LiquidML: A Model Based Environment for Developing High Scalable Web Applications

Esteban Robles Luna[1], José Matías Rivero[1,2], and Matias Urbieta[1,2]

[1] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{esteban.robles,mrivero,matias.urbieta}@lifia.info.unlp.edu.ar
[2] Also at Conicet

Abstract. The scalability of modern Web applications has become a key aspect for any business in order to support thousands of concurrent users while reducing its computational costs. However, existing model driven web engineering approaches have been focus on building Web applications that satisfy functional requirements while disregarding "technological" aspects such as scalability and performance. As a consequence, the applications derived from these approaches may not scale well and need to be adapted. In this paper we present the LiquidML environment, which allows building Web applications using a model-based approach. In contrast with existing approaches, aspects that help to improve the scalability of a Web application are modeled as first class citizens and as a consequence the applications obtained scale better than its counterparts.

Keywords: Scalability, Model driven development, Web engineering.

1 Introduction

Scalability is the ability of a system to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth [2]. Scalability problems in the applications derived using Model-Driven Web Engineering (MDWE) tools may not appear as soon as they are deployed, but rather after they has been "living" in production for some time. Fixing scalability issues requires a detail diagnosis and consumes many human resources [4] and it has been shown to take 50-70% of the development time [1].

In the Web engineering research area [6], MDWE approaches [5,3,7] have become an attractive solution for building Web applications as they raise the level of abstraction and simplify the Web application development. However, according to [8], little attention has been put to non-functional requirements such as scalability issues as they have been considered a technological aspect that does not need to be modeled. In the same line, the Web applications derived from MDWE approaches pose a rigid/static architecture that cannot be easily changed thus limiting the size of the Web applications that can be built with them. In addition, diagnosing and fixing these problems in production systems (which are the ones that present overload symptoms) becomes cumbersome and impossible to be done in the models thus forcing teams to deal with the generated code losing the high level of abstraction provided by the design models.

S. Casteleyn, G. Rossi, and M. Winckler (Eds.): ICWE 2014, LNCS 8541, pp. 519–522, 2014.

In this paper we present *LiquidML*, an environment that helps building high scalable Web applications that can be manipulated at runtime. A LiquidML application can be either derived from MDWE models such as WebML, or it can be built straight inside the environment by manually creating models. The main difference with existing tools is that we do not impose a rigid architecture and those "technological" aspects are modeled; as a consequence we can keep track of their changes and alter the models at runtime in a safe way.

2 LiquidML Models

The LiquidML environment allows engineers to model a set of features that handle Web applications requests as first class citizens. A non-exhaustive list of features includes: logging and profiling, caching, service calls, parallel, asynchronous and sequence execution modes, message queuing and the deployment process. A LiquidML application is a composition of Flows and a Flow describes a sequence of steps that need to happen within a Web request (called *Message* in our approach). A Message has a payload (body) and a list of properties while each step in the Flow is visually identified by an icon and constitutes an *Element* of it. Communication between Elements happens by means of Message interchanges.

To exemplify the concepts, we present a Flow for the product's detail page of an E-commerce web application (Fig. 1). The Element with no incoming arrow represents the Message source listener that will receive incoming requests – in this case, it will receive HTTP request and will transform them in to Messages. The Element connected to the Message source named "Route path" is a ChoiceRouter, which behaves like a choice/switch statement and it will route the message to the "Get info" processor if the request comes to a URL starting with "/product/*". The "Get info" is another router that gets information in parallel from multiple sources. It obtains the product info from the DB: ("Get product info") and triggers the computation of the product's rank, which involves two database (DB) queries ("Get user reputation" and "Get product reviews") and a Processor that computes the rank from this information ("Compute product rank"). Finally, the information gets composed ("Compose data") and used for rendering a Web page in the "Render template" processor.

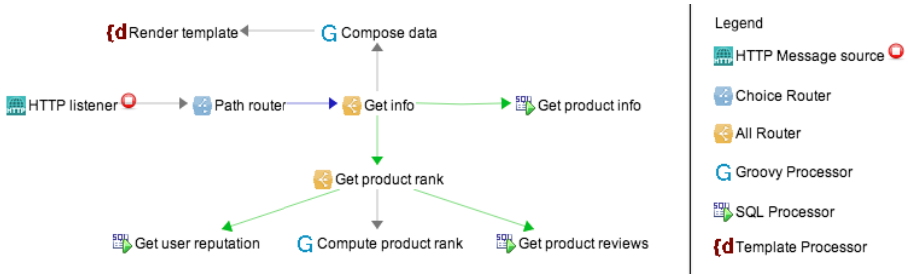

Fig. 1. Product details flow

As aforementioned, to improve the scalability of the Web application some elements suggested in [4] has been modeled as first class citizens. In Fig. 2 we show a modified version of the Product details flow of Fig 1, were some elements for caching

purposes have been added to improve the overall scalability of the Web application. In LiquidML, these changes can be applied at design time by adding the elements in the diagram or by means of runtime transformations that change model elements on-the-fly while the application is running.

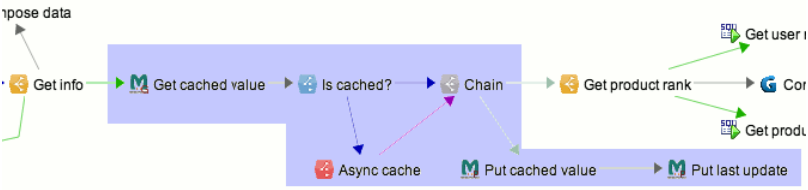

Fig. 2. Product details flow improved with caching elements

3 LiquidML Environment

The LiquidML environment is composed of 2 applications:

- LiquidML editor: it allows defining the applications, modelling Flows and to deploy them to the LiquidML servers. It also, allows developers to apply dynamic transformations to the deployed diagrams in order to deal with the scalability problems and watch for performance and logging information on-the-fly in any environments where the application is running (DEV, QA or even Production).
- LiquidML server: The server is responsible for holding the application definitions, the LiquidML interpreter and notifying the editor about how applications are running. In addition, it regularly checks if it has any pending deployments and if so, it fetches the Application and automatically deploys it.

LiquidML do not impose any rigid implementation architecture. For instance, a complete Web application can be split in 1 front-end app and 2 different service apps, which can be integrated by sending messages to their REST endpoints. In addition, each application, e.g. the front-end app, can be instantiated in multiple servers and combined by load balancers to distribute the load and improve the overall scalability of the application. As an example, in Fig. 3 we present the deployment view of a diagram, which allows us to use the diagnostic tools of LiquidML such as profiling and logging and it also supports performing runtime transformations oriented to improve the scalability of the Web application.

Both, the editor and the server have been built using open source technologies of the JEE stack. We have used Spring and Hibernate for basic service and ORM mapping, Spring MVC and Twitter bootstrap for UI and Jersey for the LiquidML API. As part of this development, we have built the CupDraw framework[1] for building Web diagram editors, which is publicly available. For the technical readers, we invite them to visit the LiquidML site http://www.liquidml.com and check the project's source code and the demonstration videos.

[1] `https://github.com/estebanroblesluna/cupDraw`

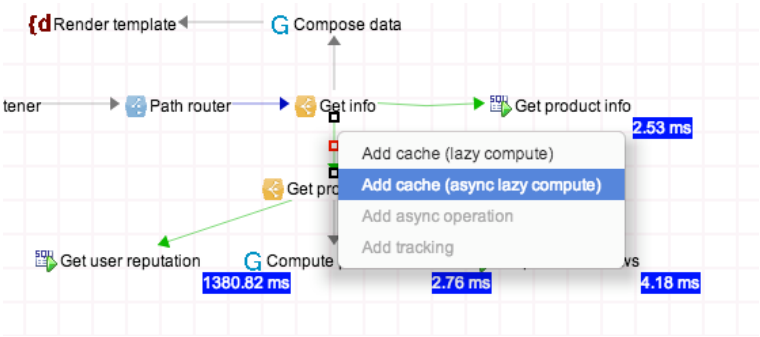

Fig. 3. Deployment view

4 Conclusions

In this demo paper, we have presented the LiquidML environment as a place where Web applications can be either derived from MDWE models or manually created from our "low level" LiquidML models. These models can be then interpreted and dynamically reconfigured at runtime. The environment is web based and it only requires a browser to be used.

References

1. Boehm, B.W.: Software engineering economics. Prentice-Hall, Englewood Cliffs (1981)
2. Bondi, A.: Characteristics of scalability and their impact on performance. In: Proceedings of the 2nd International Workshop on Software and Performance (WOSP 2000), pp. 195–203. ACM, New York (2000)
3. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): A Model-ing Language for Designing Web Sites. Computer Networks and ISDN Systems 33(1-6), 137–157 (2000)
4. Hull, S.: 20 Obstacles to Scalability. Queue 11(7), 20 (2013)
5. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-Based Web Engineering, An Approach Based On Standards. In: Web Engineering, Modelling and Implementing Web Applications, pp. 157–191. Springer, Heidelberg (2008)
6. Rossi, G., Pastor, O., Schwabe, D., Olsina, L.: Web Engineering: Modelling and Implementing Web Applications. Springer (2007)
7. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications using OOHDM. In: Web Engineering, Modelling and Implementing Web Applications, pp. 109–155. Springer, Heidelberg (2008)
8. Toffetti, G.: Web engineering for cloud computing (web engineering forecast: cloudy with a chance of opportunities). In: Proceedings of the 12th International Conference on Current Trends in Web Engineering (ICWE 2012), pp. 5–19. Springer, Heidelberg (2012)