Analyzing the Suitability of Web Applications for a Single-User to Multi-User Transformation

Matthias Heinrich
SAP AG
matthias.heinrich@sap.com

Franz Lehmann
SAP AG
franz.lehmann@sap.com

Franz Josef Grüneberger
SAP AG
franz.josef.grueneberger@sap.com

Thomas Springer
Dresden University of Technology
thomas.springer@tu-dresden.de

Martin Gaedke
Chemnitz University of Technology
martin.gaedke@cs.tu-chemnitz.de

ABSTRACT

Multi-user web applications like Google Docs or Etherpad are crucial to efficiently support collaborative work (e.g. jointly create texts, graphics, or presentations). Nevertheless, enhancing single-user web applications with multi-user capabilities (i.e. document synchronization and conflict resolution) is a time-consuming and intricate task since traditional approaches adopting concurrency control libraries (e.g. Apache Wave) require numerous scattered source code changes. Therefore, we devised the Generic Collaboration Infrastructure (GCI) [8] that is capable of converting single-user web applications non-invasively into collaborative ones, i.e. no source code changes are required. In this paper, we present a catalog of vital application properties that allows determining if a web application is suitable for a GCI transformation. On the basis of the introduced catalog, we analyze 12 single-user web applications and show that 6 are eligible for a GCI transformation. Moreover, we demonstrate (1) the transformation of one qualified application, namely, the prominent text editor TinyMCE, and (2) showcase the resulting multi-user capabilities. Both demo parts are illustrated in a dedicated screencast that is available at `http://vsr.informatik.tu-chemnitz.de/demo/TinyMCE/`.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Computer supported cooperative work, Synchronous interaction, Web-based interaction*

Keywords

Groupware, Shared Editing, Web Applications, Web Engineering

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

1. INTRODUCTION

According to McKinsey's report "The Social Economy" [6], the use of social communication and collaboration technologies within and across enterprises can unlock up to $860 billion in annual savings. Collaborative web applications represent an essential class from the group of collaboration technologies that allow multiple users to edit the very same document simultaneously. In contrast to single-user applications (e.g. Microsoft Word) which have to rely on traditional document merging or document locking techniques in collaboration scenarios, shared editing solutions such as Google Docs or Etherpad improve collaboration efficiency due to their advanced multi-user capabilities including real-time document synchronization and automatic conflict resolution.

Even though collaborative web applications have enormous potential allowing to jointly create text documents, spreadsheets, presentations, source code files, CAD models, etc., the large majority of web applications available, for example, in the Chrome Web Store or the Firefox Marketplace, offers solely single-user capabilities. The lack of multi-user facilities stems from the fact that implementing collaboration capabilities is a time-consuming and cumbersome task. Enhancing single-user web applications with multi-user features in a conventional manner requires developers to get familiar with a concurrency control library (e.g. Apache Wave [1], ShareJS [7], etc.) as well as with the application's source code. Additionally, integrating collaboration capabilities entails a multitude of scattered source code changes to capture and replay document manipulations. This task is particularly tedious for mature single-user web applications like the TinyMCE text editor [5] encompassing more than 70 000 lines of JavaScript code or the SVG-edit graphics editor [4] surpassing 30 000 lines of JavaScript code.

To ease the integration of collaboration capabilities, Sun et al. introduced the *transparent adaptation* approach aiming "to convert existing single-user applications into collaborative ones, without changing the source code of the original application" [11]. Disburdening developers from modifying the original source code comes at the cost of having to implement a specific collaboration adapter for each and every single-user application. This collaboration adapter links the original application to a reusable collaboration engine enabling the synchronization of various document instances. To speed up the application transformation for single-user

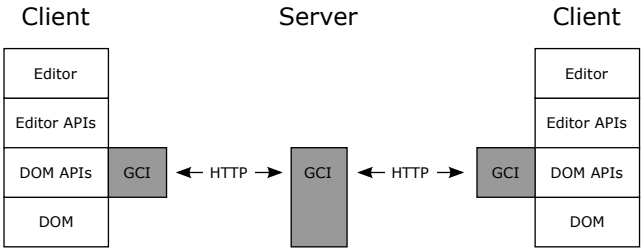

Figure 1: Architecture of the GCI [8]

web applications, Heinrich et al. devised the Generic Collaboration Infrastructure (GCI) [8] depicted in Figure 1. The key benefit of the GCI is the application-agnostic change recording and change replay feature that does not require an extra collaboration adapter for each application. Providing a generic collaboration infrastructure becomes possible since the capture and replay logic exploit standardized W3C DOM APIs (e.g. the DOM Core [9] or the DOM Events specification [10]) instead of leveraging application-specific editor APIs (cf. Figure 1). Hence, a large variety of standards-based web applications can adopt the GCI since they all share a common technological foundation, namely, the well-defined Document Object Model (DOM). In contrast to the transparent adaptation approach, the GCI-based transformation is more efficient due to the fact that only a specific GCI configuration file is required instead of a specific collaboration adapter implementation.

In [8], the GCI approach was introduced and the technical foundations were discussed in detail. In this paper, we analyze 12 single-user web applications from different application domains (e.g. word and spreadsheet processors, graphics editors, integrated development environments, etc.) to check whether they are eligible for the GCI transformation approach. Therefore, we systematize the GCI eligibility check by establishing a criteria catalog containing properties that single-user web applications have to fulfill to adopt the GCI. Moreover, we demonstrate the transformation process in detail leveraging the TinyMCE text editor [5].

The main contributions of this paper are three-fold:

- We compile a criteria catalog identifying necessary as well as critical application properties for the GCI transformation and present approaches to test these criteria.
- We analyze 12 single-user web applications leveraging the established criteria catalog and show that from 6 eligible applications 4 were successfully transformed to collaborative editors.
- We walk through the transformation process of the single-user text editor TinyMCE and demonstrate its resulting multi-user capabilities.

The rest of this paper is organized as follows: Section 2 exposes essential application properties that qualify for a GCI transformation and Section 3 discusses the eligibility analysis of 12 single-user web applications. Afterwards, Section 4 illustrates the exemplary single-user to multi-user transformation using the TinyMCE editor and Section 5 draws conclusions.

2. APPLICATION CRITERIA CATALOG

To evaluate the suitability of web applications for a GCI transformation, we compiled a list of *necessary* and *critical* application properties. While necessary application properties have to be fulfilled in order to allow for a GCI transformation, the non-fulfillment of critical properties does not exclude applications from the set of convertible applications. However, these applications may suffer from an impaired GCI performance or functionality. Furthermore, we describe approaches to efficiently check whether applications exhibit or do not exhibit certain properties.

The list of necessary application properties encompasses (1) the W3C standards compliance and (2) the DOM-based data model.

W3C standards compliance is required since the document change capturing and the document change replay is built on top of W3C standards (the DOM Core [9] and the DOM Events [10] recommendation). Conventional plugin technologies such as Adobe Flash or Microsoft Silverlight do neither comply with the DOM Events specification nor with the DOM Core standard. Hence, the record manipulations process (based on DOM Events) and the replay manipulations process (based on the DOM Core) is bypassed which disqualifies web applications leveraging plugin technologies. Testing whether web applications include elements based on plugin technologies is straightforward. First, deactivating plugins in the respective browser leads to an incomplete rendering of a plugin-based website. Second, <object> tags in the HTML source code hint the usage of plugin technologies since the <object> element marks resources that are processed by external plugins.

A **DOM-based data model** represents the second necessary characteristic. Since the DOM is the only standardized representation all modern browsers can process in a uniform way, it is essential that the data model is accommodated in the DOM. If the data model is not encapsulated in the DOM, the GCI synchronization mechanism breaks since standardized DOM APIs can no longer be exploited. Web applications structured according to the established Model-View-Controller (MVC) pattern are one example where the data model is not included in the DOM. In this case, only the view is represented in the DOM and the model is represented by a separate JavaScript data structure. The infinite set of constructible JavaScript data structures doesn't allow developing a generic record and replay mechanism and thus, these applications are not supported by the GCI. An option to discover the existence of a JavaScript data model are so called *DOM breakpoints* provided by the Chrome Developer Tools. A DOM breakpoint is a marker on a specific DOM node that notifies about certain node manipulations (e.g. subtree or attribute modifications). Once a DOM breakpoint is hit, the debugger jumps to the JavaScript code that was triggering the DOM node manipulation. Since changes to the view (represented in the DOM) commonly entail model changes, these DOM breakpoints can reveal where the data model resides.

Besides necessary application properties, there are the critical application properties: (1) model isolation, (2) DOM event frequency and (3) multi-user ready identification.

Model isolation is a critical application property enabling the seamless sync of the data model. In contrast to the data model, view-related aspects like toolbar selections, window size, scrolling position, etc., should not be synchro-

nized since these characteristics are individual to each virtual workspace. Nevertheless, web applications do not always strictly separate data model and view-related aspects in distinct DOM subtrees which impairs the GCI sync that operates on selected DOM subtrees [8]. Hence, if data model and view aspects are intermingled in the same DOM subtree, the sync incorrectly includes view elements. Analyzing if the data model is isolated can be accomplished using the DOM element inspector that is, for example, available in the Chrome Developer Tools. The DOM inspector allows selecting an element in the rendered website which triggers showing the associated DOM subtree. Selecting the editing pane of a text editor or the canvas of a graphics editor highlights the respective DOM subtrees and allows investigating linked child elements. This test is an efficient means to check if the data model is clearly separated from the view.

The **DOM event frequency** is critical with respect to the GCI performance. Currently, the DOM mutation event rate should not surpass multiple hundreds of events a second since this is the upper limit the latest GCI implementation is able to process. For example, these situations may arise carrying out drag-and-drop operations in graphics editors whereas the dragged object (e.g. a circle shape) changes its x and y coordinates hundreds of times a second. Another example are group operations where dozens or hundreds of DOM nodes are affected, e.g. a cut-and-paste operation involving numerous pages in a text document. The magnitude of these high-load scenarios can be measured attaching a JavaScript event listener (e.g. node.addEventListener(...)) to a specific DOM subtree whereas the listener logic computes the events-per-second rate.

Multi-user ready identification is the last critical application property we identified. Web applications that were not meant to be used in multi-user scenarios may adopt a simple naming scheme for referencing document artifacts that breaks when linking various application instances. For example, the graphics editor SVG-edit references created shapes using an incremented integer. If two users simultaneously construct a new shape in different workspaces, both shapes receive the very same integer ID leading to an incorrect application behavior. Identifying the applied identification scheme can be achieved in an interactive debugging session where a DOM breakpoint is installed on the data model root node and new model objects are created which leads to the logic that assigns IDs to created objects.

3. APPLICATION ELIGIBILITY TEST

Taking into account the necessary and critical application properties, we analyzed a set of 12 single-user web editors that are listed in Table 1. Thereby, we selected solely open-source applications that are widespread and also adopted by a large community. The selection process also ensured that the applications cover a multitude of domains (e.g. text editing, source code editing, etc.). Conducting the analysis, we tested applications with respect to the compiled criteria catalog (cf. Section 2) excluding the DOM event frequency property since an adequate test would require excessive and time-consuming editor usage in a variety of scenarios due to the fact that DOM event rate peaks can occur carrying out a multitude of differing editor operations (e.g. copy-and-paste operations affecting numerous DOM nodes). If one of the two necessary properties was not met, we omitted the test addressing critical application properties.

Editor Name	Editing Domain	Standards Compliance	DOM-based Data Model	Model Isolation	Multi-User Readiness
ACE Editor	Source Code	Yes	No	-	-
Canvas Painter	**Pixel Graphics**	**Yes**	**Yes**	**Yes**	**No**
CKEditor	**Rich Text**	**Yes**	**Yes**	**Yes**	**Yes**
Eclipse Orion	Source Code	Yes	No	-	-
GelSheet	Spreadsheets	Yes	No	-	-
ImageBot	**Pixel Graphics**	**Yes**	**Yes**	**No**	**Yes**
jQuerySheet	**Spreadsheets**	**Yes**	**Yes**	**Yes**	**No**
Ketcher	Chemical Structures	Yes	No	-	-
Popcorn Maker	Video	Yes	No	-	-
SVG-edit	**SVG Graphics**	**Yes**	**Yes**	**Yes**	**No**
TinyMCE	**Rich Text**	**Yes**	**Yes**	**Yes**	**Yes**
Zwibbler	Pixel Graphics	Yes	No	-	-

Table 1: Results of the Application Eligibility Test

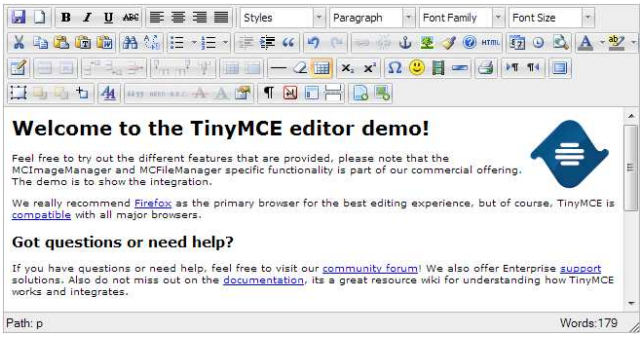

Figure 2: Screenshot of the TinyMCE Editor [5]

Ultimately, 6 editors (marked bold) from the set of 12 satisfy the necessary application properties and hence are eligible for a GCI transformation. Note that 50 percent of the analyzed applications expose an external data model which shows the large adoption of the MVC pattern in the web application ecosystem. The notion of structuring applications according to the MVC principle is promoted by numerous web application frameworks (e.g. Knockout or Backbone.js) that enforce applications to be divided into model, view and controller components. Nevertheless, the analysis shows that the GCI is a viable option for numerous applications from multiple domains.

From the set of 6 eligible editors, we converted 4 applications (CKEditor [2], jQuerySheet [3], SVG-edit [4] and TinyMCE [5]) and adopted the resulting collaborative counterparts in real-life collaborative scenarios. While the multi-user versions of the CKEditor and the SVG-edit were exploited in an extensive user study with 30 participants showing that transformed editors decently support collaborative work [8], the converted jQuerySheet editor was used for an internal project. The fourth editor, TinyMCE, was enriched with multi-user capabilities in preparation for the demo described in Section 4.

4. DEMO

In this section, we introduce the popular TinyMCE text editor [5] and demonstrate its single-user to multi-user transformation as well as the resulting multi-user capabilities.

The TinyMCE word processor is a prominent single-user web application depicted in Figure 2. The editor offers common rich text operations such as creating and formatting text documents as well as inserting tables, hyperlinks, im-

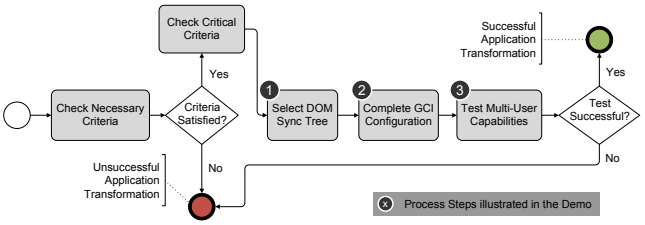

Figure 3: Overview of the Single-User to Multi-User Transformation Process

ages, videos, etc. The open-source project encompassing 70 000 lines of JavaScript code represents a widely-adopted text editor that already surpassed 350 000 downloads of its current version 3.5. Even though TinyMCE could be exploited for a variety of multi-user use cases (e.g. collaboratively writing scientific publications or project proposals), the text editor does not yet offer shared editing capabilities.

In essence, transforming the TinyMCE editor requires going through the entire single-user to multi-user transformation process depicted in Figure 3. Since our application eligibility analysis showed that TinyMCE meets all necessary and critical application properties (cf. Table 1), our demo focuses on the three remaining steps, i.e. (1) the DOM sync tree selection, (2) the GCI configuration and (3) the demonstration of injected multi-user capabilities. This GCI demo is also captured in a screencast available at `http://vsr.informatik.tu-chemnitz.de/demo/TinyMCE/`.

The DOM sync tree selection represents demo step (1). Note that synchronizing the entire DOM would confuse participants in a collaborative editing session since individual editor aspects like selecting a tool in the editor's toolbar would also be synchronized among all participants which is not the desired behavior. To ensure that only the editor's document is synchronized, the specific DOM subtree accommodating the document has to be defined in the GCI configuration. Hence, we start the demo launching the unchanged current release version of the TinyMCE and use the DOM Inspector, which is part of the Chrome Developer Tools, to mark the editing pane in the rendered TinyMCE web page. Marking the rectangular editing pane highlights the respective subtree in the tree representation of the DOM. Thus, we can identify the root node ID of the DOM subtree encompassing the editor document.

In demo step (2), we use a text editor to complete the GCI configuration in the *gci-config.js* file. First, we specify the DOM sync tree in the GCI configuration entering the DOM node ID that was determined in demo step (1). Moreover, we complete the GCI host name setting in the *gci-config.js*. These two settings are the only required GCI configuration properties. The last activity of demo step (2) is the embedment of the *gci.js* script in TinyMCE's main HTML page. The *gci.js* script encapsulates all logic that allows capturing, distributing and replaying DOM manipulations.

Demo step (3) concludes the TinyMCE transformation illustrating the injected multi-user capabilities. Therefore, we first launch the Tomcat-based GCI server and various Chrome browser instances. Afterwards, we open the URL of the TinyMCE in each browser instance and start working collaboratively. This collaborative editing session ranges from basic text operations such as enter, change or delete text to advanced rich text operations such as embed images, create tables, change font size or font face. Moreover, we invite other participants to join the collaborative session to illustrate the GCI capabilities in terms of sync speed and conflict resolution.

5. CONCLUSIONS

The generic collaboration infrastructure represents a capable means to transform existing single-user web applications into collaborative ones with minimal effort. Thereby, DOM APIs are exploited to capture and replay DOM manipulations in an application-agnostic fashion. Nevertheless, not all standards-based web applications are eligible for a GCI transformation. Therefore, we exposed necessary and critical application properties that allow assessing whether a web application may or may not adopt the GCI transformation approach. Moreover, we showed how necessary and critical criteria can be tested. To exhibit the simplicity of the GCI transformation process, we showcased the transformation procedure using the widely-adopted text editor TinyMCE and eventually demonstrated the resulting multi-user capabilities.

6. ACKNOWLEDGMENTS

This work was partially supported by funds from the European Commission (project OMELETTE, contract number 257635).

7. REFERENCES

[1] Apache Wave. The Apache Software Foundation. `http://incubator.apache.org/wave/`, 2013.

[2] CKEditor. `http://ckeditor.com/`, 2013.

[3] jQuery.sheet - The Web-Based Spreadsheet. `http://code.google.com/p/jquerysheet/`, 2013.

[4] SVG-edit – A Complete Vector Graphics Editor in the Browser. `http://code.google.com/p/svg-edit/`, 2013.

[5] TinyMCE – Home. Moxiecode Systems AB. `http://www.tinymce.com/`, 2013.

[6] M. Chui, J. Manyika, J. Bughin, R. Dobbs, C. Roxburgh, H. Sarrazin, G. Sands, and M. Westergren. *The Social Economy: Unlocking Value and Productivity through Social Technologies*. McKinsey Global Institute, 2012.

[7] J. Gentle. ShareJS - Live Concurrent Editing in your App. `http://sharejs.org/`, 2013.

[8] M. Heinrich, F. Lehmann, T. Springer, and M. Gaedke. Exploiting Single-User Web Applications for Shared Editing - A Generic Transformation Approach. In *WWW*, pages 1057–1066, 2012.

[9] A. L. Hors and P. L. HÃl'garet. Document Object Model (DOM) Level 3 Core Specification. `http://www.w3.org/TR/DOM-Level-3-Core/`, 2004.

[10] D. Schepers and J. Rossi. Document Object Model (DOM) Level 3 Events Specification. `http://www.w3.org/TR/DOM-Level-3-Events/`, 2011.

[11] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai. Transparent Adaptation of Single-User Applications for Multi-User Real-Time Collaboration. *ACM Trans. Comput.-Hum. Interact.*, 13:531–582, 2006.