

Fujaba goes Web 2.0

Nina Aschenbrenner, Jörn Dreyer, Ruben Jubeh, Albert Zündorf
University of Kassel, Software Engineering,
Department of Computer Science and Electrical Engineering,
Wilhelmshöher Allee 73,
34121 Kassel, Germany
[nina.aschenbrenner | jdr | ruben | zuendorf]@cs.uni-kassel.de
<http://www.se.eecs.uni-kassel.de/se/>

ABSTRACT

The latest research activities of the Fujaba group of Kassel University led to challenges in development of new web technologies enabling end users to wrap services into web gadgets and to combine them into complex web applications. Web applications running inside a webbrowser bring new requirements to the traditional desktop application development process. Since web applications usually don't come to life using model based approaches or Story Driven Modelling the Fujaba Toolsuite has to be adopted to fulfil the new requirements.

The Fujaba group of Kassel wanted to get all the help we are used to get from Fujaba in ordinary application development for our web applications, too. Thus, we are developing a new code generation for the generation of Google Web Toolkit compliant Java code that will then be compiled into JavaScript running on web browsers. In addition, we develop tool support for building the GUIs of such web applications. And, on the server side, we develop technologies to facilitate service development with Fujaba. This paper reports on the design of these new Fujaba web components.

1. INTRODUCTION

Since March 2008 the Fujaba group at Kassel University participates in the European Project FAST: Fast and Advanced Storyboard Tools. This project aims at the development of a new visual programming language and tools that will facilitate the development of complex front-end gadgets. A gadget in this opinion is a small web application that is designed to run inside a so called mashup platform. Some of these platforms like Yahoo Pipes[9], iGoogle[6] or Microsofts Popfly[8] are widely used in the present. Since one of the project partners has already developed a mashup platform, the so called EzWeb platform [3], the Fujaba team at Kassel tries to support gadgets for this platform at first and focuses its research on the specific needs of EzWeb web gadgets. Since Kassel sees gadgets as kind of web application, the research regarding Fujaba and the development of web applications with Fujaba is not only focused on gadgets needed for FAST, but on what we call Web 2.0 applications in general. These applications and their development differ from the standard desktop applications known to be generated with Fujaba. There are many challenges that have to be addressed to get applications running over the web in the way we want them to. The main problem is the lack of possibilities you have on the client. Since the applications are intended to run inside a web browser the client side has to be JavaScript. The old way of web applica-

tions were web formulars on client side which communicate with the server and are completely blocked while waiting for server responses. Web 2.0 applications in our manner will be Ajax applications, meaning that the whole communication between client and server will be asynchronous giving the user possibilities to interact with the application even when waiting for server responses. Another problem you have in the development of web applications is the distribution of data, or more specifically replication of data models between the server and one or multiple clients. In the domain of web applications developing client side code is done by hand coding JavaScript for the browser. The Fujaba team at Kassel wanted to have all the opportunities known from standard software development processes: code generation, model based development and story driven modelling to benefit the development of web applications and gadgets. Thus, we chose to adapt Fujaba and its tools to suffice the new needs.

Figure 1 shows the architecture of what we call Web 2.0 application. On the client side a web browser may show some web gadgets. We propose to program these web gadgets as model view controller patterns, i.e. to separate the representation and the data model. The representation may employ some domain object model of the web page shown to the user. Note, our reference architecture for Web 2.0 applications is somewhat unconventional. Usually, the client has only view data and the server has the model data and the application logic. In our reference architecture, the application logic, i.e. the business rules and model transformations are to a large extent executed on the web clients based on the model data available on that client. For GUI operations we develop a dedicated version of Fujaba Story Diagrams that will provide DOM rewrite rules within concrete syntax, cf. section 2. The application logic is based on an application data model. This application data model and its operation may be developed with usual Fujaba story diagrams. In order to deploy the resulting Java code on a web browser, we will use Google Web Toolkit technology [4], cf. section 3. Synchronization between DOM and application data model is done via the usual property change based event mechanisms.

For the development of (web) services we will use standard Story Driven Modeling technologies. Note, some service models and some application models may overlap. In these cases, we develop the data model only once and deploy it on client and on server side, equally. For certain applications it may suffice to copy (parts of) the server data to the client where the data is visualized and exploited using the

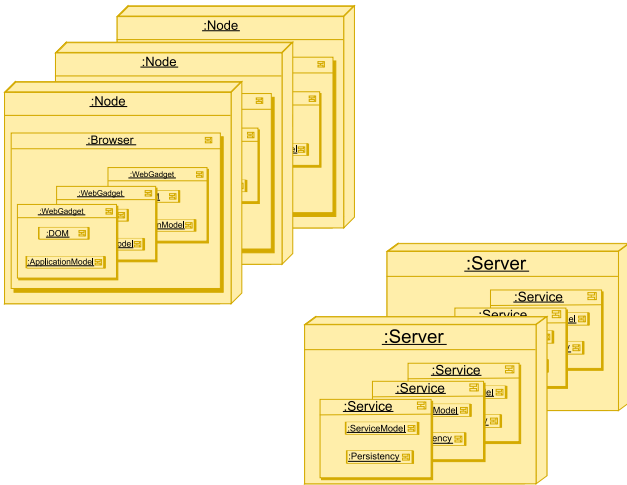


Figure 1: Reference Architecture for Web 2.0 Applications

user interface mechanisms described above. Then, modified data may be copied back to the server in order to make it persistent or in order to share it with other client instances. Therefore, we will provide data replication mechanisms and persistency mechanisms described in section 2.

Next, the problem of service orchestration is addressed by exploiting new Web 2.0 mashup technologies and by a dedicated work flow support service that we develop within the FAST project, cf. section 5.

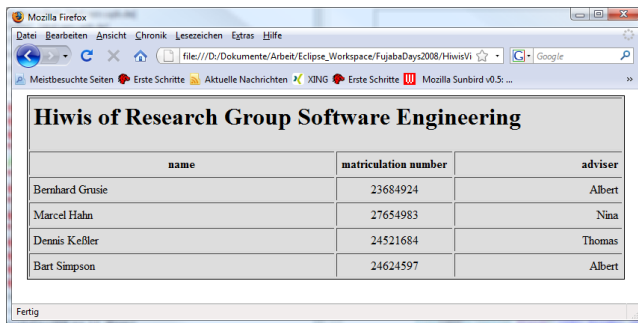


Figure 2: Tutor Management Application

As running example of our approach we modeled a web enabled homework management system for students and tutors. As first feature, the homework management system allows to manage courses, students and tutors. Only the lecturer staff is allowed to do so. This is shown in Figure 2. As second feature, the homework management system supports the grading of students' homeworks. As shown in Figure 3, we award the student homework with points. A certain amount of points are required to pass the course. Security requirements have to be considered to implement that feature, however this is not in our focus at this stage of work. As third feature, it should be possible for students to submit their homework directly via web interface in our system, for example by filling out multiple choice questionnaires, entering answer text or even uploading programming source code. The submitted homeworks should be reviewed

name of student	matriculation number	Points reached	max points: 30
Michael Adam	24260359	20	
Leslie Carter	24257834	29	
Dennis Keffler	24521684	30	
Matthias Ludwig	24392167	4	
Bart Simpson	24624597	5	
Daniel Richard	24796135	27	

Figure 3: Student Assignments Application

and corrected by our tutors. After correcting a homework, the tutor enters the results directly into the system, and students can view their homework results.

Not all of the above requirements need to be fulfilled for this example. It is sufficient to hold the data model on the server, for example. The homework management system in the design described above is more similar to traditional web applications, even if the communication between client and server will be asynchronous. For development of the Fujaba tools needed for new Web 2.0 applications it is a good example, because it will lead to less complex client side code. It will need persistency of the data, user management, multi-user capability and GUI operations, so we can test new Fujaba tools with it. For more complex examples, we will go for the ToDo list gadget developed in terms of the FAST project, cf. 5 which will then be intended to run inside the EzWeb platform and which will have a client side data model, on top.

2. GUI AND DOM OPERATIONS

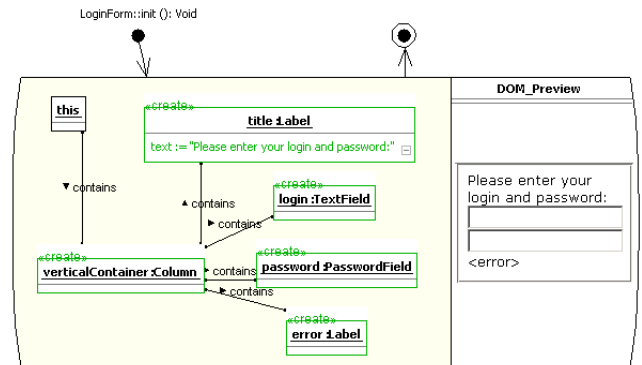


Figure 4: DOM Modeling in Fujaba

The Graphical User Interface (GUI) of our application is realized using up-to-date AJAX (Asynchronous JavaScript and XML) technologies. In order to avoid any direct JavaScript programming, we use the Google Web Toolkit. It provides a Java API with GUI Widgets as abstraction of HTML, CSS and JavaScript browser technologies. Thus, creating and modifying the GUI are just manipulations in a Java object structure representing the DOM (Document Object Model)

in the client browser. The GWT maps those operations to JavaScript calls modifying the HTML document tree. Traditionally, Fujaba Story Diagrams are used to graphically model application behaviour [1], [10]. We are currently trying to adopt these diagrams to also model the GUI in Fujaba. Figure 4 shows an example of specifying a simple login form in Fujaba, along with the concrete syntax preview on the right side.

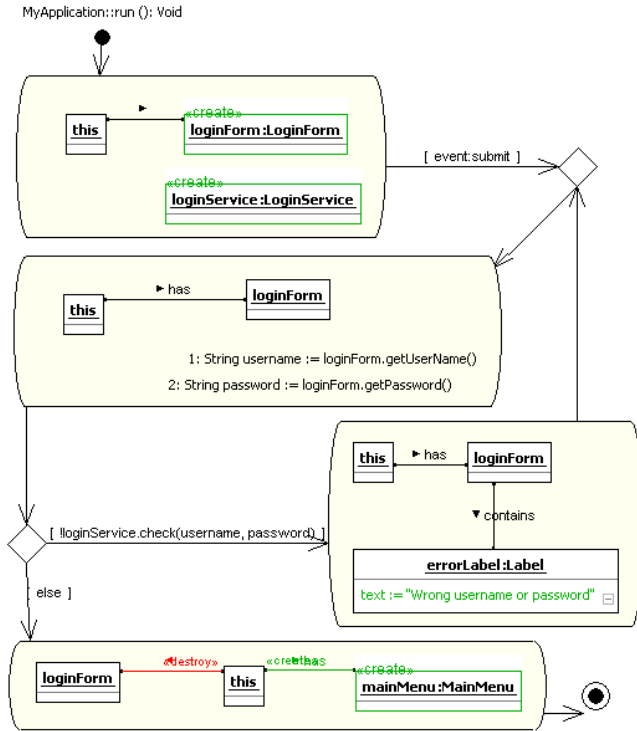


Figure 5: Application Flow Modeling in Fujaba

Changes to the current GUI (layout and data) will be usually performed when a user action occurs. This can be a click on a button, entering text or any other mouse- or keyboard related event. These events will be handled locally in the client. To model the client behavior, we plan to extend the Story Diagram or State Diagram transitions to react on user input events. Figure 5 shows the modeling of such a GUI state change: Initially, the application display a login form. When pressing enter in the text fields, which is mapped to the *submit* event, the application state switches to the main menu or displays an error message.

3. CLIENT SIDE APPLICATION MODEL

For our example assignment management application, the application model is pretty simple, cf. Figure 6.

This application model is specified with Fujaba. Then, we generate Java code for the implementation of the described data. This implementation shall be used to represent runtime data on the client side, i.e. within the client's web browser. Therefore, we use Google's Web Toolkit (GWT) [4] to compile the generated Java code into JavaScript code that is deployed on the web client. Using GWT has the advantage that the template based code generation of Fujaba needs only minimal adaption. Actually, we only need to

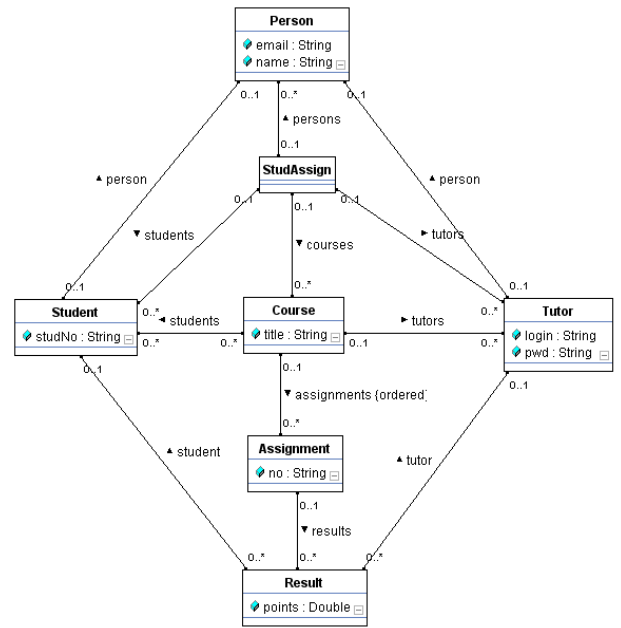


Figure 6: Assignment Management Model

adapt the container classes deployed for the implementation of to-many associations. To implement these associations data structures are needed to manage a set of neighbouring objects. GWT supports the standard Set implementations of the Java Collection API which does not support concurrent modification of its contents. Therefore, Fujaba traditionally uses its own Collection classes. Since GWT is not able to compile this and other Fujaba specific classes into JavaScript we will need to use GWT containers where they are needed. The only thing to do for this is providing new templates for the code generation. In addition to the Java-to-JavaScript compilation, the GWT approach has the advantage, that a special GWT browser allows debugging of client side code. Thus, with the usage of GWT, we will be able to generate code from Story Diagrams including graph rewrite rules and to do design level debugging of the execution of such rewrite rules for web clients.

To enable the model view controller pattern for connecting the clients DOM based GUI with the clients application data model, we need a property change management infrastructure. In [2] the CoObRA framework was presented as a generic property change and replication mechanism which we now have adapted to the specific needs of web application development. Thus, there will be a CoObRA component on our web clients that provides undo redo for the web application and that may be used to send change protocols from the clients to the services. The latter mechanism enables CoObRA based data replication mechanism and team collaborations. Thus, in our example application, multiple tutors may work with the assignment management application concurrently. Each tutor may enter the grades for his or her students and these changes are then replicated / committed to the assignment management service where this data is then replicated to other tutors. Thus, each tutor sees the progress of the overall work and whether some student has already been graded for a certain assignment by some team

mate.

The infrastructure for property changes and for CoObRA like change protocols are current work.

4. SERVICE DEVELOPMENT

On the server side, we deploy a server data model and a persistency component. In simple cases the server data model is the same data model as the application data model. This will e.g. hold for our assignment management service and for our tutor management service. This data model is generated from the same Fujaba model that is used for the generation of the application data model. As can be seen from our reference architecture, the client can have application logic and data model, too. However, in some applications, the client may load only a certain part of the whole data, e.g. for performance or security reasons. In this case some operations that require access to the whole data storage may be deployed on server side. In addition, the server needs to manage which clients are currently online and which client has loaded which part of the overall data. In case of a data update the server has to compute which clients have replicas of the corresponding data fragments and thus require an update notification.

In general, the server component needs to provide some support for the access protocols used to communicate with the web clients e.g. a simple object access mechanism.

In addition, the server needs a persistency component in order to store data e.g. for recovery purposes. Here we use the standard CoObRA mechanisms. Note, these CoObRA mechanisms allow direct storage of each change for recovery as well as a transaction concept grouping change sets for consistency purpose as well as a cvs like update / commit usage.

Using these mechanisms, the generation of simple (data replication) services becomes quite simple. Just generate it from the Fujaba model. However, note that such services have a certain scaling problem. CoObRA based applications / services keep the whole runtime data in main memory and are thus restricted to data model sizes that fit into 2 gigabyte main memory on 32 bit Java virtual machines. This should not easily become a problem for our assignment service or for the tutor service. However, one would not run a banking service or a web shop on this limited data size. To overcome this scaling problem, we think of a combination of a CoObRA mechanism coordinating the data replication and a relational database for management of large data volumes. This is future work.

5. SERVICE ORCHESTRATION

As mentioned before, within the FAST EU project the University of Kassel takes part in the development of tools to support complex web gadgets. In our own research we try to develop a ToDo list gadget with the help of the Fujaba adoptions described in this paper. The ToDo list gadget may be used to organize multiple services, cf. Figure 7.

The idea is that a number of steps is defined where each step refers to some service via an URL. This allows to visit one service after the other. Next, each step may be equipped with named input and output data ports, e.g. for the departure and destination airport for some flight booking service. Via such data ports, the mashup platform may retrieve data as e.g. the flight arrival time from one service and forward

Have a meeting

No.	Action Item	State	Documents	Responsible	Input Data	Output Data
1	Book flights	<input checked="" type="checkbox"/>	http://www.expedia.de/fluege/li-nienfluege	Nina	dates, airport of departure, destination airport	e-tickets, departure times, arrival times
2	Book hotel	<input checked="" type="checkbox"/>	http://www.expedia.de/hotels/default	Ruben	dates, location, estimated price	Booking confirmation
3	Have presentations	<input checked="" type="checkbox"/>	http://docs.google.com/Doc?docid=xxxxxxxxxxxx3	Albert	work done, ideas	-
4	Have social dinner	<input checked="" type="checkbox"/>	-	UPM	date, time, number of persons	a lot of fun
5	Reimbursement for travel expenses	<input type="checkbox"/>	http://www.uni-kassel.de/pvabt3/download/reisekosten.gtk	Nina, Ruben, Albert	dates, bills	money

Figure 7: ToDo List Gadget for Service Orchestration

it to another service. Together with some means for the automation of todo item execution, this may help to organize and to orchestrate multiple services within a common work flow. The emerging software can be run either as stand alone web application, or inside the mashup platform. Combining client side application model, multi user execution with data replication, automated todo item execution and data persistency the web based workflow management system can be seen as complex Web 2.0 application nearly entirely developed with Fujaba. One major step with this application will be the deployment inside EzWeb as mashup platform, since it is not used, maybe not even intended for gadgets as complex as our workflow example. The main difference between traditional gadgets and ours is not only its complexity, but also the fact that the workflow management system not only uses web services inside the todo item, it needs its own services in the background to establish persistency and data replication via the CoObRA mechanism. These new ideas have to be carefully transported to EzWeb together with possibilities which enable the end-user to wrap the services they want to use inside their workflow, which means producing tools to help the user to define in and out ports of the todo item in an easy way. For this purpose the Kassel University team currently tries to develop so called Clipboards to enable point and click mechanisms to do this. These technologies are in a very early stage and will be issued in detail in future publications.

6. SUMMARY AND FUTURE WORK

Honestly, most of the described ideas are current and future work. Currently, we have a prototype of a ToDo List gadget that has been build according to the reference architecture. Because the GUI building mechanisms are not yet ready to be used, we built the user interface by hand. The data replication and persistency were at first built upon a foreign GWT library called jstm4gwt [7]. Many parts of this library had to be adopted, because the property change mechanisms didn't work as expected and so we have adopted the CoObRA mechanisms for GWT earlier than expected to get rid of this library. Currently we are moving our application from the jstm4gwt version to a new one that will use

the CoObRA mechanism. Nevertheless, we could already prove that web based workflow handling is possible with the jstm4gwt version.

To get the GUI development more comfortable, besides the enhanced Fujaba Story Diagrams we plan to develop a GUI designer which will support standard GWT and GWT-Ext [5] Widgets for the programming of user interfaces. The GUI designer is planned as plugin for Fujaba4Eclipse and will enable the user to graphically design the GUI. As a reference we will try to use the former Eclipse plugin Visual Editor, which was able to build SWT and Swing GUIs inside Eclipse in a visual way. Out of the graphical representation the sourcecode was generated. The Fujaba GWT GUI designer will not generate sourcecode directly out of the graphical Widget representations, it will generate the appropriate classes and objects that will be displayed in Fujaba classdiagrams and Story Diagrams.

For further enhancement of the Web 2.0 application development process we plan to extend Fujaba in a way to automatically generate the GWT Module definition, entry class and .html host website when a web application is the intended target. Solving the problems stated in the introduction with well known software engineering principles will yield new tools and frameworks in the Fujaba Toolsuite context raising the software development process for web applications to a model centric level, yet unknown.

7. REFERENCES

- [1] T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph rewrite language based on the unified modeling language. In *Proc. of the 6th International Workshop on Theory and Application of Graph Transformation*. Paderborn, Germany, 1998.
- [2] C. Schneider. *CoObRA: Eine Plattform zur Verteilung und Replikation komplexer Objektstrukturen mit optimistischen Sperrkonzepten*. PhD thesis, 2007.
- [3] EzWeb. <http://ezweb.morfeo-project.org/>, 2008.
- [4] The Google Web Toolkit. <http://code.google.com/webtoolkit>, 2008.
- [5] GWT-Ext. <http://code.google.com/p/gwt-ext/>, 2008.
- [6] iGoogle. <http://www.google.com/ig>, 2008.
- [7] XSTM. <http://www.xstm.net/>, 2008.
- [8] Microsoft popfly. <http://www.popfly.com/>, 2008.
- [9] Yahoo Pipes. <http://pipes.yahoo.com/pipes/>, 2008.
- [10] A. Zündorf. Rigorous object oriented software development. Habilitation Thesis, University of Paderborn, 2001.