Towards Automatic Generation of Web-based Modeling Editors

Manuel Wimmer[1], Irene Garrigós[2] and Sergio Firmenich[3,4]
[1]BIG, TU Wien, Austria
wimmer@big.tuwien.ac.at
[2]WaKe Research, University of Alicante, Spain
igarrigos@dlsi.ua.es
[3] LINVI, Universidad Nacional de la Patagonia San Juan Bosco, Puerto Madryn
[4] LIFIA, Universidad Nacional de La Plata and CONICET Argentina
sergio.firmenich@lifia.info.unlp.edu.ar

Abstract. With the current trend of digitalization within a multitude of different domains, the need raises for effective approaches to capture domain knowledge. Modeling languages, especially, domain-specific modeling languages (DSMLs), are considered as an important method to involve domain experts in the system development. However, current approaches for developing DSMLs and generating modeling editors are mostly focusing on reusing the infrastructures provided by programming IDEs. On the other hand, several approaches exist for developing Web-based modeling editors using dedicated JavaScript frameworks. However, these frameworks do not exploit the high automation potential from DSML approaches to generate modeling editors from language specifications. Thus, the development of Web-based modeling editors requires still major programming efforts and dealing with recurring tasks.
In this paper, we combine the best of both worlds by reusing the language specification techniques of DSML engineering approaches for generating Web-based modeling editors. In particular, we show how to combine two concrete approaches, namely Eugenia from DSML engineering and JointJS as a protagonist from JavaScript frameworks, and demonstrate the automation potential of establishing Web-based modeling editors. We present first results concerning two reference DSML examples which have been realized by our approach as Web-based modeling editors.

1. Introduction

With the current trend of digitalization in a multitude of domains, effective approaches to capture domain knowledge are a must. Modeling languages, especially domain-specific modeling languages (DSMLs) [1], are considered an important foundation to involve domain experts in the system development. A DSML consists of *(i)* an abstract syntax that defines the concepts of a language and the relationships between them, as well as the rules that establish when a model is well formed, *(ii)* a concrete syntax that establishes the language notation which is used by the users of the language, and *(iii)* the semantics, i.e., how the modeling concepts are interpreted. In model-driven engineering (MDE) [2], the abstract syntax of a DSML is defined in terms of a metamodel. The concrete syntax can be both; textual and graphical; or even a mixture of both. To support the development of DSMLs as well as supporting tools, various metamodeling tools have emerged that allow to create textual DSMLs (e.g., consider EMFText [3] and Xtext [4]) and graphical DSMLs (e.g., GMF, MetaEdit+ [5, 6], Eugenia [8], and DSL Tools [7]).

Regarding graphical DSMLs, current approaches for developing them and generating modeling editors [8,9] are mostly focusing on reusing the infrastructures provid-

ed by programming IDEs. On the other hand, several approaches exist [11,12] for developing Web-based modeling editors using dedicated JavaScript (JS) frameworks. Compared to DSML-aware editors such as developed with EMF/xText, a Web-based editor may allow a much richer graphical representation of the DSML. Another advantage is that a Web-based modeling editor is very lightweight and simple to access. In cases where is not possible or not desired to use a modeling framework such Eclipse or MPS [10], you might still be able to integrate a Web-based modeling editor. However, these frameworks do not exploit the high automation potential from DSML approaches to generate modeling editors from language specifications. Thus, the development of Web-based modeling editors requires still major programming efforts.

In this paper, we combine the best of both worlds by reusing the language specifications for generating Web-based editors. In particular, we show how to combine two concrete approaches, namely EuGENia [8] from DSML engineering and JointJS [11] as a protagonist from JS frameworks and demonstrate the automation potential of establishing Web-based modeling editors. Finally, we discuss the results of using our approach for two existing DSMLs.

The outline of the paper is as follows. Section 2 introduces and compares the two approaches we are connecting in our proposal (i.e., Eugenia and JointJS). Section 3 presents our approach based on code generation and two concrete cases realized by our approach. Finally Section 4 presents the related work, before Section 5 concludes and outlines future work.

2. Background

In this section, we explain the basics of graphical modeling languages as well as the two worlds we connect with our approach. In particular, we present EuGENia as a concrete approach to specify DSMLs and automatically generate graphical modeling editors and JointJS for implementing graphical Web-based modeling editors.

2.1 Anatomy of Graphical Modeling Languages

A graphical concrete syntax (GCS) [2] has to define the following elements: *(i) graphical symbols*, e.g., lines, areas, complete figures such as SVG graphics, *(ii) labels* for representing textual information, e.g., for visualizing the names of modeling elements; *(iii) compositional rules*, which define how these graphical symbols are nested and combined, e.g., a label visualizing the name of a model element is centered within a rectangle representing the model element; and *(iv) mapping* of the *graphical symbols* to the elements of the *abstract syntax* for stating which graphical symbol should be used for which modeling concept, e.g., a specific model element type is visualized by a rectangle.

Current graphical modeling editors use modeling canvases which allow the positioning of model elements in a two-dimensional raster. Each element has an assigned *x*,*y* coordinate which normally stands for the upper-left corner of the graphical symbol. The model elements are mostly arranged as a graph which is contained in the modeling canvas. This graph is called diagram and represents a graphical view on the model. Please note that not all model information has to be actually shown in the modeling canvas. Several property values may be shown and may be editable in an

additional property view. This, on the one hand, allows accessing and editing every property of a model element, while, on the other hand, avoids overloading the diagram with too much information.

2.2 Eugenia

We selected EuGENia for demonstrating the bride between DSML engineering approaches and JS-based modeling editors, because it allows to introduce a GCS on an appropriate level of abstraction and complements the Eclipse Modeling Framework (EMF) in this respect. In particular, EuGENia provides several annotations for specifying the GCS for a given Ecore-based metamodel which describes the abstract syntax of a modeling language, i.e., the concepts and their properties without describing the concrete notation for the users of the language. In the following, the main annotations[1] are first enumerated and subsequently applied for an application example.

Diagram: The root element of the abstract syntax representing the model, i.e., the element containing (directly or indirectly) all other elements, is a perfect match for representing the modeling canvas.

Node: Instances of metamodel classes are often visualized as nodes within the diagrams. Thus, EuGENia allows annotating classes with the *Node* annotation. This annotation has several features, such as selecting the attribute of the annotated class which should be used as the label for the node, layout information such as border styles, colors, and either an external figure (e.g., provided as a SVG graphic) or a predefined figure by EuGENia (e.g., rectangle or ellipse) may be used to render the node.

Link: This annotation is applicable to classes as well as to non-containment references that should appear in the diagram as edges. This annotation provides attributes for setting the style of the link, e.g., if it is dashed, and the decoration of the link end, e.g., if the link end should be visualized as an arrow.

Compartment: Containment references may be marked with this annotation. It defines that the containment reference will create a compartment where model elements that conform to the type of the reference can be placed within.

Label: Attributes may be annotated with this annotation which implies that these attributes are shown in the diagram for nodes or links.

Figure 1 exemplifies the usage of EuGENia for a simple *hypertext modeling language* (HML). In the upper part there is the definition of HML by stating the three modeling concepts, i.e., the *hypertext* model is composed of *pages* and *links*. Furthermore, with annotations shown in comments notation, the concrete syntax of the modeling concepts is described. The hypertext models are represented by the *diagram* which is used to contain pages and links of the hypertext system. Furthermore, pages are shown as *rectangles* and links are shown as *arrows* pointing from the source page to the target page. The bottom part of the figure shows an example model using the concrete syntax of HML.

[1] More information on EuGENia annotations is provided at: http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial

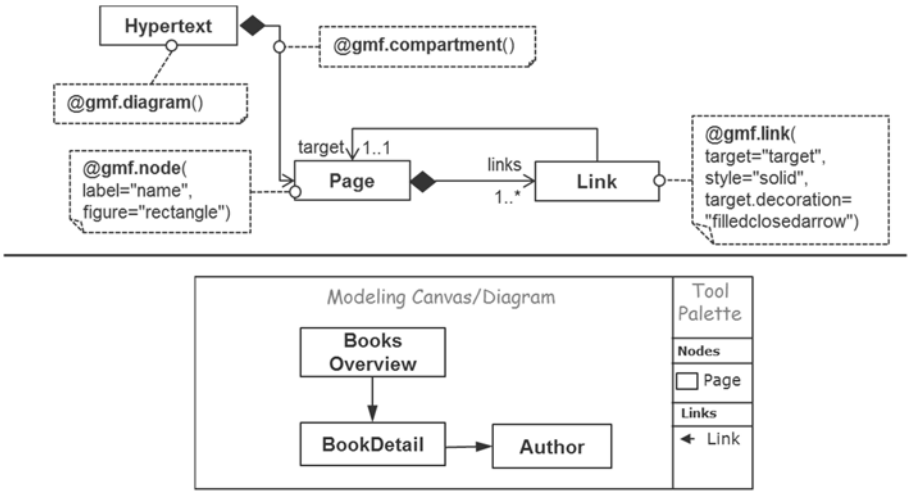

Figure 1: GCS definition with EuGENia by-example.

2.3. JointJS - JavaScript diagramming library

JointJS is an open source library for building interactive diagram-based modeling editors that run in Web browsers. JointJS comes with a commercial extension called Rappid which provides out-of-the-box UI components. Both are based on standard Web technologies such as SVG, HTML5, CSS3, and JavaScript, and follow a MVC architecture. This means, the model content is also separated from the model visualization as it is done by most DSML engineering approaches such as by EuGENia.

A modeling language which is supported by a JointJS-based editor is mostly defined with stencils. Within stencils, first, the modeling concepts have to be defined including the concrete syntax notation. For instance, consider the following code listing excerpt for defining the Hypertext modeling concept.

```
joint.shapes.Hypertext = joint.shapes.Hypertext.extend({
       markup: '<g class="rotatable">
                  <g class="scalable"><rect/></g>
                  <image/><text/><line/>
                </g>',
       defaults: joint.util.deepSupplement({
              type: 'Hypertext',
              paperWidth: pWFolder,
              paperHeight: pHFolder,
              position:{x: 0, y:0},
              …
       }… });
```

After having defined the concepts and their notational appearance, the composition rules such as before done with containment structures within the metamodel, have to be added. For instance, the following code listing specifies that *Hypertext* elements may contain *Page* elements.

```
var folder = new joint.shapes.Hypertext();
folder.prop({ inherit: { container: true,
                         canContain: [ joint.shapes.Pages ] }});
```

Finally, the tool palette has to be defined, i.e., which element types should be available to be instantiated by drag-and-drop. For instance, we want to be able to instantiate pages and links from the palette for our given hypertext modeling language.

```
modeler.stencil = [page, link];
```

3. Transforming EuGENia Models to JointJS

In this section, we describe our approach at a glance and outline the results of two experiments that we did, in particular, how to generate JointJS-based editors from existing EuGENia models of structural and behavioral modeling languages.

3.1 Overview

Our approach how to bridge current DSML engineering approaches and Web-based modeling editor programming approaches is outlined in Figure 2. As it is currently possible to generate from EuGENia models, Java code which runs on top of the Graphical Modeling Framework (GMF) in Eclipse, we developed a Model-to-Text (M2T) transformation to generate stencils for the JointJS platform which provide the definitions which we before discussed in Section 2. By this, we can follow a systematic language engineering approach based on metamodeling and at the same time, exploit rich Web-based modeling platforms without having to re-invent the wheel. The M2T transformation is implemented with Acceleo[2] which reads in the annotated Ecore models representing the abstract syntax as well as the graphical concrete syntax for the DSML contained in the annotations.

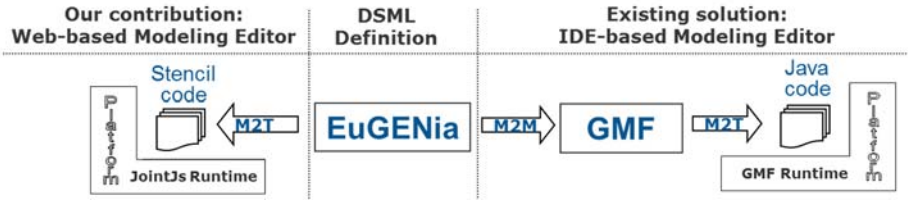

Figure 2: Extending EuGENia with an additional M2T transformation to generate Stencils for JointJS.

As Figure 2 outlines, our contribution is orthogonal to the existing support that comes with EuGENia. This means, we can now have a modeling editor inside the programming IDEs based on the GMF runtime, but at the same time, we can generate a Web-based modeling editor which runs on top of the JointJS runtime from the same DSML definition. However, an important requirement was to have a similar modeling experience in the Web-based modeling editor as in the IDE-based one. The next subsection describes how we approached this issue.

3.2 Development Methodology for the M2T Transformation

As development methodology for the M2T transformation we followed a reference system based approach. We investigated several existing EuGENia-based DSML definitions and how they are realized on the GMF platform. Based on this study, we re-implemented these projects directly with JointJS and aimed for having a similar

[2] https://www.eclipse.org/acceleo

modeling canvas, tool palette and graphical appearance of the modeling concepts. In particular, we used for this structural modeling languages such as the filesystem DSML[3] as well as behavioral modeling languages such as Petri nets[4].

Based on these projects, we developed the M2T transformation as an Acceleo template. The goal was to produce the before manually written JS code. Of course, this was an iterative process which required code adaptations in the different manually created projects to have one common template to generate the given code structures. To ensure the correctness and compatibility of the output files of the code generation process, a dedicated testing phase was required. Testing was an important part of every phase of development. Not only was the output JS file compared to the one manually written and included in the HTML5 project to see how it behaves, there was also an extensive testing phase at the end of the development cycle of the M2T transformation.

3.3 The Resulting Web-based Modeling Editors

For our investigated examples we could achieve promising results. Most parts of the DSML definition could be translated to JointJS in a similar way as it they are supported by GMF. The mentioned annotations in Section 2.2 are translatable to JointJS and the resulting modeling experience in the Web-based modeling editors is comparable to GMF. However, in our current implementation status of the M2T transformation, we do not support all EuGENia annotations and their properties. Thus, as future work, we have to further investigate if there are definitions which cannot be supported by JointJS, e.g., more complex label computations for elements which are supported by EuGENia. Thus, we see our current work as a baseline to further compare modeling editor support in IDEs and Web browsers and to learn if there are fundamental differences or not between the current approaches which emerged in different development branches in different communities.

Concerning the development efforts, we compared the Lines of Code (LoC) of the EuGENia solutions as there is also a textual concrete syntax to define such models based on Emfatic[5] and the JointJS stencil solutions. For the given examples, the LoC for the EuGENia solutions are between 30 and 40. However, for the JointJS stencil solutions 250 to 280 LoC are needed to realize the languages. This shows that there is a potential effort reduction in defining the modeling languages directly on the EuGENia level.

One important difference is of course how the models are stored in EuGENia/EMF and JointJS. In EMF the standard storage format is based on the XML Metadata Interchange (XMI) format. In JointJS models are stored as JSON files. In order to have model exchange capabilities between the Web-based modeling editors and the IDE-based modeling editors, a dedicated transformation has to be developed to convert models represented in XMI into models represented in JSON, and vice versa.

---

[3] http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial

[4] https://profesores.virtual.uniandes.edu.co/~isis4712/dokuwiki/doku.php?id=tut_eugenia

[5] https://www.eclipse.org/emfatic

4. Related Work

The Web may be considered a natural modeling platform given its straightforward support for collaboration, portability, and its very powerful, interactive and advanced UIs. As pointed out in existing literature [14], the Web became a platform where model engineering may be fully exploited in practice. The facts have shown this. A proliferation of model editor libraries based on Web technologies has been happening in the last years.

These libraries facilitate the development of Web-based modeling editors using dedicated JS frameworks. A Web-based editor is very lightweight and simple, and allows a much richer graphical representation of the DSL, as some products show [11, 13]. However, these frameworks do not exploit the high automation potential from DSML approaches to generate modeling editors from language specifications. Instead of this, they usually offer a low-level API that allows developers to code the editor behavior, model constraints, define model elements, etc. Thus, the development of Web-based modeling editors requires still significant programming efforts.

Some research works have early emerged for conducting the collaboration and groupware concern in modeling editors based on Web technologies [12,15]. Nevertheless, this issue is not related with the goal of this paper. Note that the collaboration concern in this kind of applications depends on the features of the underlying JS library being used rather than in how JS code for that library is generated automatically given a specific metamodel. As we mentioned before, the main goal of this paper is the generation of Web-based model editors starting from the design of a metamodel, making it compatible with our HTML5 modeling tool to create proper models. This whole concept as a unit seems to be unaddressed by the current state of technology, in spite that there exist some works proposing the generation of graphical editors [8, 9].

However, some existing work is more specifically related to defining the modeling language in itself, such as Clooca [16]. Clooca allows developers to define DSMLs, and the corresponding software that generates the code for a particular model instance. Although Clooca proposes a tool developed with Web technologies, this is not oriented to create the Web-based model editors for the DSML specified.

5. Conclusions

The Web is currently a platform where users perform daily tasks, and with this in mind, modeling in the context of the Web browser may be useful in several ways. Some well-known Web modeling editors have been arising for letting users define different software artifacts, such as the case of Node-RED [17] for modeling IoT application flows, which support this claim.

Although it is true that there was a proliferation of libraries for developing Web-based editors, they require advanced programming skills, which could be error prone at the moment of defining modeling constraints. The importance of having reliable Web-based modeling editors depends strongly on the possibility of specifying particular behavior for these editors regarding how model elements will be managed, their relationships, constraints, properties, etc. Our approach makes the creation of a Web-based modeling editor simpler and more guided, even without requiring programming skills on Web technologies. It reduces the possibility of introducing errors when programming the editor but also improves the editor maintenance when the underlying

metamodel evolves. The next step in our research is to perform an evaluation of the Web-based editor generation process for larger languages such as UML, SysML, or BPMN and to evaluate how to generate animation code from the operational semantic definitions of the modeling languages.

Acknowledgements

We thank Richard Sevela for his work on the M2T transformation implementation. This work has been funded by the Austrian Federal Ministry of Science, Research and Economy and by the National Foundation for Research, Technology and Development and the project TIN2016-78103-C2-2-R of the Spanish Ministry of Economy, Industry and Competitiveness.

References

1. Steven Kelly, Juha-Pekka Tolvanen: Domain-Specific Modeling - Enabling Full Code Generation. Wiley, 2008.
2. Marco Brambilla, Jordi Cabot, Manuel Wimmer: Model-Driven Software Engineering in Practice. Morgan & Claypool, 2012.
3. http://www.emftext.org
4. http://www.eclipse.org/Xtext
5. http://www.metacase.com/mep
6. Nick Baetens: Comparing graphical DSL editors: AToM3, GMF, MetaEdit+. Technical report, University of Antwerp, 2011.
7. Steve Cook, Gareth Jones, Stuart Kent, and Alan Wills: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley, 2007.
8. Dimitrios S. Kolovos, Antonio García-Domínguez, Louis M. Rose, Richard F. Paige: Eugenia: towards disciplined and automated development of GMF-based graphical model editors. Software & Systems Modeling, 16(1), pp 229–255, 2017.
9. Suzy Temate, Laurent Broto, Alain Tchana, Daniel Hagimont: A high level approach for generating model's graphical editors. In: Proceedings of ITNG, 2011.
10. https://www.jetbrains.com/mps
11. https://www.jointjs.com
12. Louis M. Rose, Dimitrios S. Kolovos, and Richard F. Paige: EuGENia live: a flexible graphical modelling tool. In: Proceedings of the Extreme Modeling Workshop (XM) @ MODELS, 2012.
13. http://concrete-editor.org
14. Manuel Wimmer, Andrea Schauerhuber, Michael Strommer, Jürgen Flandorfer, Gerti Kappel: How Web 2.0 can Leverage Model Engineering in Practice. In: Proceedings of DSML Workshop, 2008.
15. Christian Thum, Michael Schwind, Martin Schader: SLIM—A lightweight environment for synchronous collaborative modeling. In: Proceedings of MODELS, 2009.
16. Shuhei Hiya, Kenji Hisazumi, Akira Fukuda, Tsuneo Nakanishi: Clooca: Web based tool for Domain Specific Modeling. In: Proceedings of Demos/Posters/StudentResearch @ MoDELS, 2013.
17. Node-Red, http://nodered.org