Shared Editing on the Web: A Classification of Developer Support Libraries

István Koren and Andreas Guth and Ralf Klamma
Advanced Community Information Systems (ACIS)
RWTH Aachen University
Informatik 5, Ahornstr. 55, 52056 Aachen Germany
{koren|guth|klamma}@dbis.rwth-aachen.de

***Abstract*—Together with the current shift to cloud-based solutions, various Web applications have been enriched with collaborative features. These collaborative features enable users to work together on digital products like documents, diagrams and videos at the same time on a global scale. Implementing them require developers to have knowledge about both complex algorithms for maintaining consistency on one hand and usability issues on the other hand. Besides developing apps from scratch, Web application developers often meet these challenges by employing ready-made libraries for shared editing on the Web. A new generation of these shared editing frameworks has emerged recently; several of these libraries are available as open source solutions. In this paper, we first present general and browser-specific requirements for shared editing like consistency algorithms and means of workspace awareness. Then, state-of-the-art frameworks for shared editing are analyzed in respect of their support of these requirements. As a contribution, we have identified missing features like the availability of general awareness widgets and new architectural designs due to emerging Web standards. To that end, we demonstrate prototypes addressing some of these issues.**

***Keywords*—CSCW; Shared Editing; Web Applications; Real-time Collaboration; XMPP; Widgets**

I. Introduction

An increasing number of Web applications with collaborative features are presented nowadays. Applications like Google Docs[1] allow users to create documents in the cloud, share them with others and edit artifacts concurrently within the browser. Users are able to see the cursor position of their collaborators in real-time and even chats are possible within the scope of the web application. Most of these features have to be developed from scratch which involves solving several technological challenges.

The underlying foundation for all kinds of Web applications are HTTP servers delivering pages written in the *Hypertext Markup Language (HTML)* and made interactive by means of the JavaScript scripting language. The field of collaborative Web applications in particular was made possible by recent advances in web technologies including server-push technologies such as Ajax and Comet. Mostly long-polling techniques are used to get updates from other users through the server. These techniques pushed the limits of the traditional request/reply mechanism in the Web to enable near real-time interaction between clients and servers. With HTML5, emerging standards like WebSockets are introduced for stable bidirectional client-server connections that reduce the overhead of long-polling. Elaborate and easy-to-use JavaScript APIs enable developers to easily push and receive updates to and from the server. In addition, new standards within the HTML5 family such as WebRTC represent a paradigm shift in the Web. Peer-to-peer computing is now possible within the browser leading to new opportunities in architectural design. After an initial signaling phase to establish a channel between two peers, Web applications are now able to reach out to other instances running in browsers on other devices. without having to steer the communication over a server.

A. Motivation

While first, static Web pages were accessible by browsers on desktop computers, the mobile revolution now enables a variety of smartphones to be used for surfing the Web. Multiple cross-platform solutions like PhoneGap[2] and Sencha Touch[3] have been introduced to overcome the problem of having to develop specific native applications for each target platform. They allow for developing mobile apps with HTML5 and JavaScript and then exporting them as native applications to particular platforms. Additionally, new mobile operating systems such as Firefox OS and Tizen are being presented that allow to directly run packaged HTML apps that behave like native applications.

Shared editing systems built with JavaScript leverage this focus on HTML5 based apps by providing collaborative features for the Web. The biggest challenge in shared editing systems is identifying conflicts in edits by possibly remote users. Underlying consistency algorithms ensure that no user overwrites the inputs of others and that changes in the document are stored and synchronized securely without losing any data. As these algorithms are complex and error-prone, developers often consider using ready-made frameworks within their applications that ensure conflict resolution and update propagation to collaborators. They leverage recent developments in the Web such as bidirectional client/server communication over Web 2.0 technologies like Ajax. Several JavaScript libraries presented later are available as open source solutions to support developers in creating shared editing applications for the Web.

However, not only the consistency algorithms require special knowledge of the developers. Also a feature called workspace awareness, i.e. following the updates of remote

[1] http://drive.google.com

[2] http://phonegap.com

[3] http://www.sencha.com/products/touch

COLLABORATECOM 2013, October 20-23, Austin, United States

DOI 10.4108/icst.collaboratecom.2013.254097

collaborators is essential as a flawed realization can be especially confusing to users, e.g. if the user interface is constantly blinking for highlighting remote edits. The Google Wave project has shown how fulsome collaborative features may decrease the usability in the end. Presented to the public in 2009, Google planned to turn Wave into a system that would be an alternative to email. As not enough users adapted to the concepts of Wave, the work was handed over by Google in 2010 to the open source community as Apache Wave [1].

Therefore ensuring consistency and providing support for workspace awareness are crucial requirements for shared editing frameworks on the Web. Both are general aspects that are difficult to develop. Google Wave took two years to develop before it was opened publicly [2]. On the other hand, the implementation of CoWord [3], an elaborate work that enables real-time shared editing in Microsoft Office applications, took three man-years; though after the general collaboration engine was established, the implementation of CoPowerpoint building on the same infrastructure took only six man-months. Thus we consider consistency and workspace awareness as ideal candidates to be provided in the form of a framework.

B. Contributions

To remedy the current situation in the Web, this paper analyzes the current state of collaborative editing including the basic concepts like architectural models, the algorithms for keeping data consistent across devices and usability issues of making users aware of remote changes. This first takes into consideration approaches that were not specifically targeted for the Web. The requirements are then matched to the features of JavaScript frameworks for shared editing on the Web. Hereby we identify several functionality lacks. The underlying APIs of the frameworks and data models are demonstrated for showcasing their complexity of being employed in developments. This survey includes open source solutions that can be easily integrated into custom software such as research prototypes. We do not focus on the correctness of the underlying consistency algorithms.

As an overall result, we conclude indications for missing features in shared editing frameworks on the Web. Furthermore opportunities for these frameworks arise out of emerging standards such as WebSockets for resource-efficient client/server communication and WebRTC for direct peer-to-peer communication from browser to browser. We show how standardized Web widgets could resolve current issues of workspace awareness by providing awareness tools as user interface widgets. Finally, possible peer-to-peer architectures are showcased.

The rest of the paper is structured as follows. First, the related work section analyzes existing approaches of building general shared editing frameworks for native applications and the Web in special. Then, the underlying Web technologies used later are introduced. Open source shared editing frameworks for the Web are depicted thereafter, before missing features are identified. The paper concludes with possible solutions to overcome the issues.

II. Related Work

The history of developer support for shared editing system began with GroupKit, a framework for building collaborative applications based on a client/server approach [4]. The system proposed consists of a toolkit written in C++ that connects to a registrar running as a centralized instance. The client workstations run session manager processes that are responsible for creating, joining or leaving Conference objects. Conference applications are invoked by the session manager and display certain tools like shared editing widgets. Events are distributed via multicast remote procedure calls (RPC). Developers can create custom events, distribute them and listen to these RPCs calls to change the data model and user interface accordingly. Groupware widgets are available that provide general awareness tools like a participant list or telepointers. The toolkit is only available for applications developed with the interpreted Tcl language and Tk interface toolkit, no binding exists for the Web and no concept exists for variably sized user interfaces. The authors explicitely exclude direct multimedia communications.

Another approach considers collaboration as an aspect-oriented feature. The CoWord approach by Sun et al. introduces the general idea of a transparent adaptation (TA) approach that converts existing single-user applications into collaborative ones by inserting a layer that promotes and applies local changes to remote instances of the same software [3]. CoWord and CoPowerPoint are presented as evaluation prototypes that use the Microsoft Word and PowerPoint APIs to listen to document changes and promote them remotely. To manage consistency, the collaboration infrastructure uses operational transformation (OT). Telepointers for distributing the mouse position and radar views to see the scroll position of collaborators are available as workspace awareness tools.

Recent work of Heinrich et al. has brought the idea of transparent adaptation to the Web [5]. Based on document object model (DOM) based replication, i.e. listening to events concerning the insertion, editing or deletion of nodes, the document state is distributed to remote instances. However, they state that web applications that are "structured according to the established Model-View-Controller (MFC) pattern" are not transformable since they break the underlying synchronization mechanism. Besides demonstrating their approach with HTML based rich-text editors and SVG based drawing apps that are DOM based per definition, they analyzed 12 web applications from which only 6 could leverage the automatic collaboration transformation and usage of their *General Collaboration Infrastructure (GCI)* [6]. Therefore, still 50% of applications must be transformed by hand. The work was extended with general workspace awareness capabilities in further work [7]. They capture information about user interaction at the DOM level and propagate it over a central server. Participant lists and text highlighting are implemented as reusable workspace awareness widgets. Both solutions are closed-source commercial products thus not available to the general public.

The WatchMyPhone framework by Bendel et al. [8] provides a set of user interface widgets for collaborative and general awareness features for the Android mobile operating system. Therefore, the work derives from the original UI classes and transparently adds shared editing capabilities, so that developers leveraging the framework only need to exchange the concrete class they are using. To overcome mobile network outages, operation queues are built before they are sent to the server component. The framework is built on the CEFX framework [9] for consistency management. As the

target is limited to Android, it is out of scope for this paper.

Apache Wave as shortly outlined above is the successor of the discontinued Google Wave project and available as a server-based open source solution called "Wave in a Box" within the Apache incubator. The idea behind Wave is the inclusion of many different media into rich text documents within the browser. The documents called "waves" can be best imagined as e-mails in the traditional sense, though these documents remain editable even after they are sent to recipients. Two extension points are available as add-in solutions to enhance the functionality, namely robots and gadgets. Robots are server-side tools that can be added to a wave and perform automated tasks such as translating content written in a foreign language. On the other hand gadgets enhance the user interface of a wave with widget-like elments.

The related works described above specify concepts that can be used by developers to develop and provide applications to support shared editing. In the following, two available collaboration suites are presented to describe the current state-of-the-art in shared editing on the Web.

A. Collaborative Web Applications

Prominent representatives of Web applications with shared editing functionalities are Google Docs and EtherPad[4]. Here, we showcase their features and highlight state-of-the-art features for shared editing on the web.

Google Docs is a Web-based office suite available through a Google Drive account. Thereof Docs, Sheets and Slides are independent applications standing for word processor, spreadsheets and presentation functionalities respectively. All applications in the Google Drive family have rich collaboration features allowing documents to be shared to other users and edited simultaneously in real-time. Google Drive itself builds the cloud-based file container that provides a uniform user interface to share a document and notify possible collaborators via email. A public option allows anonymous users to participate in the editing. While editing the document, users can start a chat to send each other text messages. In the participant list the profile pictures of logged in users are shown while anonymous contributors get assigned a random animal name and icon. Furthermore every user is represented by a cursor with a user-specific color. For history support, Google Docs shows the user a list of changesets that can be selected. Upon selection, the part of the document that changed is highlighted in the document.

EtherPad on the other hand is a completely open-source word processing tool that can be freely hosted on any server. Both EtherPad and Google Docs provide the user with a history viewer for documents. EtherPad presents the users with an interface that resembles a minimalistic media player, enabling the user to play back the complete history of the document and watch it like a movie. Awareness features of EtherPad are shown in detail in section IV-C.

III. Web Technologies

As stated above, the Web has become a major platform to reach users on a variety of devices. Web applications can be used both on desktop and mobile browsers. Furthermore frameworks for cross-platform mobile apps are available that leverage the development possibilities of HTML5 by packaging Web content into a bundle that behaves like a native app on several platforms like Android and iOS.

[4]http://www.etherpad.org

Figure 1: Publish/Subscribe based Multi-Display Image Viewer

In the following, server-push technologies are presented including the state-of-the-art WebSockets standard. WebRTC is showcased as emerging standard that allows to use the architectural paradigm of peer-to-peer computing on the Web. Finally, widget technologies are highlighted as technology that is later used to tackle some of the issues and missing features of the shared editing frameworks surveyed.

A. Server-Push Technologies

In the early days of the Internet, Web servers served static HTML files with fixed content to the user over the HTTP protocol. Interaction was limited to HTML forms that allows users to enter information and send it to the server thus blocking the user interface while waiting for the input to be processed on the server using CGI scripts. With the advent of the Web 2.0 enabled by *Asynchronous JavaScript and XML (AJAX)*, bidirectional message flow was made possible without blocking the UI. Elaborate techniques like Comet over the Bayeux protocol use a technique called long-polling. Thereby, the TCP connection to the server is started and held open; as soon as there is processing output on the server, the information is sent on the open channel. The client then closes the old connection and opens another one to wait for the next input coming from the server. On the contrary, if the client has to send new input to the server, the old connection is closed and the new information is sent over a newly established TCP channel.

Another popular use case for long-lived TCP connections is the Bidirectional-streams Over Synchronous HTTP (BOSH) protocol used within the *Extensible Messaging and Presence Protocol (XMPP)* [10] defined in XEP-0124 [11]. Coming from a mere Instant Messaging use case, XMPP became a common protocol for a multitude of use cases like real-time middleware and *Internet of Things (IoT)* [12]. BOSH emulates a bidirectional message channel over HTTP by having open multiple request/reply pairs between the client and the server.

The latency of the underlying protocol is an important factor in real-time collaborative software. In mobile, the message size is important as well. WebSockets has been introduced by the W3C as a means to enable stable TCP connections from a Web browser to a server without the need to misuse the HTTP protocol. To provide backwards-compatibility, a standard HTTP request has to be sent by the WebSockets-enabled browser to start a connection. Hereby, an upgrade header is included to indicate that the browser wants to start the WebSocket connection. If the server understands the protocol, the HTTP connection is skipped, and a stable connection is established on the same TCP channel Now, to send a message from the client to the server or vice-versa, no additional HTTP request has to be started, thus reducing the cost of exchanging the HTTP headers. As for XMPP, in earlier work we have presented a plugin for the popular Strophe JavaScript XMPP client that enables data to be sent via WebSockets instead of the BOSH protocol described above[5]. Experiments have shown that XMPP communication via WebSockets outperforms BOSH with considerable efficiency improvements [13].

B. WebRTC

Besides WebSockets, *Web Real-Time Communication (WebRTC)* is an emerging Internet draft currently under revision at the IETF and the W3C that brings real-time communication capabilities to browsers. While previously plugins such as Adobe Flash and Java Applets had to be installed at the client side to enable real-time communication applications like video streaming, WebRTC is a complete stack of protocols and APIs built directly into the browser.

Use cases include not only audio/video communication but also direct message exchange between peers. WebRTC offers a complete stack including different APIs to acquire audio/video content such as webcam, screen capture and microphone on the client device as well as protocols that support peer-to-peer message exchange between remote browsers. Different media tracks are bundled into a single media stream that can be streamed to another peer. Every message exchange between peers is encrypted by default.

To start the communication establishment between two peers, signaling is required. The WebRTC draft does not provide a specification on this, except that the connection setup is based on the *Session Description Protocol (SDP)*. Instead, the decision on how to exchange SDP statements is delegated to the web application developer. To overcome possible firewalls and *Network Address Translation (NAT)* routers, the *Interactive Connectivity Establishment (ICE)* protocol is used together with *Traversal Using Relays around NAT (TURN)* and *Session Traversal Utilities for NAT (STUN)* servers. ICE tries to find the best path for the media stream which can be over the internet or over the local link, if peers are on the same WiFi network for instance.

For collaborative Web applications, the DataChannel API of the WebRTC specification is of high interest. It allows arbitrary binary data to be sent from one browser to another in an encrypted channel. In latency sensitive applications such as shared editing systems, messages could save the indirection time of being sent over a possibly remote server.

[5]https://github.com/strophe/strophejs

C. Widget based Web Applications

Most of today's websites provide their functionality in a monolithic page that comprises all the functionality in a single interface. One possible way to achieve a separation of concerns within the user interface is by employing HTML5 specific tags such as `header` or `menu`. Another way is dividing a website in distinctive functional pieces. This can be achieved by widgets. They describe single units of well-defined functionality embedded into a user interface component. As such, they resemble mobile apps, where every screen stands for a clearly defined functionality. A major benefit is that widgets lend themselves to be reusable across applications.

A set of widgets displayed in the same context possibly on a single webpage defines a widget container. From the technical perspective, these combinations require a managed application context with various runtime functionalities such as user management provided by the container. An example for such a widget container is the open source ROLE SDK[6] that is using an Apache Shindig[7] container. The SDK provides a widget context together with a set of features like *Inter-widget communication (IWC)*, authentication and authorization as well as *Inter-widget communication (IWC)*. IWC enables multiple widgets to exchange messages following a publish/subscribe pattern. Widgets may publish a message together with a subject and all other widgets interested in this particular subject get notified and may then act accordingly. A typical use case for IWC is the orchestration of complete Web applications from single widgets. For local IWC message exchange, the widgets communicate via the HTML5 Messaging API. Remote IWC that is applicable when the same application is running on multiple devices is coordinated via XMPP. This XMPP channel can further be used for presence updates and a multi-user chat.

IV. Characteristics of Shared Editing Systems

After the foundation of state-of-the-art Web applications was laid out above, this section presents requirements and up-to-date best practices for designing shared editing systems for the Web.

A. Consistency Algorithms

Consistency management is at the heart of every shared editing system as it ensures that all clients working on the same artefact remain synchronized and edits at diverse instances of an application do not harm each other. Possible conflicts are resolved in a fashion that is satisfying for every participant in the collaboration session. For shared editing systems this means that all users will have exactly the same data after every change. A variety of consistency preserving algorithms is available today, the most prominent one is *Operational Transformation (OT)* with its variations.

Simple approaches for ensuring consistency include floor control. In a multi-display image viewer prototype we developed, we employ a simple publish/subscribe based mechanism to save the state of the application. The system can be seen in Fig. 1. When the application starts, the tablets are allocated within a grid before an image is loaded. Whenever the image

[6]http://sourceforge.net/projects/role-project/

[7]http://shindig.apache.org

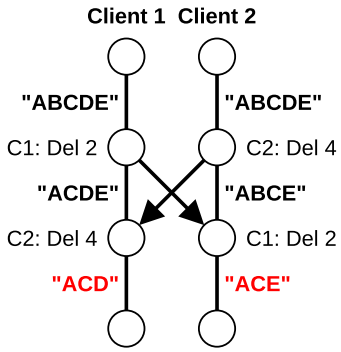

Figure 2: Example of divergence in OT

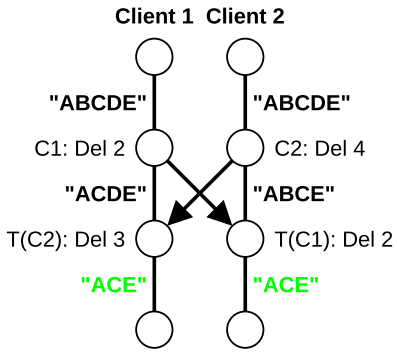

Figure 3: Example of convergence in OT

gets dragged on one screen, a message with the new absolute coordinates of the image is published onto a common channel. Upon that, the other displays are notified. In our tests, even small latencies to the publish/subscribe server affected the usability in a bad way and simultaneous drag events resulted in flickering. One strategy to resolve this would be to introduce floor control, so that only one tablet may serve as input device at one point in time. This floor control can be implemented by a token system, where only a single user who is currently holding the token may submit updates to the document. The user interface at all other instances would then be locked while not holding the token.

1) Operational Transformation: As opposed to simple floor control algorithms, operational transformation is about transforming possibly conflicting operations against each other so that conflicts are resolved. The concept of operational transformation (OT) was first introduced in 1989 by Clifford et al. [14]. Since then, a variety of refinements for special use cases have been presented. The basic idea behind OT is to capture changes done by users as operations, e.g. "insert character x at position y" and let other clients apply those changes themselves taking into consideration the version of the document the operation was executed on. All clients have their own OT engine that is able to apply and transform incoming operations.

A problem that every OT system has to solve is that of divergence. Divergence occurs when multiple users apply changes at the same time, i.e. two sets of changes are generated from the same starting point. In Fig. 2 the changeset from a different user is applied without adapting the operation to the new condition (which is an inserted character), whereas in Fig. 3 an OT engine transforms incoming operations before they are applied.

2) Differential Synchronization: In 2009, Neil Fraser proposed a different approach, called Differential synchronization [15]. In contrast to OT, DS does not work with operations but operates directly on the data itself. It works by continuously diffing and patching the copies of the clients and servers. Though the algorithm itself is fairly easy to implement, it relies on powerful diff and patch algorithms that support fuzzy and preferably also semantic diffs and patches.

B. Architectures

There are three possible architectures for shared editing systems, namely centralized, peer-to-peer or hybrid. Traditionally the Web favors a centralized approach where consistency preserving algorithms run on a server that clients connect to. Updates are sent to the server and get promoted to other clients working on the same artefact.

In peer-to-peer systems, every clients holds a record of the synchronized data. Here, every session participant get updates from all clients. Even though consistency algorithms working in a peer-to-peer fashion have been proposed (see [16]), they are not applicable for the Web. However, the emergint WebRTC standard could bring these peer-to-peer architectures to the Web.

Hybrid approaches take advantage of the power of both, centralized and peer-to-peer. In conjunction with recent developments like the upcoming of systems that support multiple devices per user as opposed to one-device-per-user [17], hybrid collaborative applications could ensure that on one hand updates of a user are promoted to other devices of the same user in a peer-to-peer fashion. On the other, these updates could be sent to a central instance that makes sure that these operations are passed to other clients of the system.

C. Awareness

Besides consistency management, being aware of other users' changes is an essential part of online collaboration. This feature called workspace awareness answers the questions *who* is working on the same document simultaneously, *what* the collaborators are doing and *where* in the document they are working [18]. User studies have shown, that users require awareness mechanisms to answer these kind of questions [5].

Depending on the type, awareness tools can either be tightly integrated into an editor's user interface element itself or be available as separate modules. Separate user interface elements may answer the question who is currently connected to a collaboration session by showing a participant list. Whereas the exact position of current remote updates such as new input in a text editor is highly application specific and needs to be tightly coupled within an UI element itself. Traditional approaches regarding awareness widgets are *telepointers*, *teleselection* or *radar views* [19]. Telepointers are duplicated cursors that

show the current mouse cursor position of other participants, possibly with a small label depicting the user's name; the same principle is applicable for a text cursor in a text field. Similarly, telesection refers to highlighting parts of the document that are currently selected at a remote user's application instance. Radar views are scaled down outlines of the whole document enriched with squares marking the currently visible screen area of remote collaborators.

For applications supporting shared editing on the Web, the approaches to achieve workspace awareness have to be examined specifically in due consideration of possible constraints within the browser. Approaches like radar views cannot be easily transferred into browser environments due to the presence of various screen sizes, resolutions and display screen densities on desktop computers compared to smartphones or tablets. Even though approaches like responsive web design tell us to adapt content to various screen sizes in a fluid way [20], they do not yet specifically consider awareness features for collaborative applications.

Means for workspace awareness can be found in the prominent shared editing tools presented above, namely Google Docs and EtherPad. Fig. 4 shows an exemplary screenshot of EtherPad with numbered labels on top of awareness tools that are explained in the following.

1) User-specific colors: As can be seen here, EtherPad uses various colors to make its users distinguishable. These colors are randomly assigned as soon as a new user enters the current collaboration session. All text that is written by the user has a colored background according to his color. Thereby EtherPad takes care of the right contrast of the background color to the text color. The inputs of no longer connected authors are shown with reduced saturation. A similar coloring scheme is available in Google Docs.

2) Mouseover effects: Text written by a user has a mouseover effect that reveals the author when the mouse is placed over a text region. The tooltip shows the fixed user identifier instead of the name the user gave himself in order to avoid confusion. The user ids are still accessible when a user has left the document editor.

3) Participant list: The user interface of EtherPad features a user counter in the top right counter. Behind that button the participant list is shown that also indicates the colors of the participants.

4) Chat: A chat widget allows remote collaborators to talk with each other without having to type messages within the document itself. Likewise to the document the chat is colored with the respective user colors.

D. Undo/Redo and History

Finally, *undoing* and *redoing* a certain edit operation and history support for saving the former states of a document are important requirements for shared editing systems. Undo and redo functionality require the system to actually store operations to later undo them. Hereby, undo and redo support is tightly coupled to the underlying consistency management algorithm. A system based on differential synchronization works in the same way as a system utilizing operational transformation, with the difference that in systems employing differential synchronization operations are basically the generated diff patches.

There are two main ways of implementing undo operations. The first way can be used if the system supports exclusion transformation. In that case an operation can be removed entirely from the document's history by rolling back the system to the state of the operation that is being undone and reapplying all changes after the transformation with ET against the undone operation. If ET is not supported, undoing an operation via inclusion transformation can be done, but the undoing operation will show up in the history as as a separate operation and the undone operation will remain in the history.

A special case of history is support of *late join*. Late join describes techniques to load the collaborative assets onto a client that has not been part of the collaboration session at all or at least for a certain period of time. Therefore this also includes the case that a client has received updates, then went offline and finally after a certain period of time online again. In this situation, the client may already keep some of the shared data in his cache. More specifically in Web scenarios the content could be saved in a HTML5 indexed database or any other client-side archive. Late join is mostly implemented by re-flushing all the content of a document from a central resource.

V. Development Frameworks

While in the sections above, necessary requirements for shared editing and the implications of a Web setting for these requirements were described in detail, this section introduces state-of-the-art shared editing libraries and frameworks that can be used by developers to create applications with collaborative features. We survey open-source solutions (with the exception of the Google Drive SDK Realtime API) that can be easily used and integrated into custom software such as research prototypes.

We have shown the importance of both consistency algorithms and workspace awareness features for shared editing systems. Hence our hypothesis is that frameworks allow for elaborate conflict resolution and rich awareness features. As so, we expect ready-made hooks for functionality like consistency management, undo/redo and user interface widgets for common awareness tasks like a participant list.

A. Google Drive SDK Realtime API

The Google Drive SDK Realtime API is the most recent shared editing framework included in our survey, as it was introduced by Google in March 2013 and has since then experienced a couple of feature upgrades. As part of the Drive SDK the Realtime API is a client-only library that can merely be used in combination with Google servers. In contrast to Google's own collaborative web applications like Google Docs anonymous users are not permitted and as such using the API requires the users to have an active Google Drive account. Before being able to take part in a collaboration session, an authorization dialog is shown to users to grant permission to the application. Applications that intend to use the Realtime API need to be registered at Google's API site.

The Realtime API follows an object oriented approach. Supported basic types that are built in are strings, lists and

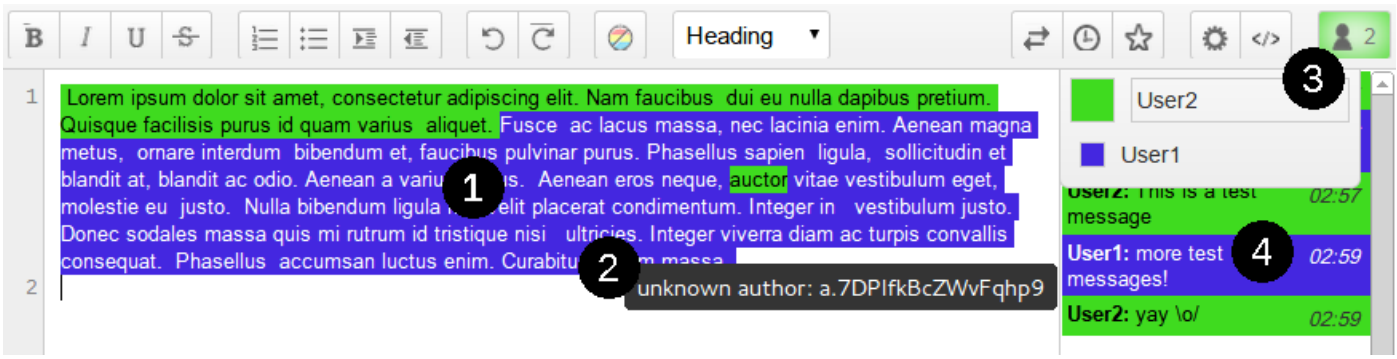

Figure 4: Awareness Tools in EtherPad

maps. Custom objects can be created that are composed out of these basic types or other custom objects on their part. Listing 1 (a) shows an example where the string variable `str` is created. The custom object `book` of type `Book` is created below. Custom objects need to be registered to the model together with their properties before they can be initialized. Object's properties can be other collaborative objects; circular structures like in graph based data models are possible. As the DOM tree represents a special kind of graph, it can be translated into the API's data model.

After being created, collaborative variables become part of the JavaScript application's data model and are therefore available on remote instances of the application. The collaborative types follow an event-based approach and provide listeners that are called upon specific events such as insertion, deletion or change of content. Besides listening for updates, string types can also be directly bound to user interface elements as shown in Listing 1 (b). Hereby, the API takes care of updating the bound text field upon remote changes and notifies remote instances upon changes in the local UI respectively.

The API features rich awareness features such as a user management with an event-based participant list. Users are automatically assigned random colors. Due to the fact, that only users logged into their Google Drive account can take part in a session, the API can be queried for the user profile including the name and a profile image. the Google Drive account's username is queriable over the participant list. Furthermore, string type variables can be supplied with the current cursor position making it easy for developers to implement workspace awareness widgets such as telecursors.

As shown in Listing 1 (c), undo/redo is implemented within the API. To bundle several operations into a single undoable operation that is sent to the server at once, compound operations can be used as in Listing 1 (d).

```
// (a) initialize string variable
var str = model.createString('Hello World!');
model.getRoot().set('text', str);

var Book = function() {};
realtime.custom.registerType(Book, 'Book');
Book.prototype.title =
  realtime.custom.collaborativeField('title');
var book = model.create('Book');
```

```
// (b) binding to user interface
var str = doc.getModel().getRoot().get('text');
var textArea = document.getElementById('edit');
realtime.databinding.bindString(str, textArea);

// (c) add undo button logic
var model = doc.getModel();
var undoButton =
  document.getElementById('undoBtn');
undoButton.onclick =
  function(e) { model.undo(); };

// (d) bundling operations
model.beginCompoundOperation();
myCollaborativeList.push("Hello");
myCollaborativeList.push("World");
model.endCompoundOperation();
```

Listing 1: Various operations with the Google Realtime API

However, as the API is the client-side API is obfuscated JavaScript code and the backend service is the Google Drive cloud service itself, its use remains little for developers wanting to create a custom application on their own servers. As we found out in our experiments, it even allows for outages in network connection; operations are persisted locally and pushed to the server as soon as the network connection is available again.

B. *OpenCoweb*

The OpenCoweb framework consists of the Coweb server and client libraries. The server is available in both a Java and a Python version, aiming to provide the same set of features with a greater target area. So called "service bots" that act on incoming operations and other data can be implemented for the system, resembling the Wave robots presented in Section II. The server communicates with the clients over the Bayeux protocol, which uses either Comet or WebSockets and is part of the CometD project of the Dojo foundation.

The underlying consistency algorithm is operational transformation; on OpenCoweb it is intended to work primarily on text on a per-character basis; still it works on any object that can be serialized as JSON. It supports 4 types of operations, namely "update", "insert", "delete", and a "null" operation that is used by the transformation engine if an operation has become superfluous. Listing 2 shows an example of how a

character in the collaborative field `examplename` gets added and deleted again.

```
collab.sendSync("examplename",
  "x", "insert", 5);
collab.sendSync("examplename",
  null, "delete", 5);
```

Listing 2: Inserting and deleting characters in OpenCoWeb

OpenCoweb comes with a set of prebuilt widgets including a rich text editor, a chat and a Google Maps widget. The Framework does not keep a history of operations, so undo/redo and history functionalities are not included. Although there is no function that handles late join itself, there are designated callbacks.

OpenCoweb's core operational transformation library is also available as a separate project called coweb-jsoe, for cases when a complete shared editing framework is not needed or a server setup is not available. Therefore this library can be used to utilize OT functionality without the server component of OpenCoweb. In that case, developers must take care of the sending, receiving, generating, applying and transformation functions for the operations of their collaborative application.

The documentation of the project is very extensive and features commented code, beginner tutorials, example applications and boilerplate code for almost every function the framework provides.

C. MobWrite

MobWrite was developed to showcase the capabilities of the differential synchronization algorithm presented in section IV-A2. To our knowledge, it is currently the only shared editing framework that is based on DS. The library is focused on synching existing HTML elements in websites, with very little customizability and a high-level API. Standard elements that can automatically be shared are "input" elements including textfields. MobWrite can either be deployed on Google's AppEngine platform or run an own server via a python daemon.

```
mobwrite.share("textfield");
mobwrite.unshare("textfield");
```

Listing 3: Sharing and unsharing text fields in MobWrite

MobWrite knows two different data types that can be shared, which are text and numbers/enums. The merging of text in form of an HTML input field happens with diffs and a fuzzy patch algorithm. Enum means any kind of field where the user can select one or more values from a static list, e.g. checkboxes or checklists. These elements are not merged, because changes on these fields are all operations that just set a specific value without a bigger context. If there is a conflict in these fields, the latest action just overwrites the previous one.

Listing 3 shows the main way to incorporate MobWrite in a project. By calling `share` with the identifier of a HTML input field, the field is synchronized with parallel instances; calling `unshare` stops the synchronization efforts.

D. ShareJS

ShareJS is a project started by former Google Wave engineer Joseph Gentle [2]. The focus of this project lies on simplicity and usability, specifically the author wants to build a shared editing framework that is simpler to employ than Google Wave. Gentle claims that "ShareJS is a simple (~4k LOC) CoffeeScript server & web client library for OT.". Most of the code is written in CoffeeScript, a language which compiles to JavaScript. The server runs one the Node.js platform. ShareJS can use either the socket.io or BrowserChannel libraries to communicate with the server, therefore communication over comet or WebSockets is supported.

In ShareJS, developers do not need to care about specific operations if the framework is only used for synchronizing strings. Large parts of the source handling the conflict resolution algorithms based on operational transformation are taken from the EtherPad project introduced earlier. OT is also supported for arbitrary JSON objects. The JSON API is modeled as a list of objects whose items can be inserted, removed, replaced and moved. Late Join is handled by getting a complete copy of the current document's state from the server; ShareJS does not have any history functionality yet.

```
doc.submitOp({i:"Some text", p:100}, callback);
doc.insert(100, "Some text", callback);
```

Listing 4: ShareJS

The framework is object oriented; the `doc` variable in the demo code in Listing 4 refers to the currently edited data. The code excerpt shows how developers may directly work on OT operations. Both the first and second line accomplish the same, which is inserting text at position 100; the second one simply uses convenience methods provided by the framework.

E. Changesets

As opposed to the libraries above, changesets is not a full framework for shared editing as it does not provide a full setting with server-side tools and network communication. Instead, the library tries to give developers easy API access to operational transformation functionality. It only features plain-text based OT; operations are generated by using the 'diff-match-patch" library by Neil Frasers that is also used in MobWrite to implement differential synchronization.

```
// (a) constructing and applying a changeset
var changes = cs.text.constructChangeset(text1,
  text2);
var finalText = changes.apply(text1);

// (b) serializing a changeset
var serialized = changeset1.pack();
var changeset2 =
  cs.text.Changeset.unpack(serialized);
```

Listing 5: changesets

Besides being solely focussed on OT, changesets is also the most low-level library in this paper. It requires developers to invoke the OT functions manually to construct the changesets and even distribute them to other collaborators. The example code in Listing 5 (a) shows how a 'changeset' is created out of two different strings. Finally the changeset is applied

to `text1`; the result is that both `text1` and `finalText` contain the same string. Besides constructing changesets it is also possible to let the library serialize the changesets for more efficient messages to be transferred to participants in the collaboration session, which is shown in Listing 5 (b). This example shows the packing and unpacking of a changeset.

F. CEFX

The Collaborative Editing Framework for XML (CEFX) proposed by Ansgar Gerlicher [9] is a library for performing consistency management for large documents. Even though it is not specifically targeted on providing shared editing functionality for Web, it is included here because of its new concepts for the architecture and conflict resolution algorithms.

The first difference to most other shared editing systems is that CEFX uses multiple history-buffers per document. History-buffers are essentially queues of operations that other clients have sent to the local client. Usually these buffers are empty because incoming operations are applied to the local copy immediately after arrival (or transformation).

CEFX has been constructed especially for XML data; even plaintext data is transformed into an XML structure. To manage a document it incorporates the hierarchy of the XML data in the sharing algorithm. For each layer of XML CEFX manages a separate history-buffer. This means that every operation that is performed on the document affects not the complete document but only specific elements in a tree of elements. A big advantage of this is that this system has to do far less transformations than systems that work with linear data. CEFX is an example for a flexible multi-target framework as it is already employed in a wide variety of research prototypes such as WatchMyPhone shown in section II. We have already used the framework in the context of cloud-based semantic multimedia annotation tools [21]. As it uses the XMPP protocol as message middleware to distribute updates, a binding for the Web would be a reasonable next step.

VI. Evaluation

In the last section, various shared editing frameworks have been presented in detail together with code excerpts. Here, we focus on the implications of the results considering the requirements we have elicitated in Section IV. As demonstrated, collaborative software requires features such as consistency management and support for workspace awareness amongst others.

The results have been summarized in Table I. As can be seen, all shared editing frameworks with the exception of MobWrite employ operational transformation to ensure conflict resolution when multiple users work on the same document. Furthermore, owing to the traditional architecture of the Web where static webpages were delivered to client's browsers to be displayed there, almost all frameworks employ a client/server model and come with server-side code. Often the heavy-weight conflict resolution algorithms are performed at the server.

When it comes to support for workspace awareness, only the Google Drive SDK Realtime API ensures state-of-the-art features such as automatic color assignments to users and hooks for subscribing to remote cursor events.

To catch the code complexity, we have computed the total lines of code (LOC) of the respective libraries. Changesets has the fewest lines of code which is due to is very low-level API. However, this enables the library to be used in various scenarios. For the Google Realtime API we were not able to get the code, as this library follows a blackbox approach that makes it impossible to see the server-side source.

The completeness of the documentation is indicated by the amount of demo code, tutorials, community support and code comments. As it happens, all frameworks with the exception of CEFX had satisfying documentation that allow developers to get started easily.

VII. Conclusion

In this paper, we provided a survey of state-of-the-art frameworks for shared editing on the Web. Therefore, we have first analyzed recent developments concerning the Web such as WebSockets for efficient bidirectional client/server communication as well as WebRTC for peer-to-peer message exchange scenarios without involving a browser after connection establishment. We have presented requirements for shared editing systems like consistency management and workspace awareness and have shown implications of employing them in the Web. Finally, frameworks with mainly JavaScript libraries were presented that provide developer support for web applications providing shared editing features.

Though workspace awareness was identified as a crucial requirement for online collaboration, we have not found many implementations of awareness tools with the exception of the Google Realtime API. Here, we see a strong demand for easily employable solutions that developers can use.

One way to develop new forms of architectures and algorithms for shared editing on the Web could be using widget based environments. In the context of collaborative software, awareness widgets could be employed, such as a participant list that highlights the currently editing user's name whenever there are ongoing changes in another widget. Finally, the XMPP channel of this widget space could be used for signaling direct peer-to-peer connections so that two instances could directly push synchronization messages to each other using WebRTC's datachannel APIs.

In recent work, we have adopted the widget container of the ROLE SDK that was presented in section III-C for a collaborative tool for creating learning design models [22]. For ensuring consistency, we used OpenCoweb's standalone OT algorithm library. To distribute change operations, the built-in IWC facility of the ROLE SDK is used that ensures that all updates are propagated to remote collaborators in near real-time. We are currently updating the application to fully take advantage of the widget environment like using the presence mechanism of the underlying XMPP functionality of the IWC system. Hereby, dedicated awareness widgets like a presence list may be easily developed.

Furthermore, the adoption of WebRTC may further drive innovation in the area of shared editing on the Web. While currently the propagation of changes in our learning design modeling tool is based on IWC, it may be extended to support WebRTC's datachannel to send operations from client to client without a propagating server.

TABLE I: COMPARISON OF SHARED EDITING FRAMEWORKS

Feature	Drive SDK	OpenCoweb	MobWrite	ShareJS	changesets	CEFX
OpenSource	✗	✓	✓	✓	✓	✓
License	Apache	BSD + AFL	Apache	MIT	MIT	unknown
Consistency	OT	OT	DS	OT	OT	OT
Awareness Support	✓	✗	✗	✗	✗	✓
Undo/Redo	✓	✗	✗	✗	✓	✗
Serialization	JSON	JSON	own format	JSON	own format	XML
Granularity	JSON	characters	characters	characters	characters	XML-nodes
Lines of Code	—	42.390	9.936	35.032	2.545	9.123
Documentation	✓✓	✓✓	✓	✓✓	✓✓	✗

After all, enhancing developer support for shared editing systems could eventually lead to a Web where we can edit any kind of document, image or video from any computing device without being interrupted by synchronization conflicts when changing from one device to another.

ACKNOWLEDGMENT

The work concerning the multi-display map described in section IV-B has been funded by a grant of the Japanese Society for the Promotion of Science (JSPS). The remaining work was supported through the LAYERS FP7 ICT Integrated Project (grant agreement no. 318209) of the European Commission.

REFERENCES

[1] The Apache Software Foundation. Apache Wave (Incubating). [Online]. Available: http://incubator.apache.org/wave/

[2] Joseph Gentle. (2011) Sharejs: Live concurrent editing in your app. [Online]. Available: http://sharejs.org/

[3] 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 Transactions on Computer-Human Interaction*, vol. 13, no. 4, pp. 531–582, 2006.

[4] M. Roseman and S. Greenberg, "GROUPKIT: a groupware toolkit for building real-time conferencing applications," in *CSCW '92 Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, 1992, pp. 43–50.

[5] M. Heinrich, F. Lehmann, T. Springer, and M. Gaedke, "Exploiting single-user web applications for shared editing: a generic transformation approach," in *Proceedings of the 21st international conference on World Wide Web*, ser. WWW '12. ACM, 2012, pp. 1057–1066.

[6] M. Heinrich, F. Lehmann, F. J. Grüneberger, T. Springer, and M. Gaedke, "Analyzing the suitability of web applications for a single-user to multi-user transformation," in *Proceedings of the 22nd international conference on World Wide Web companion*, ser. WWW '13 Companion, 2013, pp. 249–252.

[7] M. Heinrich, F. J. Grüneberger, T. Springer, and M. Gaedke, "Reusable Awareness Widgets for Collaborative Web Applications – A Non-invasive Approach," in *Web Engineering*, ser. Lecture notes in computer science. Springer Berlin Heidelberg, 2012, vol. 7387, pp. 1–15.

[8] S. Bendel and D. Schuster, "WatchMyPhone — Providing developer support for shared user interface objects in collaborative mobile applications," in *2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 166–171.

[9] A. Gerlicher, "Erweiterung bestehender anwendungen um kollaborative funktionen mit hilfe des collaborative editing framework for xml (cefx)," in *Aktuelle Trends in der Softwareforschung*, D. Spath, K. Haasis, and D. Klumpp, Eds., vol. 2. Stuttgart: Fraunhofer-IRB-Verl, 2005, p. 150–165.

[10] Saint-Andre, Peter, "RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core," 2011.

[11] Paterson, Ian and Smith, Dave and Saint-Andre, Peter and Moffitt, Jack, "XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH) ublish-Subscribe Version 1.10, Draft," 2010.

[12] A. Hornsby and R. Walsh, "From Instant Messaging to Cloud Computing, an XMPP Review," in *Proc. of the 14th IEEE Symposium on Consumer Electronics (ISCE)*, 2010, pp. 1–6.

[13] S. Govaerts, K. Verbert, D. Dahrendorf, C. Ullrich, M. Schmidt, M. Werkle, A. Chatterjee, A. Nussbaumer, D. Renzel, M. Scheffel, M. Friedrich, J. L. Santos, E. Duval, and E. L.-C. Law, "Towards Responsive Open Learning Environments: The ROLE Interoperability Framework," in *Towards Ubiquitous Learning*. Springer Berlin Heidelberg, 2011, vol. 6964, pp. 125–138.

[14] J. Clifford, B. Lindsay, D. Maier, C. A. Ellis, and S. J. Gibbs, "Concurrency control in groupware systems," in *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, J. Clifford, B. Lindsay, and D. Maier, Eds., vol. 18.2. New York and NY and USA: ACM, 1989, p. 399–407.

[15] N. Fraser, "Differential synchronization," in *Proceedings of the 2009 ACM Symposium on Document Engineering*, Uwe M. Borghoff and Boris Chidlovskii, Eds. New York and NY: ACM, 2009, p. 13–20.

[16] S. Weiss, P. Urso, and P. Molli, "Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks," in *29th IEEE International Conference on Distributed Computing Systems, 2009*. Piscataway and NJ: IEEE, 2009, p. 404–412.

[17] D. Kovachev, D. Renzel, P. Nicolaescu, and R. Klamma, "Direwolf - distributing and migrating user interfaces for widget-based web applications," in *Proceedings of ICWE 2013, LNCS 7977*. Springer, 2013, pp. 99–113.

[18] C. Gutwin, G. Stark, and S. Greenberg, "Support for workspace awareness in educational groupware," in *The first international conference on Computer support for collaborative learning*, ser. CSCL '95. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995, pp. 147–156.

[19] C. Gutwin and S. Greenberg, "The effects of workspace awareness support on the usability of real-time distributed groupware," *ACM Trans. Comput.-Hum. Interact.*, vol. 6, no. 3, pp. 243–281, Sep. 1999.

[20] M. Nebeling and M. C. Norrie, "Responsive Design and Development: Methods, Technologies and Current Issues," in *Proceedings of ICWE 2013, LNCS 7977*, vol. 7977. Springer, 2013, pp. 510–513.

[21] D. Kovachev, G. Aksakali, and R. Klamma, "A real-time collaboration-enabled mobile augmented reality system with semantic multimedia," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*. IEEE, 2012, pp. 345–354.

[22] P. Nicolaescu, M. Derntl, and R. Klamma, "Browser-Based Collaborative Modeling in Near Real-Time," in *Proceedings of the Ninth International Conference on Collaborative Computing (CollaborateCom 2013)*. IEEE, 2013.