Stefanie Betz, Ulrich Reimer (Hrsg.): Modellierung 2016 Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 123

Community Application Editor: Collaborative Near Real-Time Modeling and Composition of Microservice-based Web Applications

Peter de Lange[1], Petru Nicolaescu[1], Michael Derntl[2], Matthias Jarke[1,3], Ralf Klamma[1]

Abstract: Research shows a gap in terms of requirements elicitation between developers and end-users. Due to the low technical expertise of some members of online communities, they often cannot collaborate efficiently with developers and cannot provide continuous feedback during application development processes. However, collaborative modeling processes can play an important role in education, enforcing best-practices, support rapid prototyping and lower the communication and collaboration barriers between developers and end-users. This paper presents a tool for collaborative modeling of Web applications in near real-time and their automatic generation. Early evaluation with end-users in simulated community settings show promising results for the interplay of modeling and collaboration in requirements gathering and Web application design and development. We claim that near real-time modeling on the Web has the potential to bring together stakeholders during design and development and offers a new perspective on model-based Web Engineering.

Keywords: Near Real-time Modeling, Conceptual Models, Model Driven Web Engineering, Microservices, Web Widgets

1 Introduction

In many disciplines modeling is a key activity in defining abstract representations of the target domain. Models obtain meaning through creative social processes, which are driven by different perspectives of involved stakeholders [DFK02]. Modeling of Web applications [GCP01], model-based code generation and methods for Web Engineering with respect to Web applications [AK03] and application flow [KKK07] have been thoroughly analyzed during the past decades. However, the impact of near real-time (NRT) collaborative modeling onto requirements elicitation, information systems design and implementation with the purpose to bring together developers and end-users has not been studied so far in the context of the NRT Web 2.0. Therefore, this work presents the Community Application Editor (CAE), a tool for the development of Web applications based on a NRT collaborative modeling approach. Using a given Web application metamodel, end-users and developers involved in Communities of Practice [We98] can model together a desired application and easily generate modular code for both server-side (using microservices) and client-side (using Web widgets). The entire software used and generated is open

[1] RWTH Aachen University, Chair for Information Systems and Databases, Ahornstr. 55, 52074 Aachen, Germany, {lastname}@dbis.rwth-aachen.de
[2] Eberhard Karls Universität Tübingen, eScience-Center, Wilhelmstr. 32, 72074 Tübingen, Germany, michael.derntl@uni-tuebingen.de
[3] Fraunhofer FIT, Schloss Birlinghoven, Konrad-Adenauer-Straße, 53754 Sankt Augustin, Germany

source and targets online communities that can take advantage of NRT collaborative technologies which allow free, concurrent shared editing. Section 2 gives a short summary of the approach used for collaborative conceptual modeling on the Web. Section 3 presents the collaborative modeling process and the tool realization.

2 Near Real-time Modeling on the Web

So far, the conceptual modeling literature has emphasized little on the opportunities and challenges of NRT collaboration for conceptual modeling and model-driven application development. For the design phases of information systems, models result from the collaboration between different stakeholders. Although there are many metamodeling and modeling tools available, few of those support synchronous Web-based collaboration and are published under open source licenses using open libraries and protocols. Closing this gap, we have developed the metamodeling framework SyncMeta [De14, De15], which enables the collaborative creation of metamodels and the generation of collaborative model editors from the defined metamodels. It supports NRT collaborative (meta-) modeling in the Web browser and is implemented as open source[4] using widely known Web protocols. SyncMeta was successfully used to create (meta-) models of various conceptual modeling languages (e.g. entity relationship diagrams, i*, IMS LD). The framework was developed based on ROLE SDK, a widget-based platform offering user management and communication and collaboration between multiple users directly in the browser. It consists of multiple widgets [De14], the modeling logic and the collaborative (meta-) model editing being implemented entirely as decentralized client-side features. Web Widgets are simple interface elements that offer a single, clear-cut functionality and can be embedded in various Web applications. As a further step, this work demonstrates the power of NRT collaborative modeling for achieving modern Web applications through the active participation of online communities members with various technical background. Using a well-defined metamodel for Web applications and NRT collaborative modeling as communication and negotiation means (e.g. for an implicit requirements elicitation and the architecture specification of the designed software), developers and end-users can benefit from a structured approach to (re)design and implement Web applications. Together with model-based code generation features, benefits of such an approach are rapid prototyping scenarios, a well-designed software architecture landscape throughout the communities software stack and new opportunities for technology adoption and education means.

3 The Community Application Editor

CAE offers the means to co-design and implement Web applications with model driven Web engineering techniques. Resulting applications are based on a componentized and scalable architecture consisting of Web widgets (client-side) and microservices (server-side). Mirroring the widget components, microservices[5] are scalable server components with a clear-cut functionality which can run independently on multiple infrastructures.

[4] https://github.com/rwth-acis/syncmeta

[5] http://martinfowler.com/articles/microservices.html

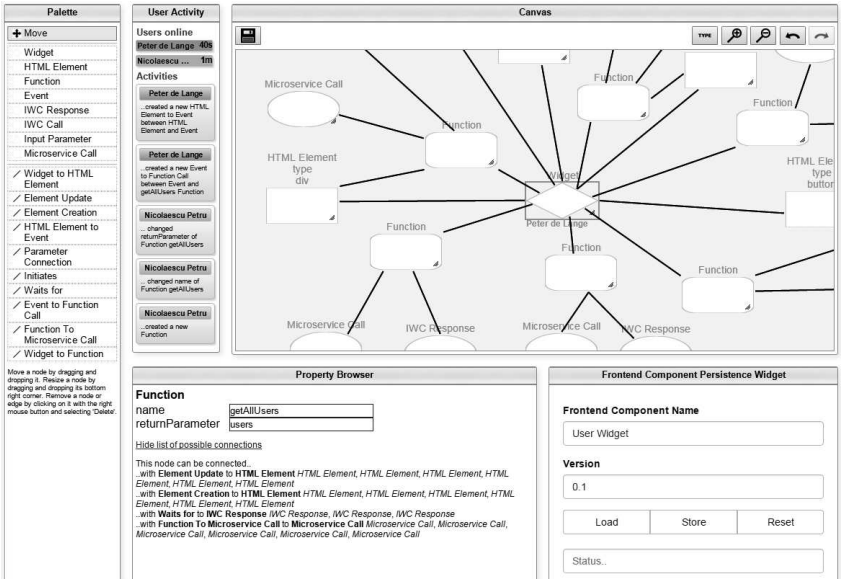

Fig. 1: Frontend Component Modeling Space

Since the target groups for our tool are online communities, users have various roles, such as developer, software architect or non-technical community member. Any community member can join a collaboration session to design and generate a Web application. Using SyncMeta, they can model in NRT an application model. The modeling process is based on a predefined metamodel, reflecting the complete application, split up into three different "views" to allow for a componentized architecture that can be developed separately or in parallel. A view is dedicated to the client-side, where the widgets and the user-interface application flow (e.g. functions, events, elements) are defined. The second view represents the server-side part of the application model, where the microservices are defined, together with the database and the service APIs. The third view, partially overlapping with the previous two is used for the communication between the various components. In this view, the application can be mashed up together using the already specified components. Once a widget or microservice is collaboratively specified, it can be persisted. While being saved, the respective component model is also transformed to code and pushed into an open source code repository, using a template structure together with the model information to generate the source code of the component reflected by the model. Optional (manual) code refinements can be done afterwards, before the complete application is mashed up from the generated components. The mash-up model view contains the dependencies between microservice and widgets and the inter-microservice and inter-widget communication. Upon agreement on the content of the three views composing the application model (i.e. the requirements are fulfilled), the complete application can be persisted. A repository gets created for the application code, which can be later used for deployment.

Figure 1 shows the CAE frontend component modeling space, which realizes the view on a single component of the client-side. It consists of five widgets. Modelers use the *Palette*

widget to select the different objects defined by the frontend component metamodel. These objects are then placed in the *Canvas* widget and connected according to the metamodel definitions. The canvas also gives information about currently selected objects by other users, raising the awareness of the modeling community. Another awareness related feature is realized by the *User Activity* widget, which shows recent changes in the model of each user, for example adding of edges or moving of objects. The *Property Browser* widget is used to display additional information about an object selected in the canvas. It shows both the attributes of an object, which can be modified by each user, as well as a list of objects the selected object can be connected to. All changes done in the modeling canvas as well as the property browser are propagated in NRT to all users. Finally, the *Frontend Component Persistent Widget* can be used to both store the model in a database and generate code from it, which is then automatically pushed to a code repository. It can also be used to load existing models for further editing.

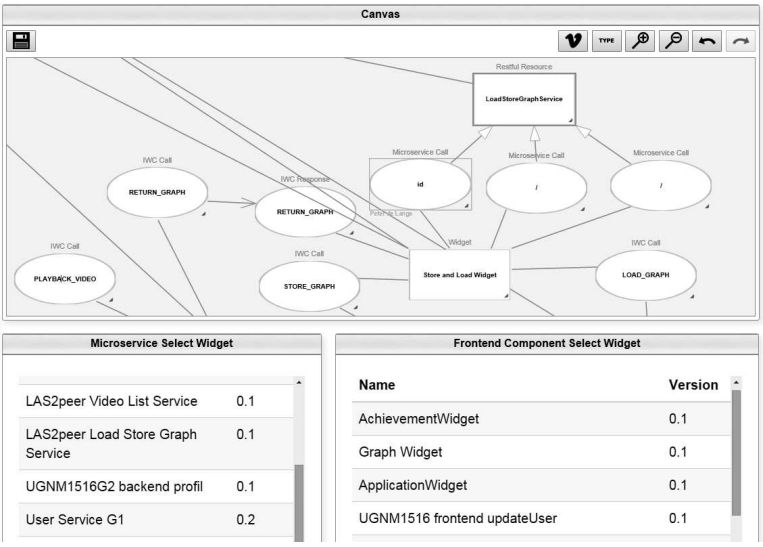

Fig. 2: Mash-Up Modeling Space

Figure 2 shows a screenshot of third view of the CAE's metamodel, the mash-up modeling space. The difference from the frontend component modeling space is the absence of the palette widget, which is replaced by a list of available frontend and microservices. The modelers can select those and place them on the canvas. After saving the application model, a so called communication view is calculated by the backend, which shows the dependencies and interrelationships of the single components. The last modeling space, which realizes the server-side (microservice) view of an application, is similar to the frontend component space, with the difference in the available objects and dependencies for modeling, corresponding to the underlying metamodel part.

We successfully used the CAE to redesign an existing collaborative Web application used for graph-based storytelling[6]. This follows a simple use-case, where members of a cultural

[6]https://github.com/rwth-acis/CAE-Example-Application

heritage community want to develop a simple system for linking videos in a graph-based manner and visualize individual videos. Helped by developers, they model in NRT a video list and a graph microservice and the corresponding widgets: a graph-rendering canvas with video element nodes, a list for graphs and videos and a video player widget. Upon saving the models, the code is automatically generated using the templates. Finally, the developers will finalize the functionality and improve the user interface presentation style. Based on this, the CAE demonstration showcases the generation of a complete Web application, similar to the already existing YouTube demonstration video[7]. We have performed a user study in a simulated community setting, using groups of two or three users consisting of at least one developer and one user with no technical background. After modeling a trivial social (blog-like) application, the feedback was gathered using an online survey and user interviews. Our user evaluation showed promising results which will be presented in future publications. The sources of the CAE can be found on GitHub[8] [9] [10] [11] [12]. The generated applications can be found in the CAE's GitHub organization[13].

Acknowledgments

This research was funded by the EU Commission in the project "Layers" (FP7-318209).

References

[AK03] Atkinson, C.; Kühne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software, 20(5):36–41, 2003.

[De14] Derntl, M.; Erdtmann, S.; Nicolaescu, P.; Klamma, R.; Jarke, M.: Echtzeitmetamodellierung im Web-Browser. In (H.G. Fill; D. Karagiannis; U. Reimer, eds): Modellierung 2014, pp. 65–80. Gesellschaft fur Informatik e.V., 2014.

[De15] Derntl, M.; Nicolaescu, P.; Erdtmann, S.; Klamma, R.; Jarke, M.: Near Real-Time Collaborative Conceptual Modeling on the Web. In: Conceptual Modeling. volume 9381. Springer International Publishing, pp. 344–357, 2015.

[DFK02] Dittrich, Y.; Floyd, C.; Klischewski, R.: Social Thinking–Software Practice. MIT Press, Cambridge and MA, 2002.

[GCP01] Gomez, J.; Cachero, C.; Pastor, O.: Conceptual Modeling of Device-Independent Web Applications. IEEE Multimedia, 8(2):26–39, 2001.

[KKK07] Kraus, A.; Knapp, A.; Koch, N.: Model-Driven Generation of Web Applications in UWE. In: 3rd International Workshop on Model-Driven Web Engineering (MDWE), volume 261. CEUR-WS, 2007.

[We98] Wenger, E.: Communities of Practice: Learning, Meaning, and Identity. Learning in doing. Cambridge University Press, Cambridge, UK, 1998.

[7] https://www.youtube.com/watch?v=Vuyj2e32ePk
[8] https://github.com/rwth-acis/CAE-Code-Generation-Service
[9] https://github.com/rwth-acis/CAE-Model-Persistence-Service
[10] https://github.com/rwth-acis/CAE-Templates
[11] https://github.com/rwth-acis/CAE-Simple-Model-Representation
[12] https://github.com/rwth-acis/CAE-SyncMeta-Additions
[13] https://github.com/CAE-Community-Application-Editor