheritance. Figure 2 demonstrates an example in the GUI of the tool.

There is a main toolbar on the top of the page. By using this toolbar we can create, import, export projects, edit the shape or color of the elements appearance and make a project read only etc. On the top right side of the page is the object browser window showing composition hierarchy rooted by Root and inheritance hierarchy rooted by FCO(first class object). This window is the core of the tool since it shows the tree structure of the system. On the top left side of the page there is the mode selector panel. When we split the tools main sheet to 2 parts, there is 2 mode selector panels for each of the sheets. Below this panel there is the part browser which contains parts of the model language which can be dragged and dropped into the sheet and actually this is the way of adding elements to the model. We can also drag and drop elements from the composition hierarchy and by means of the relation arrows existing in the main toolbar we can define the relations

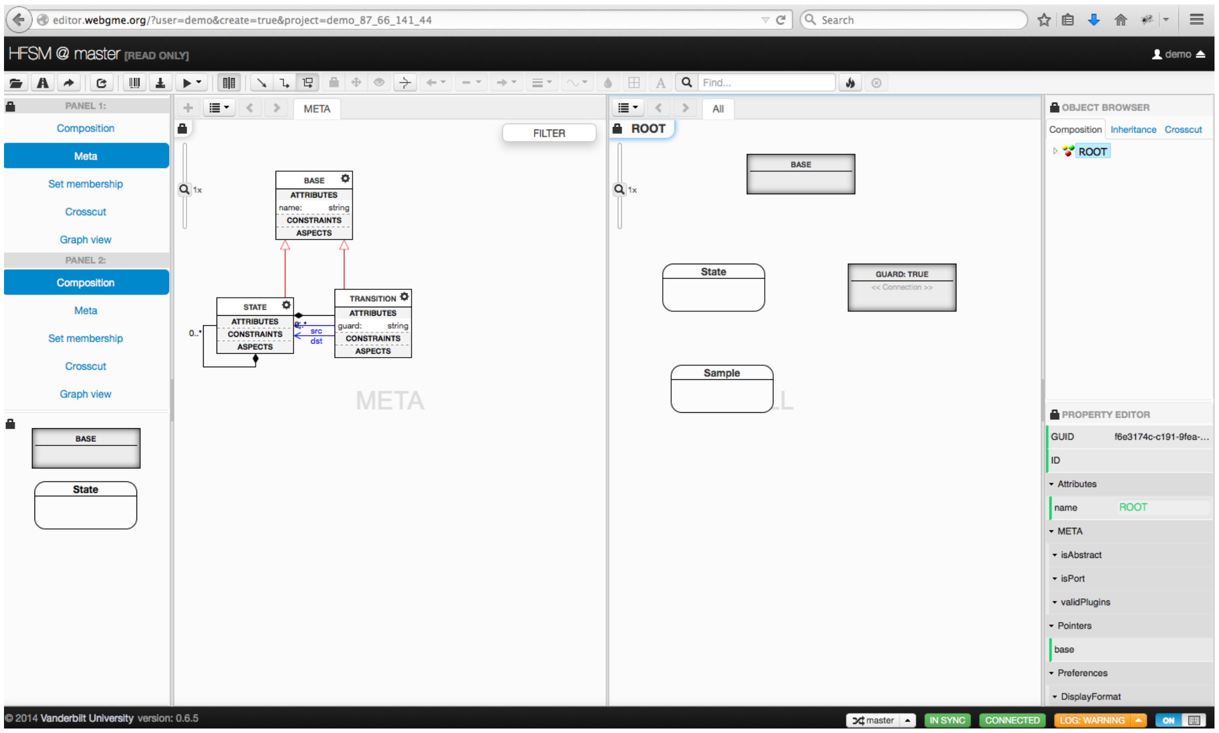


Figure 2: WebGME GUI

and connections between them. Moreover we can add or edit the attributes, constraints and aspects of every object in the class diagram. The down right side of the page is the Property editor. Here, we can edit the properties of the elements in Meta or Composition. In order to change the appearance of the model elements we can use decorator and SVGIcon. After completing the project, we can export it from the tool in a .json file format.

4. The schedule

Initially by experimenting with webGME, I came across some issues which show that the tool is not yet completed. Some of these issues are the following: - It seems that the tool has an option to sign in, but practically it does not work. Currently the tool does not support creating an account. Every user can work with a default account called demo. - An other issue is that it is not possible to choose the appearance of the model elements from my own Icons. As a result there is only a very limited choice for Icons in the tool. -

Since the sources for WebGME are very limited and thus not many examples can be found. - Finally after discussing some issues with the developers of the tool, I concluded that currently WebGME is not mature enough to fully support the modeling of an RPG game. Moreover they recommended to use GME instead of webGME for now (since they are still working on some open issues).

. Based on the above, I concluded that by using WebGme it is only possible to build the abstract syntax and concrete syntax of the RPG game. For building the transformation rules another tool must be used. There are two other tools that came along with GME called UDM and GREaT [3][8]. Initially in order to build the rule-based transformation the output of GME which is a UML class diagram in XML format, should be manually imported to the UDM tool. After that UDM's output can be manually imported in the tool GREaT where the transformation of the RPG game can be created. Finally, my plan is to use webGME for the first part instead of GME and convert the output of webGME (JSON file) to XML and give it as input to UDM and then to GreAT respectively. Figure 3 demonstrates this procedure.

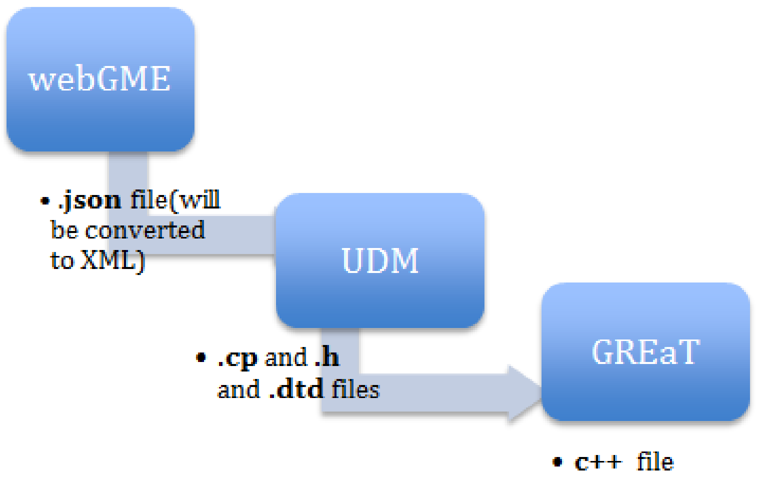

webGME
• .json file(will be converted to XML)
UDM
• .cp and .h and .dtd files
GREaT
• c++ file

Figure 3: Future plan

References

[1] M. Maroti, T. Kecskes, R. Kereskenyi, B. Broll, P. Volgyesi, L. Juracz, T. Levendoszky, A. Ledeczi, *Next Generation MetaModeling: Web- and Cloud-based Collaborative Tool Infrastructure*,Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA.

[2] M. Maroti, T. Kecskes, R. Kereskenyi, P. Volgyesi, A. Ledeczi, *Online Collaborative Environment for Designing Complex Computational Systems*,Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA.

[3] E. Magyari, A. Bakay, A. Lang, T. Paka, A. Vizhanyo, A. Agarwal, G. Karsai, *UDM: An Infrastructure for Implementing Domain-Specific Modeling Languages*,Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA.

[4] URL: https://github.com/webgme/webgme/blob/master/docs/tutorial.html

[5] URL: https://www.youtube.com/watch?v=0YCo4cpoB7k

[6] URL: https://www.youtube.com/watch?v=7eFP75n5lTY

[7] URL: http://www.isis.vanderbilt.edu/tools/GReAT