An iStar 2.0 Editor Based on the Eclipse Modelling Framework

Martin Liebenberg, Kerstin Roßmaier, and Gerhard Lakemeyer

Knowledge-Based Systems Group (KBSG), RWTH Aachen University, Ahornstrae 55, D-52056 Aachen, Germany, {liebenberg,gerhard}@kbsg.rwth-aachen.de kerstin.rossmaier@rwth-aachen.de https://www.kbsg.rwth-aachen.de/

Abstract. During the search for a replacement of our outdated OpenOME editor to obtain a new editor for iStar 2.0, we became aware of certain tools from the Eclipse Modeling Project. These tools provide the possibility to build modern graphical editors in a convenient way with little programming overhead. They also meet our requirements for an editor which should be flexible in changing the metamodel of iStar and easy to extend. In this paper, we report on our quest to find an Eclipse modelling framework which makes it easy to create and adapt an up-to-date iStar 2.0 editor.

Keywords: iStar 2.0, Model-Driven Software Development, Eclipse Modeling Project, iStar Tools

1 Introduction

For many years we have used an old version of the Eclipse-based OpenOME [16] which was adapted by our group around 2009. Today, its code base is not in a good shape and is based on old libraries. Nevertheless, in our project we needed further adaptation of this code base for a multi-agent simulation [12] and decided to get a new editor last year.

First of all, we looked for the newest OpenOME version. The SourceForge page [1] and the Trac [2] of OpenOME showed that the last modifications were made in 2011. The projects of Eclipse release a major version every year [3] and adaptation to the new frameworks, libraries and even Java language features [5] may be expensive. Furthermore, OpenOME has problems with maintainability and stability [10]. Finally, we wanted to have an editor which supports the newest version of iStar because recently the specification of iStar 2.0 had been released.

Starting from scratch with a new editor for the iStar 2.0 dialect we need [12] has the promise of a code base which is better to maintain. Such an editor should be light-weight and better to adapt to updates of libraries and language changes. OpenOME was based on the Graphical Modeling Framework (GMF) which is part of the Graphical Modeling Project [4]. To this project several frameworks are

related that generally follows the concept of Model-Driven Software Development (MDSD) [11], where programs are automatically generated using a formal model. In our case this would mean that the code for editors is generated equipped with interfaces to embed them into other software. With these modelling techniques we hoped to get a new editor fast. In the following sections we report our findings about these frameworks and our choice of an iStar editor.

2 Searching for a Suitable Framework

As stated above, we want to use MDSD for the easy creation of an editor for iStar 2.0 diagrams. To this end, we created a formal model of a fragment of iStar 2.0 which is interesting for us. Then in a further step, we investigated three different frameworks for creating graphical editors using this model. Here we only consider Eclipse-based tools, because for us it is important that we can connect the editor to other software which is quite easy to achieve within Eclipse.



Fig. 1. iStar 2.0 fragment in Ecore

All the frameworks we examined utilize an Ecore metamodel, which comes from the Eclipse Modeling Framework (EMF) [14], to describe the metamodel for the editors. Figure 1 shows our Ecore model of a fragment of iStar 2.0 as an UML diagram that we used for editor generation. Due to lack of space, here we concentrate on a fragment of the iStar 2.0 language as defined in [8] and omit elements we are currently not using and additional elements we need for our simulation. For instance, in our metamodel we need a link which indicates which agent provides a resource. In the following we present the three frameworks we looked at: the Graphical Modeling Framework, Graphiti and Sirius. All three frameworks are based on EMF and the Graphical Editing Framework (GEF) [13]. Ultimately, we chose Sirius for our editor implementation, which is described in Section 3.

2.1 Graphical Modeling Framework

As the first of the three frameworks we considered GMF [4] because OpenOME was based on it. It uses the Ecore model as base model and utilizes models from EMF for describing the editor. The Graphical Definition Model contains information about the representation of elements from our Ecore model. Further, the Tooling Definition Model describes the behavior of tools available in the editor. GMF uses other models as well as ones which depend on one another and therefore require changes after changing one of these models. Finally, after deriving several models with different purposes, GMF automatically generates Java code for the defined editor.

GMF works well for small Ecore models that result in simple editors so that little customization of the EMF-based models is required. For larger domains, like ours, for which we need to add constraints, such as tasks appearing only inside agents, the Graphical and Tooling Definition Model, in addition to the Mapping Model which links these two models, quickly become large and hard to maintain. In addition, changing one model usually requires changes to others, which makes GMF inflexible, especially when changing the metamodel. Since flexibility is a requirement of ours, this makes GMF not a good fit for our purpose. An advantage of GMF is, that advanced feature customization outside of the framework's options can simply be added to the Java code GMF creates.

2.2 Graphiti

In contrast to GMF, Graphiti [9] does not provide code generation. Instead developers have to write code using a Java API that hides the complexities of working with EMF and GEF. Same as GMF the Ecore model is the base of Graphiti and we can describe an editor by implementing so called Diagram Type Agents for all elements required in our editor. These contain the behavior of that element described through features such as create, delete, or resize. The Graphiti Rendering Engine takes care of the graphical representation that we can customize in a Pictogram Model.

Because Graphiti does not provide automatic code generation, it loses an advantage of MDSD. Standard code, for example the feature delete, has to be rewritten for every Diagram Type Agent. In addition, Graphiti requires more knowledge of Java and plug-in development with Eclipse compared to GMF. But it is more flexible than GMF, since it is not based on a complex network of models. Changes to the metamodel only require changes to our code, and Graphiti allows incremental development of an editor. It is also very customizable, since we can add additional functionality to our code. Developers might prefer Graphiti over GMF, because it is more similar to conventional coding.

2.3 Sirius

The third framework we considered is Sirius [15] which differs from the first two, since it does not use automatic code generation like GMF or require manual coding like Graphiti. Instead the Ecore model and a Viewpoint Specification Model (VSM) have to be specified in the Sirius Tooling framework; no code is written or automatically created. The Sirius Runtime, which is independent from the tooling framework, then interprets the XML-like VSM at runtime. When describing the VSM, we can access the Ecore model through a Domain-specific language (DSL) that is updated automatically after changes to the metamodel happen. Using the DSL and the Acceleo Query Language (AQL) [7], we can define elements of the editor and specify their appearance and behavior. Because the Sirius Runtime automatically updates the editor while changes are made to the VSM, it is possible to simultaneously work on the description of an editor and create diagrams with that editor.

Because Sirius does not require any code for an editor and allows changing the editor while using it, it is the most flexible out of the three frameworks. Therefore Sirius allows changes to the Ecore model more easily than GMF and Graphiti. The VSM has an intuitive structure, so it does not require much practice to work with. A risk of the lack of code generation might be that we have to rely on Sirius for many customization options. But so far the limitations seem to be only of a visual nature, as we discuss in the next section. For the reasons listed above, we decided on the Sirius framework for an implementation of our editor.

3 Implementation in Sirius

As stated in the last section, creating an editor with Sirius requires little familiarization with the framework. We simply linked our Ecore model to a new Sirius project and accessed all its elements through a Domain-specific language (DSL). After that we defined the node and link types needed in our editor. For example, we specified a task as a node-type by setting its Domain Class to *istarmodel. Task.* This description refers to the class *Task* in our Ecore model called *istarmodel* from Figure 1. Then we set a relation between Task and the IstarDiagram class through the expression *feature:elements*, which refers to the reference *elements* in Figure 1 that Task inherits from the class Intentional Element. Links between nodes are defined similarly, by referring to the Ecore model and the node types to which they belong; for example the link neededBy has to reference the Task and *Resource* node types. Next, we specified creation tools that allow users the construction of diagrams with the editor. For example, the Task Creation Tool assigns the new task a name and links it to an actor. Figure 2 shows the editor that we created with Sirius. In the center the editor displays the current diagram, while all diagrams and their elements are listed on the left. Creation tools for all elements are listed on the right side and the bottom shows properties of the currently selected element, such as references to other elements.

As shown above, providing basic functionality for an editor is very simple with Sirius. Adding more complex features, such as the capability for roles to



Fig. 2. Diagram Editor for iStar dialect, created with Sirius

contain elements or adding validation of the diagram, does not require much work either, thanks to the DSL and the query language AQL. Eclipse extension points and the ability to call custom Java functions allow even more customizations. The only limitations we found are of a visual nature, for example adding custom edge decorators outside of the ones supplied by Sirius would require code changes in the Sirius core [6]. Because of that, the AND-refinement has a different end decoration, namely an arrow instead of the T-shaped arrowhead, than in the iStar 2.0 documentation [8]. Apart from that, Sirius' advantage is that it does not have any code generation, which makes the editor creation very easy and flexible. Making changes to the Ecore model requires only a few minor adjustments to definitions we wrote with the DSL. The VSM has a good structure that speeds up work. In addition, Sirius saves created models in two files, thus separating visual from semantic information. This gives us easy access to the model which we require for translating iStar models into other languages [12]. These advantages make Sirius a good choice for our project.

4 Conclusion

In this paper, we investigated three Eclipse-based frameworks that generally follow the concept of Model-Driven Software Development. For our purposes we hoped to find a framework which makes creating an editor for an iStar 2.0 fragment easy and flexible. We think that we have found a suitable framework with the Sirius framework. The only drawback so far is that the AND-refinement can not have the right T-shaped arrowhead in its representation in the editor. To make this possible one has to change the core of the frame work. But this would not be a good option if we want a maintainable solution. At the moment we only have a basic stand-alone editor. In future work, we need to connect our Sirius-based editor with other tools needed for multi-agent simulation.

References

- OpenOME SourceForge Files. https://sourceforge.net/projects/openome/files, 2011. Accessed 23-April-2017.
- OpenOME Trac Source Browser. https://se.cs.toronto.edu/trac/ome/browser, 2011. Accessed 23-April-2017.
- Eclipse Version History. https://wiki.eclipse.org/Simultaneous_Release, 2017. Accessed 23-April-2017.
- GMF-Graphical Modelling Framework. http://www.eclipse.org/modeling/gmp, 2017. Accessed 23-April-2017.
- What's new in JDK 8. http://www.oracle.com/technetwork/java/javase/8-whatsnew-2157071.html, 2017. Accessed 23-April-2017.
- Florian Barbin. Bug: Custom "Source Arrow" and "Target Arrow" for an edge style. https://bugs.eclipse.org/bugs/show_bug.cgi?id=484508, 2016. Accessed 23-April-2017.
- Stephane Begaudeau. Acceleo Query Language Query and navigate in EMF models. https://www.eclipse.org/acceleo/documentation/aql.html, 2015. Accessed 23-April-2017.
- Fabiano Dalpiaz, Xavier Franch, and Jennifer Horkoff. iStar 2.0 Language Guide. arXiv preprint arXiv:1605.07767, 2016.
- Markus Gerhart and Marko Boger. Concepts for the model-driven generation of graphical editors in eclipse by using the graphiti framework. *International Journal* of Computer Techniques, 3(4), 2016.
- Tong Li, Alicia M. Grubb, and Jennifer Horkoff. Understanding Challenges and Tradeoffs in iStar Tool Development. Proceedings of the Ninth International i* Workshop (iStar 2016), 1674:49–54, 2016.
- Stephen W Liddle. Model-Driven Software Development. In Handbook of Conceptual Modeling, pages 17–54. Springer, 2011.
- Martin Liebenberg, Victor Mataré, Klaus Baier, and Gerhard Lakemeyer. Towards Using i* for Modeling Mega-Urban Processes. In CEUR Proceedings of the Seventh International i* Workshop (iStar14), volume 1157, 2014.
- 13. Dan Rubel, Jaime Wren, and Eric Clayberg. *The Eclipse Graphical Editing Framework (GEF)*. Addison-Wesley Professional, 2011.
- 14. Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.
- Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. Sirius: A rapid development of DSM graphical editor. In *Intelligent Engineering Systems (INES)*, 2014 18th International Conference on, pages 233–238. IEEE, 2014.
- Yijun Yu and Jennifer Horkoff an Eric Yu. OpenOME, An Open-source Goal and Agent-Oriented Model Drawing and Analysis Tool. CEUR Proceedings of the 5th International i* Workshop (iStar 2011), 766:154–156, 2011.