A Document-Centric Method For Combined Synchronous And Asynchronous Applications

Lutz Gericke, Christoph Meinel
Hasso Plattner Institute Potsdam
Prof. Dr. Helmert Str. 2-3, Potsdam, Germany
{lutz.gericke, meinel}@hpi.uni-potsdam.de

***Abstract*—In this paper, we present a concept for developing applications that allow users to work synchronously together while being able to use asynchronous features, such as work resumption from any point in time. Therefore, we formulate abstract requirements for a protocol realizing the introduced approach. Furthermore, an architecture for deployment is outlined. We show three different applications – all realizing the proposed method. An evaluation summarizes the drawbacks and advantages of the approach.**

The introduced concept should show up a practical solution especially to sufficiently store collaboration processes. By proving the combination of synchronous and asynchronous features into one application to fulfill basic user needs, it could be an efficient way for applications realizing two working modes, which mostly have been addressed separately in previous solutions.

***Keywords*—synchronous collaboration; asynchronous collaboration; collaboration process history**

I. Introduction

In the recent time, we have seen many applications moving from single user applications to multi-user environments. With the growing importance of cloud-based applications, a key features is to keep documents on servers in a central infrastructure. People are more and more used to give away data sovereignty from their local computers to third-party infrastructure. Those applications often still lack features for near real-time collaboration, although the infrastructure would be perfectly suited for implementing collaborative features easily. Functionality in different products ranges from just storing documents and making them available on other devices or for colleagues, to live-collaboration on documents, including a limited document history. What seemingly has not been commercialized before is an application environment allowing live-collaboration and a full-detailed history function at the same time.

In interviews with different corporate work groups we found out that they are demanding for both working modes at the same time. Teams are oftentimes spread over the globe, utilizing telephones, video conferences or any other data-based communication as their exchange channels. To our experience, e-mail, shared folders, and screen-sharing are the most common tools in companies, when it comes to remote collaboration. There are still difficulties with tools, enabling people to work together on the same content, as traceability and access to shared documents is often not sufficient.

Explicit versioning of content has a long history. You can often find files named after their version on shared folders. Modern document management systems are much more advanced. There are distributed systems where it is sometimes not clear, who keeps the most recent version of a document. It becomes even non-trivial to say, if there is *the right* copy of a document. This problem is often solved by building up a central infrastructure with a version history for each document.

We assume that there is a need for near real-time synchronization among clients. This need is independent from the technology being used in the applications. The problems are similar, no matter if it is based on HTML and JavaScript or any desktop technology. The general goal is to coordinate applications, in order to keep the same content at every location while preserving a central edit history on a central server. Content can be of various kinds, it can be text-based, any graphical format or even 3D models. For our approach the only restriction is that documents have a state, streaming formats can be embedded into documents, but are not in our focus as documents to be synchronized by our approach.

This paper should not be seen as a scaffold for implementing combined synchronous/asynchronous applications, but explain a possible strategy towards realizing such a system.

II. Related Work

There has been a lot of research on synchronization of applications. This research is mostly focussed on the problem of multiple clients having a copy of one document. Application of changes on one copy will be applied on the remote sides. The main challenge faced during the implementation of these approaches is synchronization and fault tolerance.

There is a multitude of approaches dealing with synchronization of documents [9]: pessimistic, optimistic [24], edit-based (e.g. based on the operational transformation algorithm), three-way merges (as found in Google Docs, Subversion etc., described by Lindholm in [22]), differential synchronization (described in detail by Fraser in [9]) and many more. According to [24], the approach presented within this paper, is a single-master, state transfer, syntactic scheduling method and thereby could also be called an optimistic replication algorithm.

"Operational Transformation" (OT) was first described by Ellis et al. [8] and became popular since. In [6], [18], [19], the authors describe the development of systems or frameworks adapting existing applications in order to synchronize their states to other locations. OT is used as a conflict resolution strategy. Applications will transfer change-sets of operations that are applied to the document. As pointed out by Fraser in [9], server-side three way merges do not scale well in certain situations.

Rather than integrating a new synchronization approach into existing applications, we want in the first place find a generally applicable methodology to combine near real-time synchronization as well as asynchronous features into an application. There is a large need for asynchronous applications. This might have several reasons (see [1]). People are oftentimes not able to work at the same time (different working habits, timezones, etc.). We agree with Barksdale et al. [1] on their statement "not all teams have this luxury" of working together synchronously. As people are increasingly spread over the globe, but working for the same company or are otherwise in need for coordination, this becomes a growing problem.

Biuk-Aghai highlights an important point: "*... in asynchronous collaboration awareness of the event history of the collaboration space is of importance.*" [2]. Only storing the latest state of a collaboration process does not necessarily establish an asynchronous application. Archiving the whole path towards the end result and being able to replay it, might be even more important.

Frameworks for easier building groupware applications are available, such as OpenCoWeb (http://opencoweb.org). It supports Websockets as well as the cometD Bayeux protocol and uses OT as conflict resolution strategy. As browsers are growingly important as a platform for applications, Gutwin et al. [16] researched on technologies for web-based networking.

Xia et al., Davis et al., and Sun et al. all experience challenges during realization of a system implementing operational transformation [6], [25], [27]. Sun et al. [25] and Xia et al. [27] are showing a method to transparently adapt single user applications to multi-user working modes. An application combing also asynchronous and synchronous working modes is presented in [3] and [23]. Masoodian et al. [23] implemented the text editor "RECOLED", which allows users to work together on text documents while the system also captures several other data streams, such as gestures or voice.

Awareness in virtual workspaces has been studied a lot. As Dourish et al. point out in [7], awareness can be related to the content or the character. Several kinds of awareness information are imaginable, such as feedback, pointers, etc.. When combining synchronous and asynchronous working modes into one application, this becomes especially challenging. Gutwin et al. say that it is important to create awareness by knowing what happens on the other side [15]. In asynchronous systems, there is often no other side, so the takeaway for us is that awareness information as to be captured as well, which is also supported by Masoodian et al. [23].

III. Requirements for a generic synchronous and asynchronous application state management

The general goal of a synchronized application is to coordinate a state between different instances. This state contains of the content within the application, but is not limited to that. Awareness information can also be part of the application state. In a spreadsheet application, the sheet consisting of cells can be enriched by highlighting cells that show the currently editing users. This information can help guiding users attentions, but could also distract users from working properly, e.g. think of a synchronized tool bar with a color picker, when everybody wants to write in a different color. For this reason, we define the term "content" as the information within the application, which users can modify and are the same in every instance of the application.

We differentiate between three synchronization approaches: input-operation-centric, document-operation-centric, and operation-centric synchronization. The major difference in those approaches is the unit of transport, which has a large influence on flexibility of the protocol, but also on technology dependence.

For the operation-centric approach, the transfer unit is focussed on capturing change operations on the editor side, transferring it to the remote location, and reproducing the changes on the other side. Those operations can be e.g. "press key 'a'". When the user at location A now triggers input on the one location, it will be reproduced at location B. A replay of this information to achieve a certain state of the document is expensive and tedious. Reproducing a document from a series of input events demands for a lot of operations, where also - in the worst case - a lot of events will not directly be visible in the final document (e.g. writing and deleting a paragraph).

In a document-operation-centric approach, operations are abstracted, so that a change message can look like "Insert the string 'abc' at position 4 in cell 23:45". This might cause problems in resolving conflicting operations, which can be solved by operational transformation [8], but will be growing in complexity when adding new operations to the system. Almost the same problems arise as before, when trying to reconstruct a document state from a log. Snapshotting might be a workaround. There are two options: Having the user snapshotting states of work (pushing the "save"-button) or having automatic snapshotting at certain times. Defining the points in time seems difficult, as fixed intervals produce long phases of irrelevancy and during phases of intense work, leave out too much information. For us, storing just every change operation is the better option, as we can also define afterwards, which state is more relevant than another, also based on other data forms (annotations, audio/video recordings etc.).

We propose a further simplification or abstraction towards a document-centric approach. We look at content as documents and items within these documents. This simple definition matches a broad range of document types: text documents (items = lines, paragraphs), graphics editing (items = shapes, images), etc.. These artifacts/items define a simple set of operations: new, change, delete. Having this model in mind, we can realize a synchronization mechanism for many different applications. The major difference to those two methods before is the perspective towards change events. The fact that there was a change of 2 letters with the second line at position 14 is seen as too fine-grained. It merely matters that line 2 changed, so we would transfer this item as a change message.

A comparison of the three approaches can be seen in Table I, which differentiates four factors:

- *replay:* The operation centric-approaches have to retrace the whole history of events and reapply them to document items, whereas the document-centric approach just needs to fetch the latest state of all items.
- *message size:* Message payload size is often smaller in the first two approaches, because only deltas are transmitted. Thus, depending on the item granularity, the difference to document-centric is rather small.
- *error-proneness by missing messages:* A document state cannot be reliably restored, if messages are missing in the operation-centric approaches. The document-centric approaches does store full representations of the last state of an item, so leaving out messages can be acceptable.
- *conflict resolution:* needs to be more elaborate when just deltas are sent around, but can be simpler in the document-centric case
- *granularity/abstraction level:* Is higher in the operation-centric approach, as there can be fine-grained operations used. Those operations are more abstract in the document-operation-centric approach, but different from application to application. Items in the operation-centric approach have a fixed set of operations.

TABLE I
COMPARISON OF SYNCHRONIZATION STRATEGIES

	input-operation	document-operation	document-centric
replay	costly	costly	cheap
message size	smallest	small	larger
error-proneness	high	high	low
conflict resolving	difficult	yes	inherent
granularity/abstraction	large	medium	limited

Table I reveals advantages and disadvantages on both ends of the scale. As we want to build systems combining asynchronous and synchronous modes into one application, we need to focus on an approach, which makes it easy and flexible to replay sessions. In order to follow this document-centric approach, we first try to find a general description of an item and a modification event describing it. We found a tuple describing every message as follows:

$$msg = (id, doc\_id, opcode, timestamp, user, type, data)$$

id describes the identifier for the item. doc_id keeps the connection to a specific document, the item belongs to, e.g. a text file or a whiteboard panel. $opcode$ has one of the values of the enumeration $\{NEW, CHANGE, DELETE\}$, to identify the operation type. $timestamp$ describes the point in time when the operation took place. $user$ is the user, who took out this operation. $type$ reveals the item type. $data$ is the actual content of the item that has been changed. A list of events can be seen as the history of a document. In a relational database, the tuple $t_{key} = (id, doc\_id, opcode, timestamp)$ will be the primary key for the event. This brings up a limitation of the approach: The granularity of the history can be limited by the granularity of the timestamp, which could be adjusted to a second. As we have for example two CHANGE operations in rapid sequence on one item of the same document, it might be that only the last of these events are stored into the database, as we use a replace rather than a insert semantic during database insertion. It can be a proper factor for adjusting the amount of data that is being stored during high-load scenarios.

Content is typically ordered within a document. Order can be very diverse in meaning seen by the application. For a drawing application it can be a combination of position and z-order, for a text application it is just the line number. Our assumption is that the application-internal ordering should be part of the application or client-logic and thereby reflected within the data-field. As this information is highly application-specific, it should not be reflected in the database schema. One exception can be using the id-field as a criterion for ordering one-dimensional information such as text-only documents. This will not be suitable as soon as there are other item types involved, e.g. figures.

Synchronization is realized by transferring item descriptions to the other clients and applying them on the content. Therefore, the tuple is packed into a message, sent to the server, where it is broadcasted to every connected client editing the respective document. While the server has complete control over the content, it can be stored in a database, in order to allow reconstruction of the whole process, but also analysis of the work process.

Regeneration of a state of the document is straightforward. Given the set of events for a document, the events can be grouped by the item identifier and ordered using the timestamps of the events. Now taking the last event that took place just before the queried time, we get the latest state of every object. If the latest state at that point in time has the opcode "DELETE", then the object is not existing at that point in time. With that procedure you can easily reproduce any point in time of the interaction history. Of course, the most frequently used timestamp will be the latest state (see Figure 1), as it will be generated on client startup in order to receive the current document version.

Apart from the reconstruction of a whiteboard state, it is

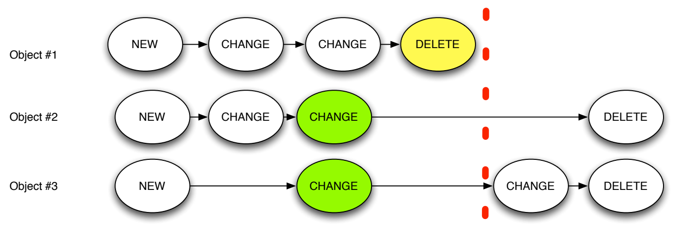


Figure 1. Schema for querying a document state. Given a point in time that should be reconstructed, all events after will be omitted and the latest event represents the latest state of the object. The x-axis represents the time. The set of items existing at the given time (red line) represent the document state.

also possible to copy one state into a new document. We refer to that as "branching". Branching is quite similar to the reconstruction of whiteboard content, because the data extracted from the database for one state will just be copied and relabeled for another document. This strategy does not copy the history information. There can be two strategies to circumvent this behavior: First, copy everything before that point in time that is going to be branched or second, copy only the state and keep a log of branch actions. For our applications, we prefer the second strategy.

The history archive can also be used to analyze team interaction processes, as we showcased in [11]. Automatically captured data can replace a lot of effort done usually via coding of captured video of the teams.

The major aspects of this section can be summarized into following points:

- A general message format is used for synchronizing diverse kinds of applications.
- Synchronization is realized over a central server system that also stores every single event message in a database.
- The stored event database can be used for reconstructing, branching, and analyzing previous work.

IV. Generalized Architecture for a Synchronizing Asynchronous Collaboration Platform

The proposed abstract protocol description is realized by two basic components - server and client. The server has three tasks: message routing, message storing, and document reconstruction. Clients continuously send update messages towards the server. Messages are distributed to the other connected clients. The server component stores the messages in an adequate format, in order to enable reconstruction of the document at any point in time of the interaction process.

Message routing describes the way incoming messages will be distributed to any other connected client. Therefore, the notion of a *Session* becomes important. The scope of a session is limited to one document, meaning there is a one-to-many (one document, many sessions) relationship of a document and its sessions. Editing a new document will assign the client to a session belonging to that specific document. All clients editing a document will receive updates for it, while being in the same session during synchronous modification.

The central server manages the sessions and handles user authorization. Depending on the transport protocol being used, the message routing is already implemented to a large extend. For instance, session handling (via multi user chat rooms) and authorization (internally or externally via LDAP) are features that are readily available with open source XMPP servers such as Openfire[1]. But for example using web technology, the effort for synchronization might be higher, as HTTP is mainly a request-response driven protocol. Although there is a variety of options to use persistent connections in the web [16]. As the general idea is not bound to a specific protocol, you could also think of any other transport protocol being used.

Another requirement for the server system is that it can capture every packet that is being sent over this server. Two possibilities are: First, self-implementing the message routing and taking care of storing the information in place. Second, using any kind of plugin mechanism to build a packet interceptor that can at least read all incoming messages. There also must be any kind of database system. It indeed makes sense to be able to have indices on attributes. The reconstruction of document states can be answered much faster, when there is at least an index on the timestamp and id fields, easing grouping, selection, and sorting on the attributes. Therefore, a relational database is highly sufficient, but thinking of more complex types of items (e.g. graphics or three-dimensional objects), it can be adequate to use specialized database types, such as XML databases or key-value stores.

We use a "last writer wins" approach including timestamps for data storage. Operations being sent to the server will be processed in the order of the timestamp. This can be used to overwrite the last state within a certain timestamp accuracy interval (e.g. one second), in order to save storage space as there can be situations having multiple operations on one element in rapid sequence.

The requirements described so far, are intentionally very technology-independent. Most of the requirements are not bound to a specific protocol, but can be applied with almost any transport protocol. In our developed applications, we used XMPP and HTTP as key technologies. We choose those technologies, because those are well-proven standards, widely adopted, and very different in their intention. The last point should show the universality of our approach.

V. Applications for the Generic Protocol

This section describes three different applications implementing the concept of a combined synchronous/asynchronous system. There are two different domains used: whiteboard interaction and text editing, both very different in their demands for user interfaces and synchronization effort. Furthermore, there are also two different protocols used: XMPP and HTTP.

[1]http://www.igniterealtime.org/projects/openfire/

A. *Tele-Board*

Tele-Board is meant to be a solution for people, who are often working at whiteboards, using paper and pens or other tangible tools, in order to visualize their ideas and designs [5]. It is our goal to support this way of working also for geographically dispersed teams. The application should retain working modes of the analog world as closely as possible [13]. Additionally, the system shall have all advantages of a digital solution, as for example saving whiteboard states and continuing at another place of work. Therefore, Tele-Board uses a paradigm of message exchange and capturing, which enables synchronous as well as asynchronous interactions [10] seamlessly integrated into one system. In contrast to other solutions (e.g. [4], [17]), we do not store images but the communication flow itself.

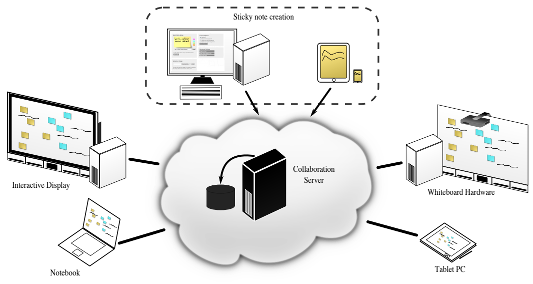


Figure 2. Tele-Board component architecture

Figure 2 shows what kind of devices can be connected to the system and the different working modes people might have using it. There are basically two different kinds of task, users can achieve using the system: creating sticky notes and working on the whiteboard. There are overlaps between these working modes.

The proposed protocol comes into play when the white-boards are synchronized. As an underlying transport protocol we use XMPP. The Whiteboard client sends updates to the communication server, which has a plugin registered that captures every packet that is sent to the server and routes it to any other connected clients of this session. A central database archives a history of the whiteboard states.

To transform the content into data units fitting into the proposed protocol format, we came up with four kinds of items: sticky notes, paths, clusters, and the whiteboard. In this implementation, the items are coded as XML. So for example a sticky note will be rendered as shown in Figure 3. It also gives you an impression on the possible actions that can be made on a sticky note: duplicate, cluster, delete, change color, pin to background.

Clusters are items to group content, Paths are drawn sketches or handwriting, and the whiteboard element stores scroll position and zoom level. This awareness information can help remote users following the editing on the remote side, as the virtual whiteboard surface can be much larger than

```
<postit id="17:312"
 color="yellow"
 x="1104.0" y="830.7427"
 sizex="60.0" sizey="40.0"
 text="Implicit storage"
 bgimage=""
 skribble="M 13.1 25.2 L 17.0..."
 pinned="false"
 skribblecolor="black"/>
```

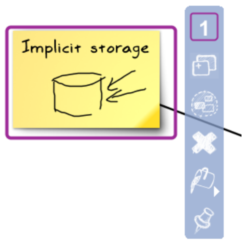


Figure 3. Sticky Note XML representation; whiteboard client view on the selected Sticky Note including a context menu bar.

what is actually shown on the screen. The scroll and zoom whiteboard events are sent continuously, but users can switch off the synchronization to their local virtual whiteboards. This can be desirable in situations, when collaborators do not want to be distracted by remote operations, e.g. while working on a private area on the board. In asynchronous settings, this information can also be used to guide users attention.

In Figure 4, screenshots of the two major user interfaces are shown. The whiteboard client, that is used to modify whiteboard content is shown in the foreground. It offers very basic tools to edit or create content with (pen, sticky notes, eraser). Content sent from mobile clients (e.g. tablets) will be delivered to the receiver area at the bottom, where people can drag content from there onto the whiteboard. The whiteboard client is meant to be used on large-scale interactive whiteboard devices as well as desktop computers. In the background, the portal is shown. It is an interface to sort panels (= documents) into projects, write sticky notes, review changes and activities, explore the history as well as starting the client.

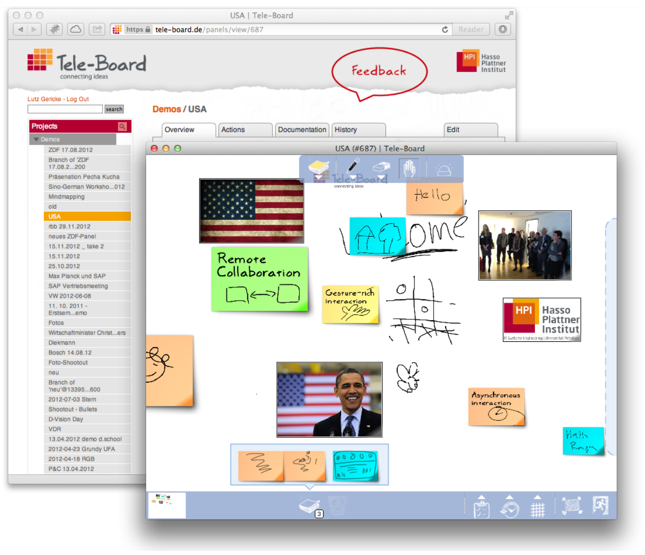


Figure 4. Tele-Board application screenshots: portal (background) and whiteboard client (foreground)

User feedback revealed that the conflict handling strategy within this application does work as expected. In the rare situation, when more than one person is dragging a sticky note to a different spot on the board, it can be that the sticky note rapidly switches from one position to another (local and remote point). According to the last-writer-wins approach, this behavior is as expected. As the irritation is minimal, we refrained from implementing special locking in the client's UI.

B. Tele-Edit

Tele-Edit is a technology demo to prove that the networking core of Tele-Board can also handle other information than just whiteboard data. Here we used a basic text editor as a front end. Users can connect to a server and start editing a document. Changes are synchronized through the server to other locations and directly archived on the server. It uses the slightly modified networking component of the Tele-Board Whiteboard client on the client-side. The server-side has not been changed at all. The only change that had to be made was for the data format according to synchronizing the text document. An item is a single text line within the editor. So a change event data payload looks as follows:

```
<line t="This is a text line." n="3" />
```

There are basically two attributes, *t* stores the text content and *n* is the line number within the document. This positional information is used by the client to sort the lines accordingly. Having in mind that inserts might trigger a lot of further change events (for all of the following lines), one could think of another approach for the position information being stored. Replacing the line number position information by a succeeding element identifier could reduce the number of messages sent to the server, but raises complexity in terms of identifier generation. A line insertion operation would then only trigger a maximum of two additional change messages. However, this linked-list approach would result in globally identical list of identifiers to be used assigned to the lines. We already use this approach in the Tele-Board Whiteboard client, where the identifier is a combination of the session identifier and a counter valid within the session. Both approaches can be used, whereas we used the one-dimensional ordering information for the special case of a text editor.

By using the system, it turns out that the last-writer-wins semantics within the server is better suited for applications having a graphical representation. When two people work on the same line, it occurs that changes of one collaborator will be overwritten by the other one. In practice, this happens not very often and can be also prevented by also synchronizing cursor locations or locking the lines for editing by just one person at the same time. This is what for example Google Docs does for inhibiting multiple editors of one line. This general problem of having awareness information of the collaborators has been studied extensively (see [14], [15], [26]) and will not be elaborated here.

C. Tele-Edit web

This application brings the previous domain - a simple text editor - to the web and shows how our concept can also be realized using web-based technology. The architecture is as before, but based on a combination of Apache and PHP on the server side and HTML5 and JavaScript on the client side. It is still a technology prototype, but can be easily adapted to other use cases. Looking at communities such as github[2], a text-editing tool using synchronous and asynchronous aspects at the same time, could offer new perspectives in terms of users really working together on source code at the same time. Figure 5 shows the application running in 3 different browsers at the same time, synchronizing the same document over these locations.

We are not relying on XMPP connections here, but use EventSource or "Server-Sent Events" as the technology for subscribing to updates. XMPP would also be an option in the browser, as there are many JavaScript-based libraries available implementing full XMPP client connectivity in the browser. Nevertheless, we wanted to showcase the still relatively unpopular EventSource approach in order to show that HTML5 has good mechanisms for synchronization already. As Gutwin et al. [16] have shown, there are a couple of technologies available being mature to be used in synchronous applications, such as Websockets. Server-Sent Events are widely supported, but one interesting fact is that fallbacks for older browsers (Internet Explorer, Firefox <6) are easy to implement[3] or one could even use a framework adapting to browser specifics[4].

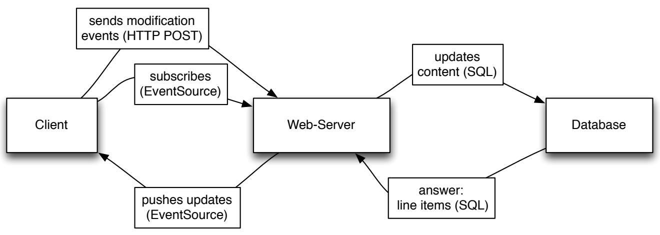
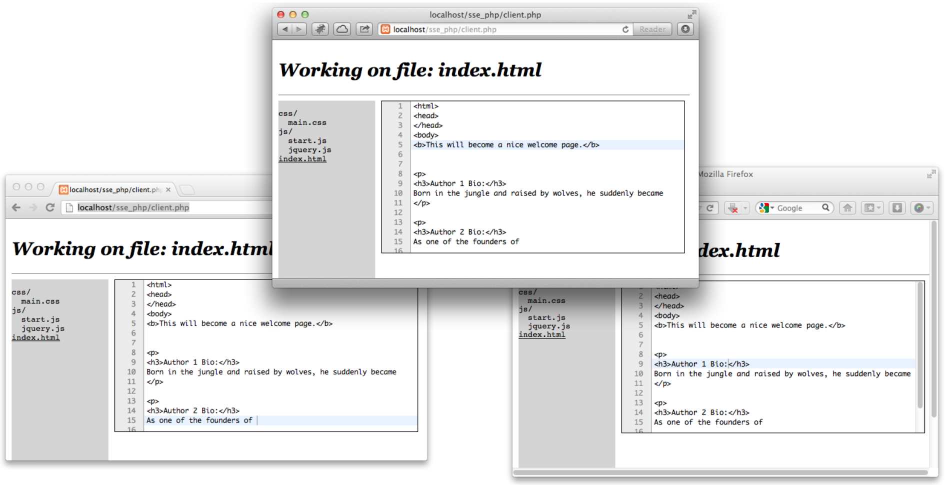


Figure 6. Tele-Edit web application general component structure

EventSource as it is specified and described in [20] and [21] as an "API for opening an HTTP connection for receiving push notifications from a server". That is why, we realized just the channel from the server to the client using EventSource. Messages from the clients towards the server are sent via HTTP POST requests (see Figure 6). There are some limitations to that approach. The PHP script serving the stream for the client side, is relying on the database and queries for changes in a fixed interval. This is delaying the message routing, because the streaming script does not keep any state itself, remaining slim in memory footprint.

The functionality is limited (see Figure 5) but it fulfills the fundamental task of having an equal copy of the document on different places and synchronizing it between them. The history of every document can be seen in the database. A possibility that arises from this infrastructure is that you can

[2]http://github.com
[3]https://github.com/Yaffle/EventSource
[4]http://www.eventsourcehq.com/browser-support


localhost/sse_php/client.php

Working on file: index.html

```
css/
  main.css
js/
  start.js
  jquery.js
index.html
```

```
<html>
<head>
</head>
<body>
<b>This will become a nice welcome page.</b>


<p>
<h3>Author 1 Bio:</h3>
Born in the jungle and raised by wolves, he suddenly became
</p>

<p>
<h3>Author 2 Bio:</h3>
As one of the founders of
```

Figure 5. A Screenshot image of a Tele-Edit web document opened in 3 different Browsers (Firefox, Chrome, Safari) simultaneously

have an almost step-less “diff” of states in-between beginning and ending of the editing process. You could also retrace which author worked on which part, who contributed most and what have been the most actively reworked parts of the document.

As a data representation we decided for JSON. The line event looks as follows:

```
{p: 1, l: 4, t: "<b>text</b> ", o: "CHANGE", u: "lutz"}
```

It transmits information about the user (u), the operation code (o), the text line (t), the line number (l), as well as the document id this line belongs to (p). This textline-item is the only one in this application. You could think of a pointer element as well, showing where remote people are currently working on, enhancing awareness between multiple locations.

VI. Evaluation

All insights presented in this section are based on informal user feedback and observations. The general idea presented in this paper definitely works for all of the different scenarios, but the “timestamped last-writer-wins” based (TSLWW) conflict resolution strategy has its limitations. While dragging sticky notes around the board, it is not problematic. Sometimes users even enjoy having little “battles”, when multiple people want to relocate a sticky. On the other hand, for the text editor scenario it could make sense to introduce more client-side concurrency control or prevention, even though conflicting situations only rarely happen. Locking the line which is currently edited by another person seams absolutely viable and acceptable from user perspective (see [23]). As already said, this strategy addresses not a technology point-of-view but more an awareness problem.

TSLWW can be also used properly in limited bandwidth scenarios. A map of queues stores the update sequence of every element. As there is the full description of an element in each message about the specific element, it would be perfectly viable to discard earlier messages about the item during limited-bandwidth scenarios and only losing some steps on the way to the end-result. This strategy does probably not work in use cases such as gaming (e.g. first person shooters) with high demands on network performance and a closer to real-time experience. There is also not a real need for a history, but more a stream-based archiving of the action is preferred.

General thoughts on performance are presented in [12], especially focussing on the impact of on-the-fly storage of the communication data. It turns out that multiple sessions in parallel are fine, but having many users in one session, all updating very frequently brings the system to its limits, as the updates have to be broadcasted, resulting in a quadratic number of messages depending on the number of users connected. We also found out that the load caused by the asynchronous features is proportional to the communication load. As this approach produces more data, depending on the granularity of items, in our scenarios (some hundreds of users produced a few GB of data) it is a very adequate way of storing information. When it comes to larger datasets, server-side compression could be an option. As the data is very structured and differences often very small between events, significant compression rates can be expected. Although, our opinion is that storage space is cheap and the workload being caused by compression is not worth the saved storage space.

Currently, security features are implemented on a document-basis. After the user having logged into the system, there is a server-based check if the user is allowed to access and edit the document. For most applications, this granularity turns out to be adequate, but you could also think of a per-item access control. Protocol level security should be realized using encryption on the communication channel (e.g. HTTPS or SSL-encrypted XMPP).

Comparing the approach to Google Docs (or Google Drive since April 2012), there are some conceptual differences.

Fraser [9] states that Google is using three-way merge [22] for synchronization. He also says that three-way merges "do not scale well when attempting real-time collaboration across a network with latency". We try to address these problems with a fault-tolerant but simple approach. Performance was shown to scale well especially on the server-side, because consistency checks are handled by the data storage inherently. We can only conjecture about the internal data structure for Google Docs document history, but the granularity of the history is not as fine, as with our approach, so that analyses of the work process can be more difficult to realize.

VII. Conclusion and Outlook

In this paper we presented a methodology describing how to easily develop applications being able to synchronize content while preserving the possibility to also work asynchronously with the data. This is achieved by storing the communication data on-the-fly and broadcast it to the connected clients. A data model describing content being structured in documents and items can be adequate for many use cases. While not transferring delta information to the database, the archived information is conveniently usable for reconstruction of activity. Additional analytic tasks of the work process can be achieved, e.g. comparing different states of the work, having statistical analyses on the usage in order to find out more about team performance, or simply analyze work processes.

We are currently working on combining the document information as presented with additional data gathered from a broad range of sensors, which usually output more of a stream format. Sensor data can range from accelerometers to captured video or audio. Having such a combination, we can analyze behavior of the people working with the system more detailed in order to come up with a system that tells you about the interesting phases of work. So far we did analyses, which are described in [11]. One of the problems of asynchronous work - especially for collaborators in different time zones - is that the archived data often does not speak for itself. Annotations or other data streams are needed to enhance understanding.

As the overall approach turned out to be adaptable in many different settings, we are confident being able to adopt this methodology in different domains.

Acknowledgments

We thank the HPI-Stanford Design Thinking Research Program for funding and supporting this project. The authors are especially grateful to Raja Gumienny and Matthias Wenzel for giving their valuable feedback.

References

[1] J. Barksdale, K. Inkpen, and M. Czerwinski. Video threads: asynchronous video sharing for temporally distributed teams. In *Computer Supported Cooperative Work*, pages 1101–1104, 2012.

[2] R. P. Biuk-aghai. *Patterns of Virtual Collaboration*. Phd, University of Technology, Sydney, 2003.

[3] M. Bouamrane, D. King, S. Luz, and M. Masoodian. A framework for collaborative writing with recording and post-meeting retrieval capabilities. *Special issue on the 6th Int. Workshop on Collaborative Editing Systems*, 2004.

[4] S. Branham, G. Golovchinsky, S. Carter, and J. T. Biehl. Let's go from the whiteboard: supporting transitions in work through whiteboard capture and reuse. *Human Factors in Computing Systems*, pages 75–84, 2010.

[5] T. Brown. Design Thinking. *Harvard Business Review*, June 2008.

[6] A. H. Davis, C. Sun, and J. Lu. Generalizing Operational Transformation to the Standard General Markup Language. In *Proceedings of CSCW 2002*, pages 58–67, 2002.

[7] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. *Proceedings of the 1992 ACM conference on Computer-supported cooperative work - CSCW '92*, pages 107–114, 1992.

[8] C. Ellis and S. Gibbs. Concurrency control in groupware systems. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, pages 399–407, June 1989.

[9] N. Fraser. Differential synchronization. *Proceedings of the 9th ACM symposium on Document engineering - DocEng '09*, page 13, 2009.

[10] L. Gericke, R. Gumienny, and C. Meinel. Message Capturing as a Paradigm for Asynchronous Digital Whiteboard Interaction. In *6th International ICST Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 1–10, 2010.

[11] L. Gericke, R. Gumienny, and C. Meinel. Analyzing distributed whiteboard interactions. In *CollaborateCom 2011*, pages 27–34, 2011.

[12] L. Gericke and C. Meinel. Evaluating an instant messaging protocol for digital whiteboard applications. In *Proceedings of the 2011 International Conference on Internet Computing (ICOMP 2011)*, pages 3–9, 2011.

[13] R. Gumienny, L. Gericke, M. Quasthoff, C. Willems, and C. Meinel. Tele-Board : Enabling Efficient Collaboration In Digital Design Spaces. In *Proceedings of 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 47–54, 2011.

[14] C. Gutwin and S. Greenberg. Workspace awareness for groupware. *Conference companion on Human factors in computing systems common ground - CHI '96*, pages 208–209, 1996.

[15] C. Gutwin and S. Greenberg. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work (CSCW)*, 11(3):411–446, 2002.

[16] C. Gutwin, M. Lippold, and T. C. N. Graham. Real-Time Groupware in the Browser : Testing the Performance of Web-Based Networking. In *Proceedings of the ACM conference on Computer supported cooperative work*, pages 167–176. ACM New York, NY, USA, 2011.

[17] L.-w. He, Z. Liu, and Z. Zhang. Why take notes? Use the whiteboard capture system. In *Proc. Int'l Conf. on Acoustics, Speech, and Signal Processing*, pages 776–779. Microsoft Research, IEEE, 2003.

[18] M. Heinrich and F. J. Gr. Enriching Web Applications with Collaboration Support Using Dependency Injection. In *ICWE'12 Proceedings of the 12th international conference on Web Engineering*, pages 473–476. Springer Berlin Heidelberg, 2012.

[19] M. Heinrich, T. Springer, F. Lehmann, 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*, pages 1057–1066, 2012.

[20] I. Hickson. Server-Sent Events W3C Working Draft 2012-04-26, 2012.

[21] I. Hickson. Server-Sent Events Editor's Draft 2013-01-02, 2013.

[22] T. Lindholm. A Three-way Merge for XML Documents Categories and Subject Descriptors. In *3rd ACM international workshop on Data engineering for wireless and mobile access*, pages 93 – 97, New York, NY, USA, 2004. ACM.

[23] M. Masoodian, S. Luz, M. Bouamrane, and D. King. Recoled: A group-aware collaborative text editor for capturing document history. In *IADIS International Conference on WWW/Internet*, pages 323–330, 2005.

[24] Y. Saito and M. Shapiro. Optimistic Replication. *ACM Computing Surveys*, V(3):1–44, 2005.

[25] C. Sun, S. Xia, and D. Sun. Transparent Adaptation of Single-User Applications for Multi-User Real-Time Collaboration. *ACM Transactions on Computer-Human Interaction*, 13(4):531–582, 2006.

[26] J. C. Tang, J. B. Begole, R. B. Smith, and N. Yankelovich. Work rhythms: analyzing visualizations of awareness histories of distributed groups. *Computer Supported Cooperative Work*, pages 334–343, 2002.

[27] S. Xia, D. Sun, C. Sun, D. Chen, and H. Shen. Leveraging Single-user Applications for Multi-user Collaboration: the CoWord Approach Categories and Subject Descriptors. In *Proc. of the ACM conference on Computer supported cooperative work*, pages 162–171, 2004.