Enriching Web Applications with Collaboration Support Using Dependency Injection

Matthias Heinrich[1], Franz Josef Grüneberger[1],
Thomas Springer[2], and Martin Gaedke[3]

[1] SAP Research, Germany
{matthias.heinrich,franz.josef.grueneberger}@sap.com
[2] Dresden University of Technology, Germany
thomas.springer@tu-dresden.de
[3] Chemnitz University of Technology, Germany
martin.gaedke@cs.tu-chemnitz.de

Abstract. Web-based collaboration tools such as Google Docs are pervasive in our daily lives since they have proven to efficiently support joint work of distributed teams. Nevertheless, the development of web-based groupware systems is a time-consuming and costly task because developers either have to become familiar with specific groupware libraries or are asked to re-implement concurrency control services (i.e. document synchronization, conflict resolution). Therefore, we propose a dependency injection mechanism using declarative annotations to incorporate concurrency control services into web applications. Instead of adopting comprehensive libraries or implementing application-specific components, synchronization capabilities are integrated in a lightweight and rapid fashion. To validate the approach, we enriched the widely-adopted Knockout framework with dependency injection facilities and transformed two Knockout-based applications into collaborative ones.

1 Introduction

In the course of the Web 2.0 movement, numerous collaborative web applications such as Google Docs or EtherPad have emerged and were rapidly adopted. In contrast to single-user web applications, the development of collaborative web applications requires additional services such as document synchronization and conflict resolution. While the document synchronization is in charge of reconciling all documents copies, the conflict resolution mechanism handles conflicts emerging when multiple users simultaneously change the very same document artifacts. To incorporate concurrency control services (i.e. document synchronization, conflict resolution), developers either have to learn new Application Programming Interfaces (APIs) or are asked to implement the required functionality themselves. Both traditional approaches are time-consuming and costly. Therefore, we propose to annotate existing single-user applications with dependency injection tags which can, in contrast to traditional approaches, considerably ease the task of integrating groupware-specific features. Thus, the groupware development efficiency can eventually be increased.

M. Brambilla, T. Tokuda, and R. Tolksdorf (Eds.): ICWE 2012, LNCS 7387, pp. 473–476, 2012.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 describes the system architecture, the workflow to inject collaboration services as well as the demo applications and Section 4 exhibits conclusions.

2 Related Work

In this section, we expose a number of approaches promising to rapidly integrate concurrency control services in web applications.

The generic transformation approach [1] aims to enhance existing single-user web applications with shared editing capabilities exploiting the so-called Generic Collaboration Infrastructure (GCI). The GCI allows recording, propagating and replaying arbitrary Document Object Model (DOM) manipulations among all sites. In contrast to our approach, the GCI cannot synchronize web applications leveraging a separate JavaScript data model (e.g. Knockout-based applications).

Apache Wave [2] is a full-fledged collaboration framework facilitating concurrency control and allowing to either create *extensions* or *client applications*. While extensions are defined adopting a specific XML syntax, client applications are built using a Java or Python API. Therefore, Apache Wave is dedicated to develop widgets or applications from scratch rather than enriching existing ones.

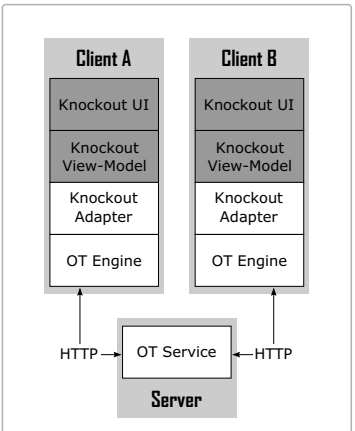
ShareJS [3] and OpenCoWeb [4] are two JavaScript libraries supplying concurrency control services. Including document objects in the synchronization procedure requires various API calls (e.g. object registration, value propagation or callback implementation). In comparison to our compact dependency injection syntax, the libraries expose a verbose binding language entailing cumbersome and scattered code changes.

3 System Architecture and Demonstration

Dependency Injection (DI) has proven to be an efficient means to eliminate boilerplate code and thus, it has been adopted in numerous development toolkits (e.g. the Java Enterprise Edition 6 or the Eclipse e4 framework). We leverage DI in order to furnish a lightweight integration approach capable of speeding up the incorporation of concurrency control services in web applications. In this section, we present the devised collaboration architecture and the enhanced Knockout[1] framework [5] which has been enriched with DI facilities.

The system architecture materializing the approach of concurrency control injection is depicted in Figure 1(a). The shown sync server connects numerous clients and provides a sync service that is based on the prevalent concurrency control algorithm called Operational Transformation (OT) [6]. All clients exhibit a stack encompassing Knockout components (the UI and the View-Model) as well as synchronization components (the Knockout Adapter and the OT Engine). The OT Engine is in charge of sending out local changes, receiving remote changes

[1] We chose to enrich the Knockout framework because of its massive developer adoption (e.g. Knockout 2 reached 110 000 downloads in 3 months).



(a) Sync System Architecture

```
...
  <!-- <script type="text/javascript" src="viewmodel.js"/> -->
  <script type="text/javascript" src="knockoutadapter.js"/>
...
  <input data-bind="value: input"/>
  <button data-bind="click: addTask">Add Task</button>
  <ul data-bind="foreach: tasks">
     <li>
         <span data-bind="text: name"></span>
         <a href='#' data-bind="click: delete">Delete Task</a>
     </li>
...
```

(b) HTML View incl. Knockout-Adapter and -Bindings

```
...
 // @Session("MySession")

 // @Class("Task")
 var Task = function (data) { this.name = ko.observable(data.name) }

 Task.prototype.delete = function() { viewModel.tasks.remove(this) }

 // @Sync("viewModel")
 var viewModel = { input: ko.observable(),
                   tasks: ko.observableArray() }

 viewModel.addTask = function() {
        viewModel.tasks.push(new Task({'name': viewModel.input()}));
        viewModel.input("") }

 ko.applyBindings(viewModel);
...
```

(c) Knockout View-Model enhanced with DI Annotations

Fig. 1. System architecture and minimal dependency injection example

and incorporating all those modifications in a dedicated OT data model. The associated Knockout Adapter links the OT model with the View-Model (VM), i.e. VM changes are propagated to the OT model and vice versa.

To introduce the workflow enriching Knockout applications with concurrency control support, we use a minimal example where multiple users can simultaneously edit a list of tasks. Figure 1(b) depicts an excerpt of an HTML page representing the Knockout UI. The main HTML elements are the input element to enter the task name, a button to add the new task and a list showing all tasks accompanied by a delete button. Additionally, Knockout-specific `data-bind` expressions establish data-bindings to the VM. Our UI enhancements, highlighted using bold text, are limited to the replacement of the original Knockout VM (encapsulated in the `<!-- / -->` tags) with our "knockoutadapter.js" script. This script contains the generic sync adapter as well as the parser logic which allows to locate and eventually to replace DI annotations with the actual source code carrying out the synchronization. Moreover, the script imports an application-specific configuration to specify the file name of the VM, the VM elements that should be excluded from the sync, etc. The VM associated to the HTML view is illustrated in Figure 1(c) whereas changes to the original VM are reflected once again in bold text. The complete set of DI annotations encompasses @Session, @Class and @Sync annotations which are always accommodated in comments to prevent JavaScript errors. The @Session annotation enables session management by exploiting the Session-ID argument (e.g. "MySession"). The @Sync annotation specifies the model that should by synchronized among all clients. In our example, the `input` property and the `tasks` array are part of the sync model. Note that Knockout VMs can comprise three types of observable objects

(properties, computed properties and arrays) which all supply a notification mechanism capable of informing subscribers about changes. This notification mechanism is exploited to record local changes which are eventually propagated. Besides recording changes, remote edits have to be replayed locally. Therefore, the last annotation @Class marks object constructors allowing the sync mechanism to re-create objects (e.g. a `Task`) using the appropriate constructor function.

To validate our approach, we implemented the proposed architecture on top of the OT platform SAP Gravity [7] and the widely-adopted Knockout framework. In the validation we included two Knockout-based to-do applications. While TodoMVC [8] is a ready-for-use single-user application, our MyTodoApp was developed from scratch. Both applications were enhanced with DI annotations injecting concurrency control services and eventually could support collaborative work. The results are exposed on our demo page `http://vsr.informatik.tu-chemnitz.de/demo/DI/`.

4 Conclusion

In this paper, we presented an approach to add concurrency control support by means of dependency injection. In contrast to adopting verbose programming libraries, the proposed approach exposes an easy-to-learn and compact syntax. Thus developers are empowered to efficiently program new collaborative applications or to rapidly migrate existing single-user applications to collaborative applications. Even though the Knockout framework was exclusively enriched with dependency injection facilities, the approach could be transferred to other JavaScript libraries exhibiting a separation of the view and the data model. Moreover, we noted that besides concurrency control, workspace awareness is another essential collaboration service exposing what other users are doing in the shared space. Thus, injecting awareness services is a challenging research question we will try to tackle in future work.

References

1. Heinrich, M., Lehmann, F., Springer, T., Gaedke, M.: Exploiting single-user web applications for shared editing: a generic transformation approach. In: WWW, pp. 1057–1066 (2012)
2. Apache Software Foundation: Apache Wave (2011), `http://incubator.apache.org/wave/`
3. Gentle, J.: ShareJS – Concurrent editing in your app. (2012), `http://sharejs.org/`
4. The Dojo Foundation: OpenCoWeb Framework (2012), `http://opencoweb.org/`
5. Sanderson, S.: Knockout: Home (2012), `http://knockoutjs.com/`
6. Ellis, C.A., Gibbs, S.J.: Concurrency Control in Groupware Systems. In: SIGMOD Conference, pp. 399–407 (1989)
7. Rickayzen, A.: Simple way to model processes in the Web (2011), `http://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/25360`
8. Osmani, A., Boushley, A., Sorhus, S.: TodoMVC (2012), `http://addyosmani.github.com/todomvc/`