Exploiting Single-User Web Applications for Shared Editing - A Generic Transformation Approach

Matthias Heinrich
SAP AG
SAP Research Dresden
matthias.heinrich@sap.com

Franz Lehmann
SAP AG
SAP Research Dresden
franz.lehmann@sap.com

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

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

ABSTRACT

In the light of the Web 2.0 movement, web-based collaboration tools such as Google Docs have become mainstream and in the meantime serve millions of users. Apart from established collaborative web applications, numerous web editors lack multi-user support even though they are suitable for collaborative work. Enhancing these single-user editors with shared editing capabilities is a costly endeavor since the implementation of a collaboration infrastructure (accommodating conflict resolution, document synchronization, etc.) is required. In this paper, we present a generic transformation approach capable of converting single-user web editors into multi-user editors. Since our approach only requires the configuration of a generic collaboration infrastructure (GCI), the effort to inject shared editing support is significantly reduced in contrast to conventional implementation approaches neglecting reuse. We also report on experimental results of a user study showing that converted editors meet user requirements with respect to software and collaboration qualities. Moreover, we define the characteristics that editors must adhere to in order to leverage the GCI.

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*

General Terms

Design

Keywords

Groupware, Shared Editing, Web Applications

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

1. INTRODUCTION

In 2011, Gutwin et al. stated that the "standard web browser is increasingly becoming a platform for delivering rich interactive applications" [17]. The rise of the web browser as application platform is associated to its unique capabilities such as instant application consumption, device-agnostic provisioning and ease of maintenance. However, the limitations of the web browser in areas such as protocol support, sandbox restrictions or browser API inconsistencies challenge web developers. Moreover, groupware applications enabling numerous users to jointly edit documents (e.g. Google Docs [16]) demand sophisticated features such as real-time synchronization, conflict resolution or workspace awareness. Thus, implementing web-based groupware systems (e.g. a word processor capable of shared editing or a graphics editor furnishing shared drawing) is a costly and complex endeavor. For instance, the real-time collaboration suite Apache Wave [14] primarily developed by Google encompasses more than 200 000 lines of Java code [26]. A second example demonstrating the enormous implementation effort is the web-based business process modeler called SAP Gravity [30]. SAP Gravity consists of an editor accounting for 180 000 lines of JavaScript code and a synchronization server adding another 100 000 lines of Java code.

One convenient means to lower the development effort for groupware systems is the *transparent adaptation* approach pioneered by Sun et al. [34]. Transparent adaptation advocates the idea of converting new or existing single-user applications into collaborative multi-user applications. The transformation process requires a collaboration adapter and a generic collaboration engine. While the collaboration adapter records, converts and transmits operations, the generic collaboration engine merges conflicting operations and synchronizes all document copies. The approach reduces development effort since the generic collaboration engine may be reused by arbitrary applications. Another approach tailored for web applications was proposed by Lowet and Goergen [23] which was especially suited for co-browsing scenarios. The task of synchronizing multiple views was tackled through an *output synchronization mechanism*, i.e. once the document object model (DOM) was modified through a UI event (e.g. mouse click) the DOM mutation event (e.g. DOM node removed) is recorded, distributed and replayed among all application instances. Since the output synchro-

nization mechanism relies exclusively on the standardized DOM Core [20] and DOM Events specification [32], the solution may be applied to arbitrary web applications.

Although the aforementioned solutions may speed up the time-consuming task of implementing a web-based groupware system, they exhibit a number of limitations. First, the presented transparent adaptation scheme [34] entails the implementation of a collaboration adapter. If the transparent adaptation approach is adopted naively i.e. the collaboration adapter leverages application-specific APIs, the reuse of the adapter is not permitted and a re-implementation for each application is required. Second, the output synchronization mechanism [23] lacks a conflict resolution scheme. Consequently, shared editing scenarios where multiple users edit the same document simultaneously are not supported.

To address these limitations, we propose a novel generic collaboration infrastructure (GCI) for web applications comprising a generic collaboration adapter and an operational transformation engine. While the generic collaboration adapter is capable of tracking DOM manipulations (e.g. create, delete, and modify DOM elements), the operational transformation engine provides a synchronization service as well as a conflict resolution mechanism based on the prevalent concurrency control algorithm called operational transformation (OT) [12]. Besides implementing the proposed GCI, we have validated the transformation of single-user applications into multi-user applications with two distinct web-based applications: the graphics editor SVG-edit [2] and the word processor CKEditor [7]. After transforming SVG-edit and CKEditor, the two collaborative editors were evaluated conducting an extensive user study.

The main contributions of this paper are three-fold:

- We propose a generic transformation approach for web applications capable of turning single-user editors into multi-user editors supporting shared editing.
- We report on a user study with 30 participants assessing collaboration qualities in shared editing and shared drawing scenarios leveraging the collaboration-enabled CKEditor and the converted SVG-edit editor.
- We carve out the limitations of the generic transformation approach and derive crucial characteristics of web applications that are required in order to apply the generic conversion scheme.

The rest of this paper is organized as follows: Section 2 provides motivating examples introducing two single-user editors suitable for shared editing scenarios. Section 3 elaborates on the challenges designing a generic transformation approach for web editors. Section 4 illustrates an architecture and an implementation realizing the generic transformation approach and the process of converting existing single-user editors into multi-user editors. Section 5 shows the results of the conducted user study evaluating the collaboration qualities of two converted editors. Section 6 discusses the limitations of the generic transformation approach and derives requirements for convertible web applications. Section 7 compares our work to the state-of-the-art and Section 8 exhibits conclusions as well as future work.

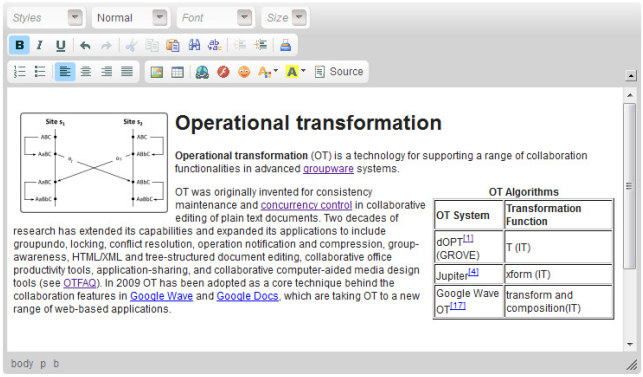

Figure 1: User interface of the single-user word processor CKEditor [7]

2. MOTIVATING EXAMPLE

Web-based groupware systems have proven to support geographically dispersed teams in a variety of tasks. For example, teammates can jointly create documents, spreadsheets or presentations leveraging web-based office suites such as Google Docs or Microsoft Office Web Apps. However, numerous web-based editors provide solely single-user support without shared editing capabilities. For instance, there are word processors (e.g. Adobe Buzzword, CKEditor), integrated development environments (e.g. Cloud9 IDE, Eclipse Orion) and graphics editors (e.g. Adobe Photoshop Express, SVG-edit) featuring only single-user support. Transforming these editors into collaborative multi-user editors could significantly broaden the application scope since converted editors enrich existing capabilities with shared editing support. Hence, multiple users may work simultaneously leveraging converted editors.

One prominent single-user editor is the open source project CKEditor [7] depicted in Figure 1. CKEditor is a word processor offering common features such as text formatting, image insertion or text alignment. In contrast to standalone office suites (e.g. Google Docs), CKEditor is primarily meant to be embedded in web pages. Since its first release in 2003, CKEditor has been widely adopted resulting in 3.5 million downloads [8]. The large CKEditor customer base consisting of enterprises, nonprofit organizations and individual users is also confronted with joint tasks where shared editing capabilities could speed up the task execution. However, introducing shared editing facilities in an established open source project with a JavaScript code base of more than 110 000 lines of code is not a trivial task. Amongst other things, the traditional development approach comprises the following tasks: (1) implementing a state-of-the-art conflict resolution scheme, (2) setting up a communication infrastructure to distribute synchronization messages and (3) intercepting as well as serializing user input in a dedicated messaging format. In Section 4 we demonstrate how the proposed generic collaboration approach can transform CKEditor into a collaborative multi-user editor. Additionally, we show in Section 5 that the transformed CKEditor meets user requirements with respect to accepted software qualities (e.g. usability, efficiency) and collaboration characteristics (e.g. coordination).

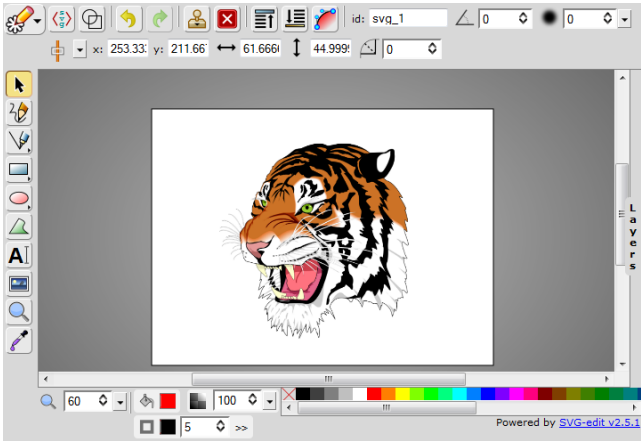

Figure 2: User interface of the single-user graphics editor SVG-edit [2]

Besides shared editing, shared drawing represents a second important collaboration domain. Therefore, we selected a graphics editor capable of producing scalable vector graphics (SVG). The single-user editor called SVG-edit [2] is another widespread open source project supported by a very active community (i.e. SVG-edit has more than 18 000 downloads of its current release version 2.51 and a development team of 17 committers and contributors). SVG-edit provides common graphics tools to accomplish typical drawing operations such as create lines, ellipses, rectangles, text, fill shapes or modify stroke styles. The tool palette as well as the canvas of the SVG-edit is shown in Figure 2. Like CKEditor, SVG-edit is also implemented in JavaScript accounting for approximately 30 000 lines of code. Again, supporting real-time collaboration in a project of this size is a demanding endeavor since the same implementation steps are required as for the conversion of the CKEditor. In the following section, we will present the challenges that emerge devising a GCI which allows to efficiently transform single-user editors into multi-user editors.

3. CHALLENGES

Reuse is one of the fundamental software engineering principles [18], since it increases software robustness, furnishes encapsulation and lowers initial development costs as well as maintenance costs. While devising a reusable collaboration infrastructure for the web, we identified the two main challenges that had to be tackled:

- the complexity of the conflict resolution scheme for large sets of editor operations and
- the heterogeneity of application programming interfaces (APIs) provided by various editors.

Currently, the predominant conflict resolution scheme for shared editing in real-time is denoted as operational transformation (OT). In 1989, Ellis and Gibbs proposed the first OT algorithm [12] and after two decades of research, OT has been advanced to support more sophisticated features including undo [29], operation compression [33] and HTML document support [10]. Moreover, OT has been incorporated in various industrial-strength products such as Google

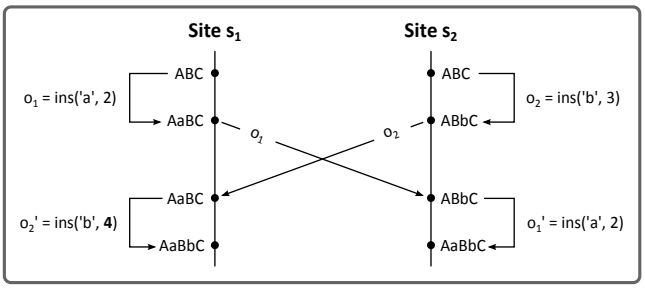

Figure 3: Conflict resolution example using the operational transformation algorithm

Docs [16] and SAP Gravity [30]. One key characteristic of OT is that each OT implementation is tailored to a predefined set of operations. Therefore, it is unsuitable for generic applicability since each editor employs its own set of operations. For example, while a word processor may require an operation to insert and delete a character, a graphics editor may demand a create shape as well as a remove shape operation.

To illustrate the link between the OT conflict resolution scheme and the number of editor operations, suppose the following example: a simple text editor exclusively supports the operation insert character $ins(c_i, x_j)$ where c_i denotes the character to insert and x_j the insertion index. In the scenario depicted in Figure 3 two users simultaneously insert a character in the document ABC at different index positions. After the local operations $o_1 = ins('a', 2)$, $o_2 = ins('b', 3)$ were applied to the local documents, operations o_1, o_2 are transmitted to the other site in order to reconcile document copies. Note that the naive replay of the local operations o_1, o_2 would result in two diverging documents containing $AabBC$ and $AaBbC$. The OT conflict resolution scheme transforms operations o_1, o_2 according to the function f

$$f(o_1, o_2) = \{o_1', o_2'\}$$

$$f(\text{ins}(c_1, x_1), \text{ins}(c_2, x_2)) = \begin{cases} x_1 > x_2 & \{\text{ins}(c_1, x_1 + 1), \text{ins}(c_2, x_2)\} \\ x_1 < x_2 & \{\text{ins}(c_1, x_1), \text{ins}(c_2, x_2 + 1)\} \\ x_1 = x_2 & \{\text{ins}(c_1, x_1), \text{ins}(c_2, x_2 + 1)\} \end{cases}$$

and generates o_1', o_2'. Executing the remote operations o_1', o_2' renders two documents both containing the identical string $AaBbC$. Thus, the conflict is successfully resolved. In this simple example, the transformation function f only has to respect the case where two insert operations are applied concurrently. If the simple text editor would also provide a delete operation, the function f had to consider 3 additional combinations of operations (s_1, s_2 delete concurrently; s_1 deletes, s_2 inserts; and s_1 inserts, s_2 deletes). Since the transformation function f has to consider all possible combinations of operations, the number of conflict resolution cases grows quadratically with the number of operations. For instance, a regular graphics editor offering 15 operations requires 225 conflict resolution cases. A reusable collaboration infrastructure has to encompass all operations of all supported editors which may lead to thousands of conflict resolution implementations. Creating such a complex system is neither feasible nor cost-effective.

Besides the complexity of the collaboration infrastructure injected by the multitude of editor operations, another challenge is heterogeneity of editor APIs. Editor APIs are required in order to record local operations and to replay remote operations. For example, to support the scenario illustrated in Figure 3, the simple text editor needs to provide an API capable of informing about insert notifications and allowing for character insertions. Thus, all local insert operations could be recorded and eventually be replayed. However, the task of adapting the collaboration infrastructure to various editor APIs is time-consuming since multiple editors may offer hundreds or even thousands of operations altogether. Not only is the editor-specific adapter implementation costly, but also the familiarization process with numerous APIs is tedious and demanding. This is particularly true for real-life examples such as the CKEditor with a code base of 110 000 lines or the SVG-edit exposing 30 000 lines of code.

4. GENERIC COLLABORATION INFRASTRUCTURE

In this section, we revisit the outlined challenges, describe the GCI architecture consisting of the generic collaboration adapter (GCA) as well as the operational transformation engine (OTE) and demonstrate the conversion of single-user editors into multi-user editors. Moreover, we describe the processes to record as well as to replay DOM manipulations and conclude with the presentation of our GCI implementation.

4.1 GCI Architecture

As stated in Section 3, the main challenges devising a GCI are the complexity of the conflict resolution scheme for the multitude of editor operations and the heterogeneity of editor APIs. Considering these challenges, we set out to create an application-agnostic collaboration infrastructure neglecting editor-specifics such as tailored operations or dedicated APIs. We identified the application-agnostic elements of the browser platform and carved out the abstract editor architecture as depicted in Figure 4.

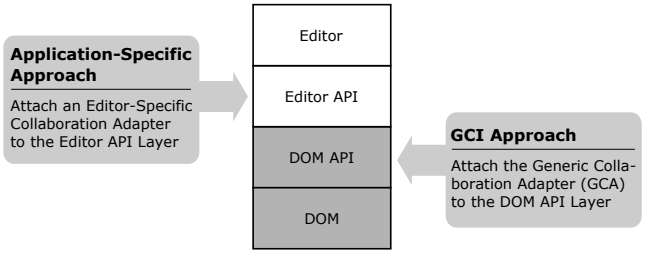

Figure 4: Abstract web editor architecture and options to attach the collaboration infrastructure

The architecture illustrates the basic building blocks which are available in all web editors (excluding editors built upon plug-in technologies such as Adobe Flash [4] or Microsoft Silverlight [24]). Figure 4 groups the editor-specific parts (Editor, Editor API) as well as the editor-agnostic parts (DOM API, DOM). In contrast to the naive approach linking the collaboration adapter to the specific editor API, we tackle the conversion of single-user editors into multi-user editors by attaching the GCA directly to the DOM API layer. Exploiting this DOM API layer is the key to overcoming the specified challenges. Since all editor-specific operations are eventually translated into a set of DOM operations, the GCI has to exclusively support DOM operations instead of numerous sets of editor-specific operations. For example, a graphics editor may provide a specific draw rectangle operation with the method signature drawRectangle(width, height) which translates into the standardized DOM operations (1) create *rect* element document.createElementNS('`http://www.w3.org/2000/svg`', 'rect'), (2) set the width attribute rect.setAttribute('width', 20) and (3) set the height attribute rect.setAttribute('height', 10). Hence, the focus on DOM operations limits the complexity of the conflict resolution scheme since the number of DOM operations is fixed and does not depend on the number of supported editors.

Besides limiting operations to the fixed DOM operation set, the approach of binding the GCA to the DOM API layer is also beneficial with respect to the second challenge: heterogeneity of editor APIs. As demonstrated before, all editor-specific manipulations (e.g. draw rectangle) can be accommodated through multiple DOM manipulations (e.g. create element and set attributes). Therefore, local operations transmitted to a remote site can be replayed using solely the DOM API. To record local editor operations, the standardized DOM Events API is exploited. The DOM Events API provides methods to observe all kinds of DOM manipulations. For example, event listeners can be attached to DOM nodes to react upon DOM node changes, DOM node insertions or DOM node removals. Consequently, model manipulations can be recorded and replayed on the DOM API level without having to consult the editor-specific APIs. Note that the limitations of the DOM-based approach are presented in Section 6.

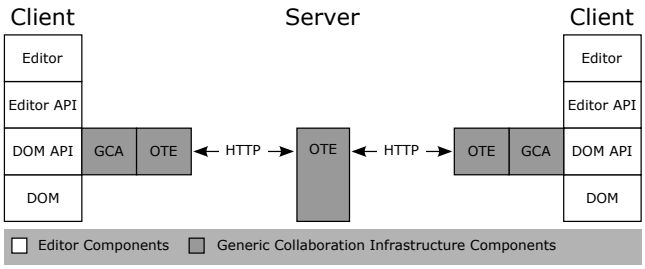

Figure 5: Overview of the Generic Collaboration Infrastructure (GCI) for web applications

The architecture of the GCI is shown in Figure 5. The white boxes denote editor components and the grey boxes are GCI components. While designing the GCI, we embraced a non-intrusive development approach, i.e. GCI components are clearly separated from the editor. The programmatic link between editor and GCI is exclusively realized through DOM interfaces (DOM Core [20] and DOM Events [32]). Thus, it is feasible to connect editors generically to the GCI since no editor-internal API is used. The central server depicted in Figure 5 represents the communication hub. Hence, clients are limited to interact with the origin server without peer-to-peer communication support. This network topology was selected due to the web security restriction called *same-origin policy* [13] which permits clients to reload data solely from the server where the application was retrieved from.

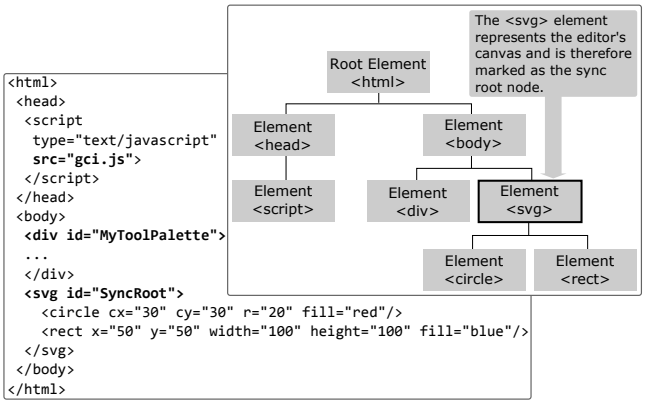

Figure 6: Example HTML page of a minimal collaboration-enabled web editor and the corresponding DOM tree

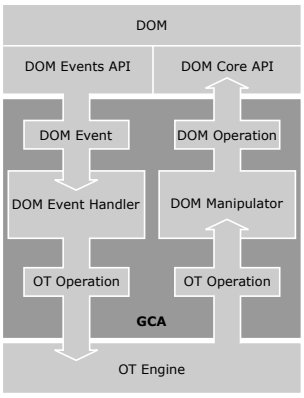

Figure 7: Client-side Generic Collaboration Adapter (GCA) capable of recording as well as replaying DOM manipulations

4.2 GCI Configuration

In order to integrate the GCI into an existing web editor, two configuration steps are required: (1) the client-side GCI implementation *gci.js* has to be included in the editor's HTML page and (2) a configuration file named *gciconfig.js* has to be adapted. Figure 6 shows an HTML page properly including the *gci.js*. Moreover, the HTML page in Figure 6 contains a minimal SVG editor consisting of a tool palette accommodated by the *div* element and an editor's canvas represented by the *svg* element. To support a selective synchronization behavior, the dedicated configuration file *gciconfig.js* allows to mark multiple DOM nodes as *sync root nodes* meaning that solely manipulations affecting these root nodes and their respective child nodes will be propagated to all clients.[1] For example, assuming that the *svg* element (cf. Figure 6) is configured as a sync root node, all modifications changing the *circle* or *rect* element would be distributed to all clients. Furthermore, child node manipulations such as adding or removing nodes would also be included in the synchronization process. However, selecting a new tool in the editor's palette encapsulated in the *div* element would not trigger synchronization since the *div* node is not a sub-node of the *svg* node.

4.3 DOM Synchronization Processes

After the GCI configuration has been completed, the adapted web editor can be loaded with an arbitrary web browser which triggers the GCI to switch to operation's mode. In the operation mode, the GCI is devoted to execute the record and replay DOM manipulations' processes. As shown in Figure 5, the two major components of the GCI are: (1) the generic collaboration adapter (GCA) and (2) the operational transformation engine (OTE). A detailed view of the GCA is shown in Figure 7 outlining also the steps which are executed to record and to replay DOM manipulations. For the former case, DOM changes are recorded leveraging the DOM Events API [32]. The *addEventListener* method is used to register event handlers on the sync root nodes. For example, the *svg* node in Figure 6 represents an appropriate sync root node since all graphics elements drawn onto the canvas are sub-nodes of the *svg* node. In contrast, the selected tool in the editor's palette, for instance, is not a sub-node of the *svg* node and therefore excluded from the synchronization. Once event listeners are registered on the sync root nodes, all DOM manipulations triggered by the editor are monitored by the DOM event handler component (cf. Figure 7). Currently, the following DOM mutation events are supported [32]:

- DOMNodeInserted: Fired when a node has been added as a child of another node.
- DOMNodeRemoved: Fired when a node is being removed from its parent node.
- DOMAttrModified: Fired after an attribute has been modified on a node.
- DOMCharacterDataModified: Fired after character data within a node has been modified but the node itself has not been inserted or deleted.

After one of the listed events is fired, the respective event handler starts constructing an OT operation leveraging data exposed by the event object (e.g. attributes of an inserted node). The OT operation is then passed to the interchangeable OT engine that has to support common tree operations such as insert or remove node. Consequently, DOM operations can be mapped to OT operations. OT engines capable of handling tree-structured documents are for example Apache Wave [14] or SAP Gravity [30]. Once the OT engine receives an OT operation from a remote site, the operation must be transformed against local concurrent operations in order to resolve potential conflicts. Eventually, the remote operation is integrated in the OT document. Afterwards, sync messages containing novel local operations, timestamps, etc., are sent to the server. The server integrates the OT operation into its local persistent OT document and broadcasts the sync message to all other clients to assure that they are aware of the changes and update their local document copies accordingly.

In addition to the record DOM events process, the GCA supports a second process capable of replaying remote DOM

[1] Currently, children of sync nodes cannot be excluded from the DOM synchronization. In the future, we plan to establish an XPath-like addressing scheme [9] to configure exceptions.

manipulations locally. As depicted in Figure 7, the sync messages distributed by the server are first processed by the OT engine in order to resolve conflicts and to update the local OT document. For example, in a shared editing session a conflict could occur if two users change the same string ABC (cf. Figure 3). While the local user changes the string to $AaBC$, the remote user edits the very same string resulting in $ABbC$. The local conflict resolution scheme processes the following steps: (1) the local operation $o_1 = ins('a', 2)$ is incorporated in the OT document, (2) the remote operation $o_2 = ins('b', 3)$ is received by the local OT engine, (3) the two concurrent OT operations o_1, o_2 are locally transformed against each other, (4) the resulting transformed operations $o_1' = ins('a', 2)$, $o_2' = ins('b', 4)$ are passed to the GCA that has to replay the appropriate transformed operation locally. Before the operation can be replayed, the DOM manipulator (cf. Figure 7) has to map the OT operation to an eligible DOM operation. Currently, the mapping leverages common DOM operations such as document.createTextNode(), node.removeChild() or node.setAttribute(). In the described example scenario, the OT operation $o_2' = ins('b', 4)$ would actually result in a *node.nodeValue = 'AaBbC'* DOM operation. Before DOM manipulations can ultimately be applied, the DOM manipulator component has to detach all event handlers registered by the GCA. Otherwise, the DOM manipulation triggers a DOM mutation event that is handled by the GCA recorder even though the DOM change was not caused by the editor. After detaching event handlers and applying DOM operations, the event handlers are attached once again and the remote operation is successfully integrated in the editor model.

4.4 GCI Implementation

The described DOM manipulation processes have been realized in a dedicated GCI prototype that implements the architecture depicted in Figure 5 with all associated components (OTE, GCA). This GCI implementation allowed us to successfully transform the illustrated SVG-edit and CKEditor as well as the spreadsheet application jQuerySheet [1] and the text editor TinyMCE [3]. In this paper, jQuerySheet and TineMCE were not presented in detail since both editors were not part of the evaluation. Furthermore, the GCI implementation is based on the SAP Gravity [30] OT engine. However, first tests showed that the OT engine incorporated in Apache Wave [14] is also a viable solution. Adopting Apache Wave instead of SAP Gravity entails modest GCA implementation changes. The resulting collaborative editors are demonstrated on our GCI demo page `http://vsr.informatik.tu-chemnitz.de/demo/GCI/`. Currently, the demos [19] exhibit the collaboration enabled SVG-edit as well as the CKEditor.

5. EVALUATION

The GCI described in the last section was adopted to convert the presented graphics editor SVG-edit and the word processor CKEditor. In this section, we report on the results of a user study analyzing the software and collaboration qualities of transformed editors.

The user study was carried out employing the usability testing technique [11] with 30 participants that had to solve two collaborative tasks in teams of two. While the first collaborative assignment aimed to exploit the shared drawing capabilities of the converted graphics editor SVG-edit, the second team assignment encompassed a shared editing scenario using the transformed CKEditor. In this user study, students studying computer science or a related subject took part.

Participants of the evaluation started out with the shared drawing scenario that was directly succeeded by the shared editing scenario. Both evaluation parts followed a fixed evaluation schedule comprising (1) a 5 min editor tutorial, (2) a 15 min shared editing session and (3) a 10 min questionnaire completion phase. First, subjects taking part in the usability testing had to learn the editor's capabilities in a 5 min training session. In the training session participants had to go through 10 small, non-collaborative exercises. Detailed instructions illustrated the steps to master the exercises. For example, the SVG-edit training session included exercises like create, move or fill shapes. After participants had become familiar with the editor, teams of two were formed. Members of a team were located in the same room having its own office desk equipped with a standard PC. Since desks were separated by a room divider, participants could speak to each other but they could neither see each other nor see each other's screens. To accomplish collaborative tasks, each team had a limited 15 min time slot. In the shared drawing session, the teams had to draw the floor plan of the evaluation room as detailed as possible. Moreover, they had to sketch the existing inventory as well as new inventory items that could improve the work atmosphere (e.g. plants). In the second shared editing exercise, participants had to author a document using the collaborative CKEditor. The document should accommodate the inventory list of the evaluation office associated with typical product characteristics such as purchase price, maintenance costs, life span, etc. Besides listing existing and proposed inventory items, team members had to write a letter to the office manager explaining why suggested inventory items were indispensable and should therefore be purchased. After finishing the shared editing assignments, each participant had to complete the questionnaire depicted in Figure 8. This questionnaire was created taking into account multiple software metrics tailored for our use case of evaluating collaborative editors.

In order to conduct a thorough user study allowing for resilient conclusions, we set out to identify relevant metrics for the evaluation of multi-user editors. On the one hand, converted editors represent regular software tools that have to be assessed against general software qualities such as functionality, usability or reliability. On the other hand, the user study is a means to evaluate the collaboration quality of converted editors. The collaboration characteristics are emphasized since they validate the usage of the GCI to transform single-user editors into multi-user editors. Based on these considerations, we selected two established software metrics: (1) the ISO/IEC TR 9126 standard for product quality [21] exposing various evaluation criteria for arbitrary software products and the groupware-specific *mechanics of collaboration* catalog [28].

The former ISO/IEC standard defines the following software quality characteristics: (1) functionality, (2) reliability, (3) usability, (4) efficiency, (5) maintainability and (6) portability. According to Bevan [5], assessing software quality from the end user's point of view encompasses mainly characteristics (1) to (4). Since the conducted user study addressed the end user, we reflected the four ISO/IEC product quality characteristics in questions 1–16 (cf. Figure 8).

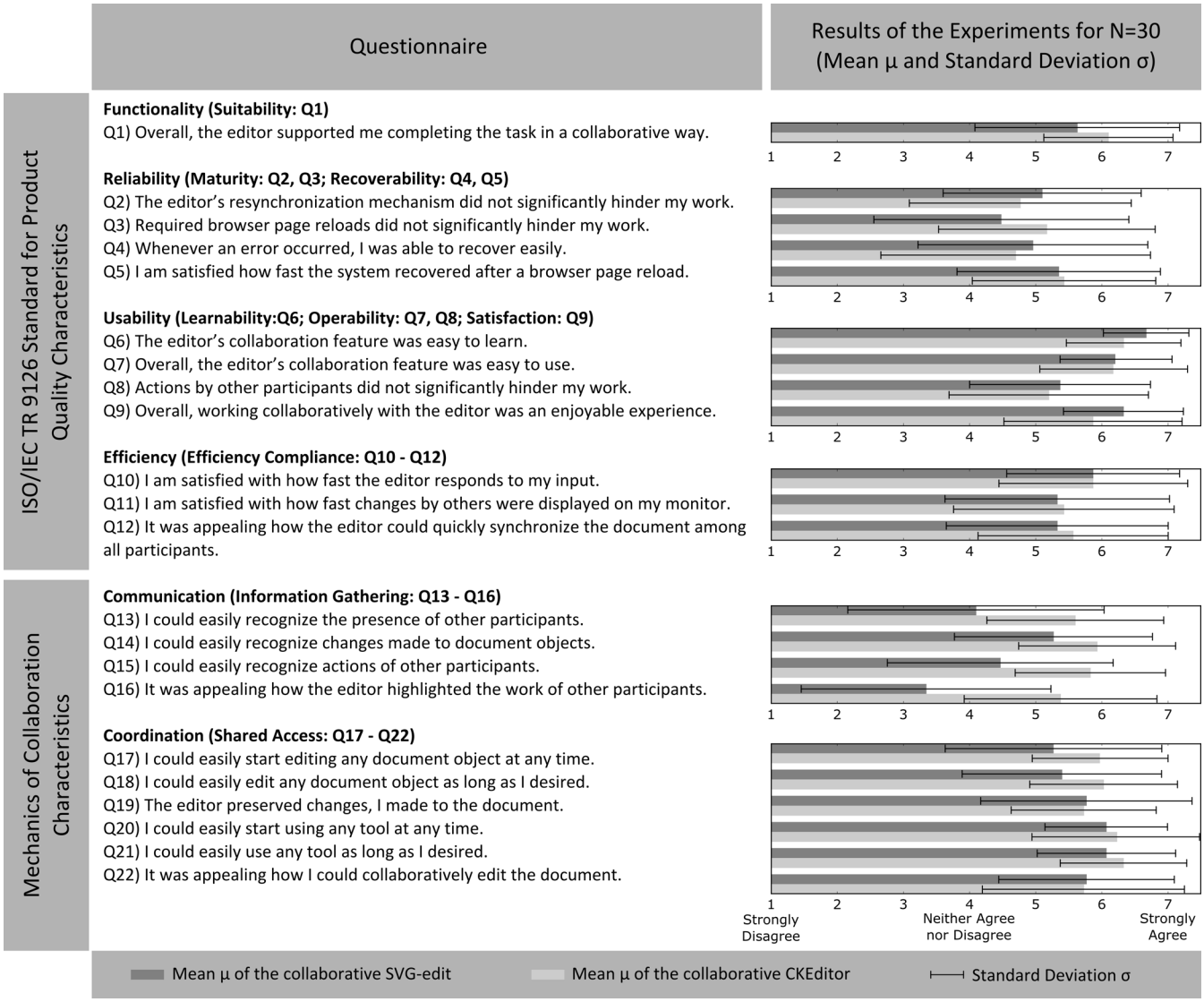

Figure 8: Questionnaire and the corresponding evaluation results

The ISO/IEC standard [21] specifies only the characteristics without providing standardized questions and therefore, specific questions tailored to assess functionality, reliability, usability and efficiency for collaborative editors had to be authored. In addition to incorporate a validated and widely accepted standard, we also included software metrics denoted as mechanics of collaboration (MoC). The MoC respects communication aspects (e.g. workspace awareness) as well as coordination aspects (e.g. shared access to tools, objects, etc.). Questions 13–22 (cf. Figure 8) address these particular MoC quality characteristics.

As illustrated in the evaluation schedule, participants of the user study had to complete the questionnaire in Figure 8 directly after the collaborative work sessions. To respond to the listed questions, they had to choose one answer from a balanced seven-level Likert scale [22]. The Likert scale offers seven possible answers: (1) strongly disagree, (2) disagree, (3) disagree somewhat, (4) neither agree nor disagree, (5) agree somewhat, (6) agree and (7) strongly agree. The results of the experiments are depicted in the bar charts in Figure 8 right next to the questions. The 30 collected responses were utilized to calculate the mean μ and standard deviation σ. While the mean μ is depicted by grey bars, the standard deviation σ is visualized using black error bars. Besides μ and σ, another interesting distribution aspect is exposed through the lower limit of the standard deviation σ. If the lower limit of the standard deviation is above 4 (i.e. $(\mu - \sigma) > 4$) and assuming the experiment follows a normal distribution, at least 84 % of all participants responded positively to the respective question. This means that the large majority agrees with the given question.

Analyzing the results regarding the ISO/IEC software quality characteristics (cf. Q1–Q12), we draw the following conclusions. In general, the results of the experiment demonstrate that converted editors (SVG-edit, CKEditor) decently support collaborative work (cf. Q1: $\mu_{svg} = 5.63$, $\mu_{ck} = 6.10$). Usability aspects (cf. Q6–Q9) were also constantly assessed with a rating of $\mu \geq 5.20$ and $\sigma \leq 1.52$, mean-

ing that the majority was able to easily learn and use collaboration features. The results of the efficiency category (cf. Q10–Q12: $\mu \geq 5.33$) again allow to conclude that respondents are satisfied with the editors' performance. However, usage peaks during the evaluation sessions occasionally rendered the browser unresponsive due to high processor loads. Therefore, the editor application had to trigger browser page reloads, resynchronization or error handling mechanisms. Hence, the rating for the reliability category ($4.48 \leq \mu \leq 5.43$) shows that the exception handling leaves room for improvement.

The second group of results addresses the collaboration qualities (cf. Q13–Q22). First, the communication category embracing workspace awareness and information gathering shows ambivalent results. While the communication category regarding the collaborative SVG-edit shows poor ratings ($3.35 \leq \mu \leq 5.27$), the respective CKEditor ratings exhibit good results ($5.38 \leq \mu \leq 5.93$). The divergence of results is linked to the awareness capabilities of the GCI. Currently, the GCI includes only awareness facilities for text editors (e.g. highlight text authored by remote participants) but lacks support for graphical editors. As outlined in the future work section, we plan to revise and enrich the workspace awareness features in future GCI releases. The last group of questions Q17–Q22 evaluating the coordination characteristics proves that converted editors support real unconstrained collaboration. The findings expose a high rating $5.27 \leq \mu \leq 6.33$ showing that users appreciate how they can freely access document objects and editor tools.

6. DISCUSSION

In this section, we discuss the limitations of the generic transformation approach and present crucial properties that a web editor must exhibit in order to allow a conversion into a collaborative editor with shared editing capabilities. From our experiences gathered over the last 12 months, the following limitations could be noted:

External Data Model: Analyzing various web-based applications, we observed that editors targeting large documents (e.g. multi-page office documents) often separated the application model from the view model. This design decision is motivated by performance considerations. Having a multi-page document comprising thousands of lines represented in a DOM can easily consume more than 100 MB of RAM. In contrast, creating a specific application model using a tailored JavaScript implementation can efficiently compress large size documents. Then, the DOM is used as a view model only reflecting an excerpt of the application model. In this case, the GCI is not able to access the model through the standardized DOM API and hence, the GCI cannot be properly bound to the application model. Consequently, the GCI does not operate correctly. This limitation can be fixed with a moderate implementation effort connecting the GCI directly to the specific application model API.

Scattered Application State: Besides maintaining a complete external data model, there are also applications where parts of the applications' state reside in a separate JavaScript data structure. For example a *select all* operation capable of highlighting all shapes on the canvas of a graphics' editor might be realized using a JavaScript array where all object references to created shapes are preserved. Assuming participant 1 inserts a new rectangle, the DOM is immediately synchronized and participant 2 can instantly see the novel rectangle. However, the JavaScript array holding all shape references is only updated locally since the sync mechanism exclusively encompasses DOM objects. The erroneous application behavior is disclosed once participant 2 carries out the *select all* function and notes that the newly inserted rectangle is not highlighted. Fixing this limitation can be tedious since there might be a multitude of individual JavaScript data structures representing parts of the application state.

Naming Collisions: Converting single-user editors into multi-user editors exposed naming conflicts. Some editors manage their content identifiers by means of an incremented integer. If two users simultaneously create a content object, both application instances assign the very same integer identifier to two independent objects and therefore, the uniqueness of identifiers is no longer ensured. However, this limitation can typically be addressed through a lightweight change of the editor implementation.

Plug-in Technologies: As noted in Section 4, the foundation of our approach is the assumption that all editors comply with the abstract editor architecture depicted in Figure 4. Plug-in technologies such as Adobe Flash [4] or Microsoft Silverlight [24] do not adhere to this architecture. Instead of leveraging the DOM as a representation of the application model, plug-in-based web applications use the DOM only as a container to embed the plug-in frame. User interactions within the plug-in frame cannot be monitored by the GCI because they bypass the DOM APIs. Hence, editors built upon plug-in technologies are not supported by the GCI. Only a complete reimplementation based on DOM-standards would allow to adopt the GCI.

Lack of Application Model Isolation: In Figure 6, the application model was clearly isolated below the *svg* sub-tree from the rest of the DOM nodes (e.g. tools palette). Therefore, the GCI could easily be configured to synchronize solely manipulations addressing the *svg* node and its child nodes. However, this separation also denoted as the tools and material approach [31] is not always implemented throughout the web application. In particular, personalized views (e.g. selection highlighting) are occasionally interwoven with the application model and thus, are eventually displayed among all participants. This limitation does not affect the GCI operations, but rather impairs the communication and coordination quality of the shared editing sessions.

High Load of DOM Events and Operations: Certain editor operations affect numerous DOM nodes and produce vast sets of DOM events. All events are processed by the GCI and will in the end be reflected in DOM replay operations. In rare cases, these high loads can crucially impair the performance and responsiveness of all application instances. Examples for high load scenarios are: (1) fade animations or drag shape operations (produced up to 150 DOMAttrModified events per second), (2) copy and paste operations involving numerous objects (trigger multiple DOMNodeRemoved and DOMNodeInserted events), and (3) format operations addressing various DOM nodes (e.g. change the fill color of 100 table cells). This issue could be tackled by installing an operation composer capable of reducing the number of operations.

In essence, single-user editors that are eligible for the proposed generic transformation approach have to expose the following characteristics. They are (1) entirely based upon DOM standards without leveraging plug-in technolo-

gies, (2) are designed for multi-user readiness (e.g. identifier management does not break in multi-user scenarios), and (3) feature an application model that is represented by the DOM.

7. RELATED WORK

In this section, our work is put in context of existing collaboration frameworks and approaches that are commonly used to drive development efficiency implementing interactive groupware systems.

BeWeeVee [6] is a commercial, OT-based framework targeting desktop applications as well as web applications. The framework automatically keeps track of the operation history and thus, enables features like endless undo and the playback of arbitrary document changes. BeWeeVee can operate in client-server as well as in peer-to-peer mode. The support for web applications depends upon the Microsoft Silverlight plug-in [24] which introduces availability, compatibility and security issues.

Open Cooperative Web Framework (OpenCoWeb) [27] is an open source project that provides a JavaScript operation engine using OT to resolve conflicting, simultaneous changes. OpenCoWeb is especially tailored for web applications. While the framework is suited for new development projects, existing web editors can only profit from OpenCoWeb if large parts of the implementation are going to be rewritten.

MobWrite [25] also facilitates real-time synchronization and collaboration services. It is based on the differential synchronization algorithm [15] instead of the established OT algorithm. The MobWrite system is devoted to synchronize the content of HTML form elements. To integrate MobWrite into an existing single-user application, only one specific JavaScript library has to be added to the HTML file without having to change the application code. MobWrite easily extends existing web applications with form synchronization capabilities. Nevertheless, the framework supports only a small fraction of potential collaborative applications.

Apache Wave [14], formerly known as Google Wave, is a platform for shared editing applications. All application components have to be implemented in Java and eventually, the client component is compiled to JavaScript using the Google Web Toolkit compiler. The synchronization mechanism is also OT-based and offers an operation composer to reduce the number of exchanged operations. Even so Apache Wave offers a rich platform; a major drawback is the dependency to the Google Web Toolkit development methodology that is not compatible with differing approaches (e.g. Java Servlet or PHP projects).

In addition to the introduced frameworks primarily targeting the development of novel multi-user applications, there are also approaches supporting the transformation of existing single-user applications into collaborative multi-user applications.

Transparent adaptation was proposed by Sun et al. "to convert existing single-user applications into collaborative ones, without changing the source code of the original application" [34]. The main idea is to reuse a generic OT-based collaboration engine for synchronous real-time collaboration and to furthermore leverage a collaborative adapter to map application-specific operations to primitive OT operations. Hence, the adapter bridges the gap between the existing single-user application and the generic framework without requiring changes to the original application source code. The major development efforts induced by the transparent adaptation approach emerge from the implementation of the collaborative adapter. If the adapter implementation relies on an application-specific API, the adapter is exclusively dedicated to this particular application and reuse is not permitted. For example, CoWord [34] or CoPowerPoint [34] incorporate specific adapter. The proposed GCI approach anchors its generic collaboration adapter (GCA) at the standards-based DOM APIs [20, 32] which are used by a myriad of web applications and therefore the GCA can be reused for those web applications. Application specifics (e.g. which part of the application should be synchronized) are entirely captured in a configuration file. Therefore, the adaptation cost to convert existing single-user web applications is reduced in contrast to devising an individual application-specific adapter for each web application.

The output synchronization mechanism was devised by Lowet and Goergen [23] and the authors propose to enrich existing single-user web applications with co-browsing capabilities. A reference browser records all DOM manipulations and sends them to all client browsers which replay the DOM manipulations. Distinguishing between the *reference browser* and *client browsers* is required, since only the reference browser is allowed to distribute DOM manipulations. Thus, conflicts cannot occur but the solution is not suitable for shared editing scenarios.

In summary, none of the existing approaches offers "out-of-the-box" collaboration support to the extent of the presented GCI approach. In particular, our approach is unique with respect to the GCI configuration strategy incurring minimal conversion effort. Moreover, in contrast to domain-specific solutions (e.g. HTML form synchronization), the GCI approach has a broad application domain supporting numerous single-user web applications.

8. CONCLUSION

Web 2.0 technologies paved the way for complex interactive applications like word processors, graphics editors and integrated development environments. Although collaborative multi-user tools like Google Docs are well established and adopted by millions of web users, the majority of web applications only facilitates single-user support. Adding shared editing capabilities to existing single-user applications requires complex concurrency control systems which are costly to implement, test and maintain.

In this paper, we introduced an approach capable of transforming existing single-user web applications into collaborative multi-user applications. Consequently, the reuse of existing application functionality and complex collaboration functionality is promoted and the development effort for multi-user applications is crucially reduced. Furthermore, users can collaborate using applications they are familiar with.

To achieve this, we proposed a generic collaboration infrastructure consisting of an operational transformation engine for concurrency control and a generic collaboration adapter to couple the OT engine with existing web applications. A generic collaboration adapter renders the approach to be application-agnostic since neither changes to the application source code are required nor an application-specific adapter has to be implemented. Nevertheless, tailoring the generic adapter to the DOM API layer imposes limitations to its

applicability. For example, applications based on browser plug-in technologies are not supported.

To demonstrate the feasibility of our generic transformation approach, two existing web applications have been transformed into multi-user applications with shared editing capabilities. Both editors were evaluated with respect to software and collaboration qualities. The results of an extensive user study showed that the transformed editors decently support collaborative work. In particular, characteristics like functionality, usability, efficiency and coordination were constantly assessed with high ratings meaning that users were satisfied with the offered support. However, especially the reliability and awareness aspects leave room for improvement.

The conducted user study has revealed the importance of awareness features in shared editing sessions. Therefore, in future GCI releases, the workspace awareness will be revised and extended. In addition, the GCI will be extended by an operation composer to reduce the number of synchronization messages and eventually to improve the GCI performance.

9. ACKNOWLEDGMENTS

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

10. REFERENCES

[1] jQuery.sheet - The web based spreadsheet. http://code.google.com/p/jquerysheet/, 2011.

[2] SVG-edit - A complete vector graphics editor in the browser. http://code.google.com/p/svg-edit/, 2011.

[3] TinyMCE - Javascript WYSIWYG Editor. http://www.tinymce.com/, 2011.

[4] Adobe. Adobe Flash Platform. http://www.adobe.com/flashplatform/, 2011.

[5] N. Bevan. Quality in use: Meeting user needs for quality. *Journal of Systems and Software*, 49(1):89–96, 1999.

[6] BeWeeVee. BeWeeVee - Life collaboration framework. http://www.beweevee.com, 2011.

[7] CKSource. CKEditor - WYSIWYG Text and HTML Editor for the Web. http://ckeditor.com/, 2011.

[8] CKSource. What is CKEditor? http://ckeditor.com/what-is-ckeditor, 2011.

[9] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath/, 1999.

[10] A. H. Davis, C. Sun, and J. Lu. Generalizing operational transformation to the standard general markup language. In *CSCW*, pages 58–67, 2002.

[11] J. S. Dumas and J. C. Redish. *A Practical Guide to Usability Testing*. Intellect Ltd, 1999.

[12] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, SIGMOD '89, pages 399–407, New York, NY, USA, 1989. ACM.

[13] D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, 6th edition edition, 2011.

[14] A. S. Foundation. Apache Wave. http://incubator.apache.org/wave/, 2011.

[15] N. Fraser. Differential synchronization. In *ACM Symposium on Document Engineering*, pages 13–20, 2009.

[16] Google. Google Docs - Create and share your work online. http://docs.google.com/, 2011.

[17] C. Gutwin, M. Lippold, and T. C. N. Graham. Real-time groupware in the browser: testing the performance of web-based networking. In *CSCW*, pages 167–176, 2011.

[18] G. T. Heineman and B. Councill. *Component-Based Software Engineering: Putting the Pieces Together*. Addison-Wesley Professional, 2001.

[19] M. Heinrich. GCI Demo Page. http://vsr.informatik.tu-chemnitz.de/demo/GCI/, 2011.

[20] 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.

[21] ISO/IEC. *ISO/IEC 9126-1: Software engineering - Product quality - Part 1: Quality model*. 2001.

[22] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):5–55, 1932.

[23] D. Lowet and D. Goergen. Co-browsing dynamic web pages. In *WWW*, pages 941–950, 2009.

[24] Microsoft. Microsoft Silverlight. http://www.microsoft.com/silverlight/, 2011.

[25] MobWrite. google-mobwrite - Real-time Synchronization and Collaboration Service. http://code.google.com/p/google-mobwrite/, 2011.

[26] A. North. Wave open source next steps: "Wave in a Box". http://googlewavedev.blogspot.com/2010/09/wave-open-source-next-steps-wave-in-box.html, 2010.

[27] OpenCoWeb. Open Cooperative Web Framework - Project intro. http://opencoweb.org, 2011.

[28] D. Pinelle, C. Gutwin, and S. Greenberg. Task analysis for groupware usability evaluation: Modeling shared-workspace tasks with the mechanics of collaboration. *ACM Trans. Comput.-Hum. Interact.*, 10(4):281–311, 2003.

[29] A. Prakash and M. J. Knister. A Framework for Undoing Actions in Collaborative Systems. *ACM Trans. Comput.-Hum. Interact.*, 1(4):295–330, 1994.

[30] A. Rickayzen. Simple way to model processes in the Web. http://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/25360, 2011.

[31] D. Riehle and H. Züllighoven. *A pattern language for tool construction and integration based on the tools and materials metaphor*. ACM Press/Addison-Wesley Publishing Co., 1995.

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

[33] H. Shen and C. Sun. Flexible notification for collaborative systems. In *CSCW*, pages 77–86, 2002.

[34] 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, December 2006.