Science of Computer Programming 94 (2014) 53–66

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Enriching single-user web applications non-invasively with shared editing support

Matthias Heinrich [a,*], Franz Lehmann [a], Franz Josef Grüneberger [a], Martin Gaedke [b], Thomas Springer [c], Alexander Schill [c]

[a] SAP AG, SAP Research, Dresden, Germany
[b] Dept. of Computer Science, Chemnitz University of Technology, Chemnitz, Germany
[c] Dept. of Computer Science, Dresden University of Technology, Dresden, Germany

HIGHLIGHTS

- We propose a transformation approach allowing to non-invasively incorporate shared editing capabilities in existing single-user web applications.
- We report on a user study with 30 subjects assessing collaboration qualities in joint work scenarios leveraging two converted editors.
- We carve out characteristics web applications have to expose in order to adopt the generic transformation approach.

ARTICLE INFO

Article history:
Received 17 October 2012
Received in revised form 28 May 2013
Accepted 30 July 2013
Available online 13 August 2013

Keywords:
Web applications
Shared editing
Groupware

ABSTRACT

Collaborative real-time applications like Google Docs allow multiple users to edit the very same document simultaneously which supersedes traditional document merging and document locking techniques. However, developing collaborative web applications is a time-consuming and complex endeavor since it requires implementing document synchronization and conflict resolution services. To accelerate the development of collaborative web applications, we present a rapid transformation approach allowing to non-invasively introduce shared editing capabilities into existing single-user web applications. Instead of changing the application's source code, our non-invasive approach leverages a generic collaboration infrastructure that requires only a configuration to provide document synchronization and conflict resolution services. Hence, the effort to incorporate shared editing capabilities is considerably reduced in contrast to conventional approaches where the use of a programing library entails scattered source code changes. Moreover, we report on the results of a user study demonstrating that converted editors are convenient for collaborative work.

1. Introduction

In the light of the globalization, geographically dispersed teams became a pervasive phenomenon in international organizations as well as in large enterprises. Supporting distributed teams requires appropriate tools such as collaborative real-time editors. In contrast to single-user applications, collaborative editors allow numerous users to edit the very same document in parallel while ensuring document consistency through a dedicated concurrency control mechanism.

* Corresponding author.
E-mail addresses: matthias.heinrich@sap.com (M. Heinrich), franz.lehmann@sap.com (F. Lehmann), franz.josef.grueneberger@sap.com (F.J. Grüneberger), martin.gaedke@cs.tu-chemnitz.de (M. Gaedke), thomas.springer@tu-dresden.de (T. Springer), alexander.schill@tu-dresden.de (A. Schill).

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.07.017

Nowadays, a myriad of web applications support solely single-user scenarios even though the targeted application domain (e.g. text editing) is also suited for collaborative work. For example, there are web-based word processors (CKEditor [1], TinyMCE [2]), graphics editors (SVG-edit [3]) or development environments (Eclipse Orion [4]) lacking native multi-user support. To unfold the potential of single-user applications for collaborative work, shared editing capabilities have to be incorporated, which induces the need for concurrency control services (i.e. document synchronization and conflict resolution). While document synchronization allows synchronizing numerous document copies without notable delay, conflict resolution is capable of resolving shared editing conflicts (e.g. if two users simultaneously add a character at the very same document position or assign different fonts to the very same paragraph). Traditionally, concurrency control services are incorporated using programming libraries and thus, developers have to get familiar with the application's source code as well as with the concurrency control library itself. Moreover, developers have to implement the collaboration functionality entailing verbose and scattered source code changes. Therefore, enriching single-user web applications with shared editing capabilities is a costly and complex endeavor.

To reduce the effort to convert single-user editors to collaborative ones, we propose a Generic Collaboration Infrastructure (GCI) capable of transforming editors non-invasively, i.e. no source code changes are required. Incorporating shared editing capabilities into existing web applications requires to complete a GCI configuration as well as to embed a dedicated JavaScript file materializing the GCI logic. Once the GCI setup is finished, document changes are instantly synchronized and editing conflicts are automatically resolved. Besides the substantial reduction of development effort, the GCI approach promises a broad applicability due to its application-agnostic nature. To validate the viability of the GCI approach, we transformed four single-user web applications. Moreover, we conducted an extensive user study and the results show that users are generally satisfied working collaboratively with converted editors.

The main contributions of this paper are threefold:

1. We propose a transformation approach allowing to non-invasively incorporate shared editing capabilities in existing single-user web applications.
2. We report on a user study with 30 subjects assessing collaboration qualities in joint work scenarios leveraging two converted editors.
3. We carve out characteristics web applications have to expose in order to adopt the generic transformation approach.

The rest of this paper is organized as follows: Section 2 discusses the challenges devising a GCI and Section 3 elaborates on the GCI architecture as well as the transformation process. While Section 4 exposes insights about the conducted user study including evaluation results, Section 5 derives necessary GCI adoption criteria. Section 6 exhibits a related work discussion and eventually, Section 7 summarizes conclusions.

2. Challenges

Instead of implementing tailored collaboration extensions for single-user applications, we propose a GCI serving arbitrary standards-based web applications. Nevertheless, devising an application-agnostic GCI induces two major challenges:

1. The heterogeneity of Application Programming Interfaces (APIs) exposed by various editors.
2. The complexity of the conflict resolution scheme for numerous sets of editor operations.

The first challenge is due to the diversity of each and every web application. The collaboration services *document synchronization* and *conflict resolution* have to be attached to the editor implementation to capture document changes and to replay these document changes at all remote sites. For example, an editor document is changed if a character is entered or if the font size increased. To track and replay these document changes, developers have to identify the editor APIs which allow observing and applying changes. Consequently, programmers have to get familiar with the code base. After the familiarization step, developers additionally have to implement the actual capture and replay logic. Apparently, the familiarization and implementation are specific for each editor which leads to time-consuming and costly editor integrations.

The second challenge – the complexity of the conflict resolution scheme – is closely related to the mechanics of the operational transformation algorithm which represents the predominant concurrency control algorithm. Operational Transformation (OT) [5] has been introduced by Ellis and Gibbs in 1989 and in the last two decades the algorithm has been advanced to tackle consistency issues, to suit differing document structures (e.g. SGML [6]) as well as to support sophisticated concurrency control operations (e.g. undo [7], operation compression [8]). Besides this extensive research exploration, OT is also the prevalent concurrency control algorithm in the industry. The majority of advanced collaborative web applications such as Google Docs [9], Apache Wave [10], Etherpad [11], SAP Process Flow [12], etc., adopt some variation of the OT algorithm. Other concurrency control algorithms such as differential synchronization [13], causal tree models [14], or the class of commutative replicated data types [15,16] are not widely spread and thus, we focus on the OT algorithm.

To grasp the complexity entailed by the OT scheme, we introduce a simple example. Let's assume a text editor provides only a minimal set of operations: insert character and delete character. The operation insert character is expressed as $\text{ins}(x, i)$ where x denotes the character to insert and i the insertion index starting with 1. Correspondingly, the operation delete character is expressed as $\text{del}(i)$ where i denotes the deletion index also starting with 1. In the scenario depicted