DSLFORGE: Textual Modeling on the Web

Amine Lajmi[1], Jabier Martinez[2,3], and Tewfik Ziadi[3]

[1] Software Architect, Paris, France, amine.lajmi@dslforge.com
[2] SnT, University of Luxembourg, Luxembourg, jabier.martinez@uni.lu
[3] LIP6, Université Pierre et Marie Curie, Paris, France, tewfik.ziadi@lip6.fr

Abstract. The use of Model-Driven Engineering in software development is increasingly growing in industrial applications as the technologies are becoming more mature. In particular, domain-specific languages bring to end-users simplicity of use and productivity by means of various artifacts generators. However, end-users still need to cope with heavy modeling infrastructures and complex deployment procedures, before being able to work on models. In this paper, we propose a centralized lightweight approach for performing textual modeling through web browsers. DSLFORGE is a generator of online text editors. Given a language grammar, the tool allows to generate lightweight web editors, supporting syntax highlighting, syntax validation, scoping, and code completion. DSLFORGE allows also automatic integration of existing code generators into the generated web editor providing a complete online modeling user experience.

Demo: http://youtu.be/KN6cneWhhKY

Keywords: textual modeling, online editor, model-driven engineering, domain-specific languages

1 Introduction

Domain-Specific Languages (DSL) [10] let end-users feel the advantages of using domain abstractions instead of general-purpose language constructs. Therefore, Model-Driven Engineering (MDE) is gaining more success in industrial applications as the available methods and tools are becoming more mature. Modeling technologies have made big steps towards better integration and simplicity of use and, as a consequence, domain-specific standards have met great success within multiple communities (e.g. Modelica [15], AUTOSAR [2], and SysML [17]).

However, end-users still need to cope with heavy development infrastructures and complex deployment procedures when it comes to using modeling tools. This is because most of the existing tools are based on general purpose Integrated Development Environments (IDEs) such as Eclipse or MPS, or standalone proprietary applications as MetaEdit+ [11]. Indeed, in most of the tools, whether bundled into standalone Rich Client Platforms (RCP), or packaged separately as individual features, both *Tooling* and *Runtime* need to run on top of heavy infrastructures. The deployment of modeling tools is still a tedious task for non-developers and goes sometimes against the adoption of DSLs in enterprises.

Indeed, there is a lack of practical solutions, lightweight and easy-to-deploy. Moreover, modeling resources, as any kind of software resources today, experience an increasing demand to be accessed using different devices such as tablets and smartphones and from any place in the world at any time.

In this paper, we present the DSLFORGE tool, as support of a lightweight centralized approach that allows end-users to edit and process textual models through web browsers. The paper is organized as follows: Section 2 describes current web-based approaches and Section 3 presents the DSLFORGE tool, its methodology of use, two functional examples and its internal architecture.

2 Modeling in the web

There are still technological challenges to achieve modeling on the web. Nevertheless, some initiatives targeting *graphical* languages are worth to notice. GenMyModel [4] allows end-users to create UML diagrams and launch artifacts generation. AToMPM [16] is an online graphical modeling environment, which uses a subset of UML for the definition of modeling languages and renders model elements in SVG. GEFGWT [6] allows to build graphical editors based on GEF and could be the basis for porting GMF on the web. Also, the Eclipse-based Remote Application Platform (RAP) [7] makes easier to port Rich Client Applications (RCP) [9] to the web, since most of the SWT [13] libraries are transparently handled by the framework. RAP-based EMF editors have been provided in tools like EMF on Rails [8] or MUVITORKIT [12]. They are based on the tree viewer and a properties page offering limited user experience when a textual representation is more appropriate.

Few tools provide flexible means to define *textual* languages and they come with in-house formalisms. For example, in Concrete Editor [3], models are represented by DOM nodes with specific CSS classes. An Xtext-based online editor has been also prototyped [5]. This prototype, which is based on Orion [14], was proposed as an initial exploration on the feasibility of online textual editors. It allows editing EMF resources online but the main part of resource processing is done on the server. Within the server, each service (e.g. syntax highlighting, content assist, hover, etc.) is held by a dedicated servlet asynchronously, and the entire document is sent to the server and back again to the client at each user interaction, leading to serious performance issues. Moreover, neither methodology nor tool has been provided to show how one could bring a DSL to the web. Processing EMF-based DSL resources intensively on the client side is not reasonable also, as one has to port the entire Eclipse workbench to JavaScript. The frontier of what should be done on the client side against what should be done on the server side is an open question. DSLFORGE resource processing is distributed between the client and the server. We take advantage of two open-source technologies which are RAP and ACE [1]. The integration with RAP allows the easy integration with standard widgets such as file system navigators, file uploaders, forms, etc.

3 DSLFORGE

DSLFORGE is a framework for the generation of web textual DSL editors. The generated editors are packaged into workbench web applications which let users create, edit, and launch transformations from models online, making it possible to work simultaneously with partners or colleagues on the same resources. These online editors are also easily customizable and extensible. DSLFORGE is proposed to two main categories of users: (i) *DSL developers*, and (ii) *DSL users*. The former is given a technology to build and publish online editors on the web. The latter uses the DSL editor through lightweight applications (enterprise intranet, mobile devices, tablets, etc.).

3.1 Methodology

The framework is packaged into two features: *Tooling* and *Runtime*. DSL developers use the Tooling which contains all the needed components to generate and deploy the editor on application servers. The steps below are followed iteratively by DSL developers to get an operational online editor:

1. Design/enhance the DSL
2. If applicable, design/enhance transformations to generate some kind of artifacts from DSL instances
3. Automatically generate from the grammar the RCP and the RAP editor
4. If needed, enhance the web editor generated code. Third party plugins could provide extra functionalities.
5. Automatically package and deploy the editor.

DSL users use the deployed editor to edit and process models online, namely they can: create, edit and save models, trigger code generation from a model, and execute other actions or functionalities if the web editor was extended by third party plugins.

3.2 Examples

Generating a State Machine web editor: As a first example, we use the Martin Fowler's state machine example provided by Xtext. The example comes with a grammar allowing textual specification of state machines. Our objective is to provide a web editor that allows modeling state machines and generating java test classes from these state machines on the server using the browser. An extra functionality should be integrated for compiling and executing the test classes through the browser too.

Using DSLFORGE, as shown in Figure 1, we select the Xtext grammar to generate its online editor. At this moment we are able to select the existing transformations that will be available online for end-users. In this case, we select the existing state-machine to java transformation. The online editor is automatically packaged into a web application, together with a workspace navigator.

To execute the web application we launch the browser. Figure 2 shows how the editor handles syntax validation, content assist and how the web application is enhanced with a code generation action contributed to the Tools menu.

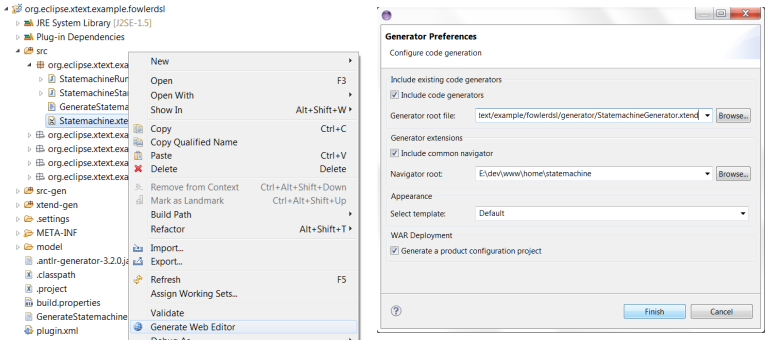

Fig. 1. DSL developers using DSLFORGE to automatically create the ready to use textual web editor

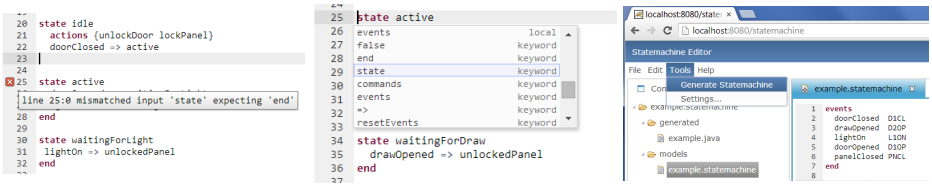

Fig. 2. DSL users using online syntax validation, content assist and model transformation launch for code generation

Semantic highlighting, scoping, history management (undo/redo), key bindings and folding are also supported by default. In addition, a custom action is integrated to the web application for compiling and executing Java classes on the server, to show an example of the online editor extensibility.

A web editor for the specification of conference websites: As a second example, we use DSLFORGE in the context of a conference DSL. This DSL allows end-users, who are not necessarily HTML/JavaScript experts, to generate and update on the server the conference or workshop website. This way, they can update the conference website wherever they are and without knowing the website technical details.

3.3 Architecture

The *Tooling* feature is shipped with Xtext, EMF, and ANTLR development features, and contains the editor generator and other contributed plugins which may be integrated with the editor into online workbenches (e.g. workspace navigator, authentication, file uploader). The *Runtime* contains the standard RAP target platform together with EMF target components, Xtext runtime, and additional plugins used during the execution of the editor. The editor can be integrated with any third party RAP-compatible plugin.

DSLFORGE follows the RAP standard architecture and lifecycle. Indeed, each generated editor widget has its counterpart in JavaScript. Resource processing is dispatched between the server and the client. The communication between the client and the server uses the JSON format. Multi-threading is used on the client side to handle the computationally expensive resource processing. Indeed, dedicated workers parse the user input and feed-back the UI with annotations. Shared workers are used to manage the global index which maintains available the cross references to all editors opened within the same user session. On the server side, parsing and validation is triggered when saving the resource. Asynchronous server pushes are used to notify the client about workspace change events and server-side resource validation. The generated web applications have been successfully deployed on two application servers (Tomcat and Jetty). Future work will include securing the communication between the client and the server and enhancing the management of resource access privileges.

References

1. Ace: Ace, Cloud 9 IDE (2014), `http://ace.c9.io`
2. AUTOSAR: AUTomotive Open System Architecture (2014), `http://autosar.org`
3. ConcreteEditor: Concrete Editor (2014), `http://concrete-editor.org`
4. Dirix, M., Muller, A., Aranega, V.: GenMyModel: An Online UML Case Tool. In: ECOOP (2013)
5. Eysholdt, M., Behrens, H.: Xtext: implement your language faster than the quick and dirty way. In: SPLASH/OOPSLA Companion. pp. 307–309 (2010)
6. GEFGWT: GEF in the web browser (2014), `http://www.gefgwt.org`
7. Lange, F.: Eclipse Rich Ajax Platform: Bringing Rich Client to the Web. Apress, Berkely, CA, USA, 1 edn. (2008)
8. López-Landa, R., Noguez, J., Guerra, E., de Lara, J.: EMF on Rails. In: Proc. ICSOFT. pp. 273–278 (2012)
9. McAffer, J., Lemieux, J.M.: Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications. Addison-Wesley Professional (2005)
10. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-specific Languages. ACM Comput. Surv. 37(4), 316–344 (Dec 2005)
11. MetaCase: MetaEdit+ (2014), `http://www.metacase.com/`
12. Modica, T., Biermann, E., Ermel, C.: An Eclipse Framework for Rapid Development of Rich-featured GEF Editors based on EMF Models. In: GI Jahrestagung (2009)
13. Northover, S., Wilson, M.: SWT: The Standard Widget Toolkit, Volume 1. Addison-Wesley Professional, first edn. (2004)
14. Orion: Orion Project (2014), `http://www.eclipse.org/orion`
15. Saldamli, L., Fritzson, P., Aronsson, P., Bunus, P., Engelson, V., Johansson, H., Karström, A.: The Open Source Modelica Project. In: Proc. Modelica Conf. Modelica Association (2002)
16. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S.V., Ergin, H.: AToMPM: A Web-based Modeling Environment. In: Demos/Posters/StudentResearch@MoDELS. pp. 21–25 (2013)
17. SysML: Systems Modeling Language (2014), `http://www.uml-sysml.org`