A Proxy-Based Infrastructure for Web Application Sharing and Remote Collaboration on Web Pages

Richard Atterer[1], Albrecht Schmidt[2], and Monika Wnuk[1]

[1] Media Informatics Group
Ludwig-Maximilians-University Munich, Germany
richard.atterer@ifi.lmu.de, monius@fnuked.de
[2] Fraunhofer IAIS; BIT, University of Bonn
Schloss Birlinghoven, St. Augustin, Germany
albrecht.schmidt@acm.org

Abstract. When people collaborate remotely, the WWW is part of the shared resources they use together. However, web pages do not offer support for collaborative interaction such as viewing or influencing another user's browsing session – additional software needs to be installed for these features. In this paper, we present UsaProxy 2, an HTTP proxy that allows the same web page or application to be viewed and used in two browsers at the same time, without client-side software installation. This includes a visualisation of the remote user's mouse pointer, scrolling, keyboard input, following links to other pages and more. Our open-source proxy modifies HTML pages before delivering them to the browsers. The added JavaScript code provides session monitoring and shared browsing facilities. We conducted an experimental evaluation which shows that our approach works for different scenarios, such as shopping online and exchanging ideas on what to buy. The user study showed that our approach is accepted and liked by users. Combined with audio or text chat communication, it provides a very useful tool for informal, ad-hoc collaboration.

1 Introduction

Informal collaborative use of the WWW is nowadays very common, but not supported well by current technologies. Typically, users have a chat or audio connection and use their web browsers independently of each other, exchanging URLs via text messages or by spelling them. As a result, much of the conversation is about the current context and the content the users look at, and not about the task at hand that needs to be solved. Often, problems arise – for example, a web page may not be viewable by the remote user because of the use of session cookies. In our paper, we investigate how shared browsing sessions can be supported with lightweight technologies, i.e. in such a way that neither client-side software installation nor server-side changes are necessary. We have designed and built a system which uses an HTTP proxy and AJAX technology to enable users to collaborate on web pages. The WWW is increasingly used as a platform for web applications. With our work, we present a general approach for application sharing in this domain. In the following, we present version 2 of UsaProxy, our system for remote collaboration while using the WWW. We consider two different modes of collaboration:

C. Baranauskas et al. (Eds.): INTERACT 2007, LNCS 4662, Part I, pp. 74–87, 2007.

Monitoring. In this mode, all actions performed by a user on a web page are monitored and immediately communicated to the second user's browser. There, the captured interaction is visualised and the browser follows the actions taken by the first user, such as scrolling, text selection or following a link to another page. The second user is not able to influence the browser session. A typical scenario of use is to provide support and guidance to a user while being aware of the state and actions of their browser.

Shared Browsing. Here, two browsers' sessions are locked together. Actions in one browser result in identical changes in the other browser. Both users can see the other user's mouse pointer and both users have full control over the web page. This also means that the users can perform conflicting actions, e.g. by clicking on different URLs at the same time. For resolution of these conflicts, the presence of social protocols is assumed, and no token passing is required. In this paper, we explore two scenarios of shared browsing: Two people looking for a present in an online shop, and a teacher-learner experience.

The central contribution of this paper is a web-based technical infrastructure for monitoring and shared browsing. It was essential for us to create a solution that does not take over the complete desktop of a remote user, require software installation or changes to the web-based applications. The technology is based on UsaProxy which was introduced in [2]. Whereas our earlier work on UsaProxy concentrated on supporting remote usability tests of websites, this paper highlights a different use of the technology: Providing support for collaborative work processes. A significant number of technical problems had to be solved for this, such as how to "replay" users' actions in the second browser, distinguishing UsaProxy sessions from private browsing sessions, finding adequate visualisations for the actions, and dealing with varying formatting of pages due to differing window sizes and font sizes. A further contribution is a study that explored how people use such a web-based collaboration technology and what the potential application scenarios are.

This paper first discusses related work in section 2. In section 3.1, we introduce the concept of web-based collaboration using lightweight web technologies and describe the implementation in section 3.2. To evaluate the feasibility (with regard to the technical solution) as well as the usability of the approach, we conducted an experimental evaluation with 12 users exploring different scenarios in section 4. Finally, section 5 discusses our findings and highlights potential improvements to the technology.

2 Related Work

Client-side Software Installation. One of the earliest efforts in the direction of a shared browser application was GroupWeb [6] from 1996. The GroupWeb prototype was a custom-built browser application which could display the pages viewed by other GroupWeb users. W4 [5] is similar to GroupWeb, but also allowed bookmarks that are shared between users. Furthermore, it supported the creation of "sticky notes" on visited web pages and included a number of other tools, e.g. a drawing editor. Kobayashi et al. [9] propose a system that is comparable to our approach. However, they use a plug-in to control a standard web browser, so the plug-in has to be installed on each client machine.

In comparison to the systems above, our own efforts have resulted in a more lightweight shared browsing solution. Whereas these projects demand that all participants first install special software on their computers, our solution better supports spontaneous shared browsing sessions, and it supports situations in which users are unable or unwilling to install software. Furthermore, unlike with GroupWeb and W4, the users are not forced to abandon the web browser they normally use, with all its custom settings, personal bookmarks etc. Thus, the remainder of this section concentrates on technologies which do not require client-side software installation.

CGI-based Filter. The CoBrow project [11] used a CGI-based web interface to share pages: In this case, the browser tells a CGI program to serve it a certain page on the WWW. Before returning the page, this program rewrites all URLs on the page to make them refer back to the CGI. Only a prototype version of the tool was ever implemented. This technique is more likely to fail with complicated web pages than our own technology, which is based on an HTTP proxy. In particular, problems can be expected if pages create URLs dynamically using JavaScript.

HTTP Server, Java Applets. CoWeb [8] used a different strategy to share the content of web pages between users: Similar to CoBrow's CGI program, the CoWeb HTTP server acts as a filter for pages. It replaces some parts of the HTML content (e.g. form fields, images) with Java applets which perform the same function as the original HTML element, but provide additional functionality, such as painting inside an image and discussing its contents in a text chat. As the page is modified heavily by the CoWeb server before being passed on to the browsers, the page layout will often be significantly altered, and problems will arise if the page includes JavaScript code which accesses and changes the DOM tree.

HTTP proxy, Java Applets. With the approach of Cabri et al. [3], users reconfigure their browsers to access the web via an HTTP proxy. This proxy adds a single applet to all pages. The applet contains a list of all users taking part in a shared browsing session and a list of pages visited by them. Due to technical restrictions, the system can only visualise other users' mouse pointer positions inside selected images on the page. Additionally, users can paint in these images, and a text chat is available.

WebSplitter [7] is another proxy-based system. A Java applet causes the browser to load new pages. The system concentrates on providing different partial content to different users depending on their devices or role. Special XML pages and policy files need to be created manually, it is not possible to collaboratively browse arbitrary web pages. Furthermore, mouse movements, key presses etc. are not shared between the participating users.

Java Servlet, JavaScript. The Collaborative Web Browsing system introduced in [4] takes advantage of JavaScript to track user actions. It is designed as a lightweight system which can be used immediately by visitors of a site. The system only supports collaborative browsing on a single website which must have been prepared in advance by making a Java servlet available on the server. Collaboration is restricted to one particular page at a time – it is not possible to move the shared session to another page by following a link.

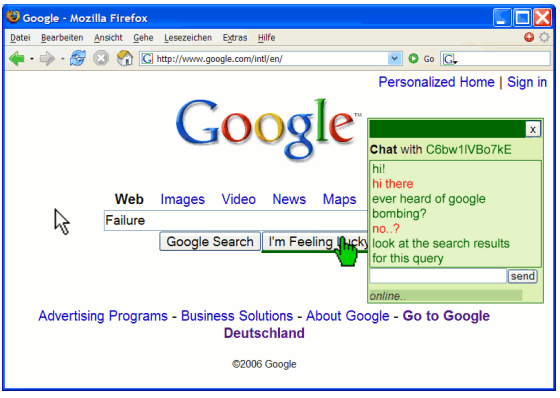

Fig. 1. During a shared browsing session, both users can interact with the page or watch the other user's actions, including his mouse movements. They do not need to install software.

Our technology combines the advantages of [3] and [4]: By using an HTTP proxy, the necessity to rewrite URLs is avoided, as all page requests and responses automatically pass through the proxy. UsaProxy 2 causes its own JavaScript code to execute on all pages when they are displayed by browsers.

Different architectures are possible when building a system for shared use of applications. In [1], this is discussed for desktop application sharing, but some of the results are applicable to the area of web application sharing. Our solution uses the "replicated execution" approach, but a "single-site execution" approach is preferable and an area for future work.

A scenario for which a shared browsing architecture may be useful is presented in [10]: During online shopping, users can leave a shared session temporarily and look at "nearby" items in the same shop. The technical solution relies on the implementation of certain APIs by the owners of web shops.

3 Using the Web Browser for Interaction and Collaboration

As many developers and companies move classical desktop applications to the WWW, new technical opportunities for creating collaborative applications arise. Inspired by tools for application sharing that are available for different operating systems, we investigated how a comparable tool can be created for web-based applications. We looked at how current browsers can support sharing and collaborative use of web pages and applications. Figure 1 depicts a simple example showing the basic functionality: In their browsers, two users see the same web page. In addition to their own mouse pointer, they are provided with a visualisation of the collaborator's mouse pointer (which was doubled in size in the figure to make it better visible) and a chat application. In this section, we first describe the concept and requirements and then explain our implementation, which is based on client-side JavaScript code and a special HTTP proxy.

3.1 General Concept

Our focus is on an approach to enable synchronous sharing of WWW pages and web-based applications without modifications to servers and clients. Compared to solutions which provide access to the complete desktop, this provides better protection of the user's privacy, as only the web page becomes visible to the remote partner. Furthermore, a platform-independent, browser-independent and non-invasive implementation is possible – an application-sharing platform can be created using standard web technologies, without extensions to the browsers or additional software installation on the client side. Moreover, no changes to web content or to the applications are required to share them. The web application sharing functionality includes:

- Visualisation of a remote mouse pointer
- Synchronisation of the displayed content (on an inter- and intra-page level)
- Visualisation of remote interaction (e.g. selection of text)
- Synchronisation of interaction with the content (e.g. text entry in a form)
- Provision of synchronous communication (e.g. chat)
- Means for shared drawing and annotation (e.g. shared blackboard)

Technically, our approach uses a proxy that adds JavaScript code to existing applications and pages before delivery. The added code tracks user actions in one browser, transmits information about it back to the proxy and replays it in the remote browser.

With regard to the mode of operation and the control in sharing we discriminate two cases, remote monitoring (one user watches the other's actions) and shared working (both users have full control and can interact with the page). Furthermore, the proxy can be inserted in various ways between the client and the server:

- Manual reconfiguration in the browser
- Transparent proxy for all machines in a local network
- Transparent insertion of the proxy in front of a server

Due to the fact that users have different browsers, different window sizes and use different font sizes, there is no true WYSIWIS ("what you see is what I see"), so that e.g. a submit button may appear in different positions. Our approach to synchronisation is not to use screen coordinates, but rather to identify the object a user interacted with by its position in the browser's DOM tree.

3.2 Implementation

UsaProxy is an open-source Java program which implements an HTTP proxy. UsaProxy 2 always forwards browser requests to the web server, except in one case: It implements caching of the server's response and uses the cache to serve identical content to the two browsers of a shared browsing session. Furthermore, a small modification is made to returned HTML pages in order to run custom JavaScript code in the context of all pages. This JavaScript code records information on user actions, such as mouse movements, and transmits it back to the proxy. Through a polling mechanism, the second browser gets notified of this. It then downloads and visualises the information, e.g. by moving a layer which symbolises the remote user's mouse pointer. See figure 2 for an overview

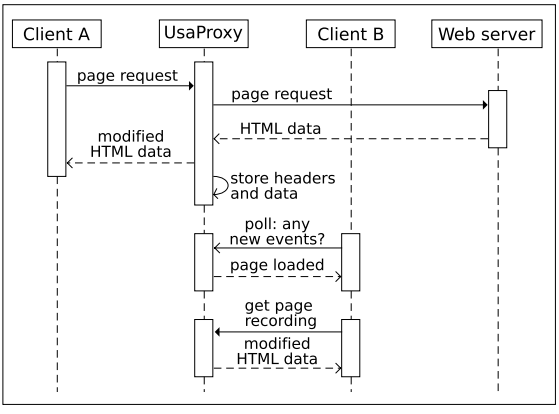

Fig. 2. When a page is requested, identical content is served to both participants of a shared session. By modifying the HTML content, UsaProxy 2 executes its JavaScript code on all pages.

of the communication between the involved entities. The following requirements were identified for the system:

- Real-time tracking of user actions, such as navigation between pages, actions on a page (mouse movements, scrolling), and any input provided to the browser (clicks, key strokes, text input, drop-down selection)
- Platform independence both from the server technology and the client operating system/browser
- As few client-side and server-side changes as possible
- Easy, flexible deployment in order to support a variety of usage scenarios
- Intuitive visualisation of remote user activity
- System should work even if the two browsers have different browser window sizes, different font settings, or render a page in slightly different ways
- Integrated chat functionality as one means of interpersonal communication

Tracking User Activity on Web Pages. When UsaProxy forwards server responses to the browser, it adds a <script src='...'/> tag to HTML pages on the fly before returning them to the browsers. This minimal change is very unlikely to interfere with the correct operation even of complex web pages. The exact URL used in the script tag depends on the proxy mode. However, in all cases the URL contains a special path starting with /usaproxylolo/, which the proxy recognizes: If it is present, the request is not forwarded to any server. Instead, UsaProxy itself acts as an HTTP server, performing one out of a number of different actions depending on the exact URL. In the case of the script tag's URL, the client-side JavaScript part of UsaProxy 2 is served.

The JavaScript code is loaded in the context of the web page, so the browser's security model does not prevent its execution. It is carefully written so as not to interfere with any JavaScript code used by the page. It records all interesting data and periodically sends it back to the proxy using another, different /usaproxylolo/ URL. On the proxy, the data is logged to a file together with the headers of any requests that were sent to servers, and with the headers and response bodies of all replies sent back by servers.

Compared to earlier versions of UsaProxy (see [2]), the range of captured events has been extended: The logged data now also includes onmouseover events, changes to form field values (radio buttons, drop-down menus, checkboxes, text fields), and the selection of text (in text fields or elsewhere on the page).

Additional Element Properties. When an event (e.g. a mouse click) has taken place for an element on the page, the element needs to be uniquely identified in the log message that is sent to the proxy. However, many elements, such as anchors or images, are not assigned an id property by page authors. Utilizing the href and/or src properties helps to identify elements in some cases, but we developed a more robust technique: We uniquely identify the elements by mapping their position in the DOM tree to a string representation. More accurately, we describe the path from the root document element to the node by recursively specifying which nth child is an ancestor of the node.

Session Cookies. We use session cookies in order to uniquely differentiate between users on all sites they visit. For security reasons, browsers do not support "global" cookies which are sent to all visited sites, so if UsaProxy 2 simply added its own cookie to response headers before forwarding the web server's response to the browser, then identifying the different users would only work as long as they did not leave the current website. For this reason, the cookie is not set for the domain that the user visits, but always for the same fixed domain name: The site the UsaProxy JavaScript code is fetched from.

Mouse Movements and Scrolling. When capturing the mouse cursor position with each corresponding event, the transmission of absolute x/y page coordinates is less valuable than the identification of the object the mouse pointer currently moves over, due to differences in output when pages are rendered. Therefore, UsaProxy 2 records the offsets of the mouse pointer position relative to the hovered-over DOM element. Apart from an offset property (e.g. offset=25,10), the DOM path and any i.e. id value are also specified in a mousemove event log. Scrolling is handled similarly: The rendered height of a page will differ, so the page offset is recorded as a percentage which gives the position of the viewport relative to the entire page height.

Clock Synchronisation. UsaProxy 2 records a variety of events which are timestamped both by the client-side JavaScript (e.g. mouse movement) and the proxy server (e.g. page requests). As the system time of the client and proxy machines may differ, the proxy transmits a timestamp to the client when it serves an HTML page. The JavaScript code then only calculates its timestamps relative to the proxy's timestamp, avoiding problems related to clock skew.

Steering a Browser Using Tracking Data. In order to reconstruct what has occurred in the partner's browser, each client constantly polls UsaProxy 2 for remote events. This is achieved using an XMLHttpRequest object. A request is composed of the special path component /usaproxylolo/getevents followed by the user's session ID and an ID specifying the last fetched event, for instance `/usaproxylolo/getevents?sid=GH6zD4k233nw&lastid=37`. UsaProxy 2 captures the request, queries its event index structure which holds all clients' event logs, and returns a list of new events encoded in an XML document. Figure 3 shows the

```
<?xml version="1.0"?>
<event>
  <sid>8hK8283u23Hh</sid>
  <type>mouseover</type>
  <coord>247,152</coord>
  <id>f</id>
  <dom>abae</dom>
</event>
```

Fig. 3. After polling the proxy for new remote events, each client receives the other user's actions in the form of XML-encoded data. Elements are identified using their position in the DOM tree.

XML representation for a mouseover event. Having received the XML response, the JavaScript code can parse it and replay the event in the user's browser.

Remote Event Representation. A second green mouse pointer is displayed to represent the remote user's pointer. It is implemented by inserting a layer containing the pointer image into the document. Remote mouse movements are directly applied to the mouse layer by assigning it the coordinates specified in the XML event description. As the coordinates are always relative to the hovered-over element, the remote pointer appears in the right spot even if the page is formatted differently in the local browser. In the case of a mouseover event, a hand-shaped cursor image appears instead of the pointer. Additionally, the hovered-over element is underlined in green. Remote scroll bar movements are repeated locally by calculating the local offset of the viewport in the HTML page using the transmitted percentage offset value. The result is assigned to the page offset, which causes the user's page to automatically scroll to the new position.

Value changes in text fields, drop-down menus, radio buttons or check boxes are directly applied to the corresponding field. Furthermore, we provide basic support for AJAX applications by re-executing locally all those event handlers which were also executed in the remote browser as a result of an event like mousedown, mouseup, mouseover or change. As testing of our prototype has shown, this system for re-executing event handler code on all browsers is not reliable for advanced AJAX applications in all cases. A future version of UsaProxy could be extended to support copying the DOM tree of one "primary" browser to the other participants of a shared session.

Putting it All Together: A Web-based Shared Browsing Application. As pointed out in section 3.1, UsaProxy 2 may be launched in several different modes, which are specified using command line arguments. By default, it starts in HTTP proxy mode. In this mode, the proxy settings of clients must be modified to make them send requests to UsaProxy. Additionally, in this mode the software also automatically recognizes whether it is being used as a transparent proxy. Thus, instead of reconfiguring the browsers, the local network gateway can be set up to forward all outgoing HTTP requests to the proxy, and no client reconfiguration is necessary.

Finally, the software can also be deployed in server mode. In this mode, it is installed in front of a particular website. It behaves like a regular web server to clients, but forwards all requests to the actual web server. The browsers of site visitors do not need to be reconfigured. On the server side, only a simple reconfiguration is necessary: Either the original server needs to listen on a different port to allow UsaProxy 2 to

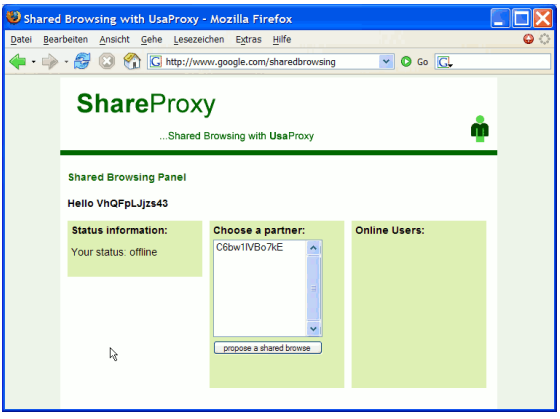

Fig. 4. Overview page for the management of shared browsing sessions. Users can select a partner and send him a proposal to start a shared session.

filter incoming requests on port 80, or the website's DNS entry needs to be pointed to a separate server on which a copy of UsaProxy 2 is running.

Independently of the above modes, UsaProxy 2 may also be configured for a specific shared session mode. With the simple variant – the asymmetrical remote monitoring – users may only view another browser's actions, e.g. any pages that are visited. The browser whose actions are monitored does not poll the proxy for remote events. The more advanced symmetrical shared browsing mode allows two browsers to participate in the same browsing session, where each user can e.g. click on links to visit new pages. In this mode, both browsers poll for new events. The mode of operation is selected on the command line when UsaProxy 2 is started.

In order to start a shared session, one party needs to send a "proposal" which must be accepted by the other party. This process is realized by a session initiation handshake during which both parties verify their mutual acceptance. If UsaProxy 2 is deployed for live support usage, only a simple "Live Support" button needs to be added to the website. In that case, the support staff is provided with an overview web page displaying a list of proposals, each of which was initiated by a click on the support button. For symmetrical usage, both parties can access a similar page with a list of potential session partners – see figure 4 for a screenshot. When a partner is selected and the button "propose a shared browse" is pressed, a pop-up immediately appears in the other browser, asking for the remote user's consent to start a session. Once this consent has been given, either user can simply enter a new URL in the address bar or use other means to visit a new page, and his actions will be replayed in the other user's browser.

After a shared session has begun, each participant's browser window is augmented with the remote user's pointer, and an integrated chat panel may be dragged around and positioned where the users like – see figure 1 for a screenshot. This permits users to directly communicate without resorting to external tools. The chat window is UsaProxy's metaphor for the fact that the shared browsing session is active; if either user closes it (with a click on the "×" in the upper right corner), the shared session is terminated. This way, users can easily distinguish between shared web pages and privately visited pages.

For the purpose of steering another user's browser correctly, it is required that the same content is delivered to it as to the local browser. As identical requests to a server may lead to different server responses, the approach taken by UsaProxy 2 is to record each server response (headers and data) together with a unique page ID. Later, the recorded version of the page is fetched by the second browser via a special request which contains the ID (see figure 2).

In order to permit users to have other browser sessions which are independent of the shared session, the shared session window is assigned the name "sharedsession_UsaProxy" the first time our JavaScript executes. When the JavaScript code executes on later occasions (new page visited, URL entered manually in address line), it can read the name and decide whether it is the shared session window, i.e. whether tracking and logging should be performed.

4 Experimental Evaluation

In order to analyse whether our technology is suitable for the envisioned areas of use, we have performed a number of tests with users who had no prior experience with application sharing over the web, and no relation to the project. The 12 test persons were students of Computer Science (8) and members of staff. Teams of two people had to complete the same three tasks, which were designed both to test whether UsaProxy 2 worked at a technical level and whether it worked for the different application areas. Before the tasks, the participants were given a short five-minute overview of UsaProxy's features and of how to use it to create a shared or monitored browsing session. The tests took place in a lab where several machines had already been prepared for use – a copy of UsaProxy 2 had been started and the browsers had been reconfigured to access the web via the proxy. After the tasks, the users were asked to fill out a questionnaire.

4.1 Looking for a Present Online (Shared Browsing)

The two members of each team could not see or hear each other directly for this scenario, whose objective it was to test UsaProxy's "shared browsing" mode. They were given the following instructions:

After you have started a shared browsing session, visit `amazon.de`. *Try to find an item to give to a common friend of yours as a present. Do this by proposing items to each other and looking at them.*

This task causes a fairly large amount of communication between to the test users. It was considered complete as soon as one of the users put the product into the website's shopping basket.

For most participants, the same observations could be made with regard to their first reactions: Upon discovering their counterpart's mouse pointer and the ability to perform actions on the same web page, they were impressed by the system, expressed that they liked the idea and played around a little, e.g. by chasing their team member's mouse pointer. However, after exchanging a few chat messages and starting to use the website for the given task, this was often replaced by a certain amount of confusion: They had

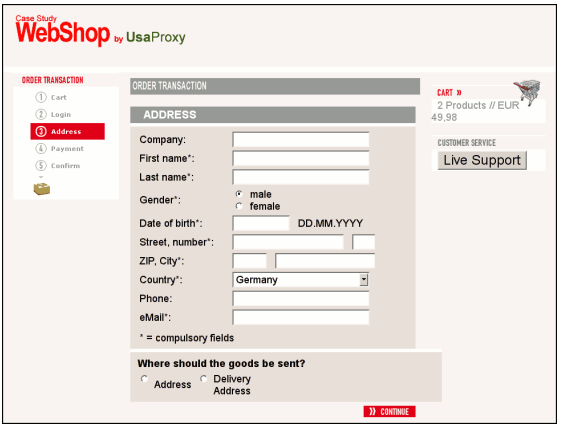

Fig. 5. On the example website, visitors can request immediate help using the "Live Support" button. From that moment, their mouse movements and other actions on the page become visible to the website's live support staff.

not been given any rules as to who should do which things in the browser window. This could lead to conflicts, such as situations where one user was still typing in a search query, but the other user clicked on a link, discarding the semi-complete query. While some teams continued to interrupt each other in a rather chaotic way, others began to develop a "social protocol" – typically, one user would stop performing actions after a while, watch his partner's actions and give comments via the text chat. On at least one occasion, the roles were switched once more, after the active user had written in a chat message that he had run out of ideas.

It is imaginable that with more experience in using the system, and with more complicated tasks, the participants would have developed more elaborate usage patterns. For example, they might have started taking turns for the main interaction and pass control to their counterpart via a chat message. Another alternative would have been to start a separate, non-shared browser session in another window or tab and only to use the shared session when something interesting is actually found.

Despite the problems they had, at the end of the three experiments 11 of the 12 test participants said that they liked the general idea, or that they liked it very much, and only one person said he did not like it. (For all such questions, five options were available in the questionnaire, ranging from "I like it very much" to "I do not like it at all".) However, only about half of the participants said they would actually use this mode of the proxy in practice (6 would tend to use it, 5 would not tend to use it, 1 undecided).

4.2 User Support in a Web Shop (Remote Monitoring)

With this scenario, UsaProxy's "remote monitoring" mode was tested. Again, the two test users were not able to communicate directly. A web page was set up for this task (figure 5), it showed a delivery address form, as found on many shopping websites. The first user was given the following task:

Complete the form on the website. As the delivery address, enter "Kingsgard Dry-Cleaning, Munich main station, 80335 Munich". In case of problems, use the "Live Support" facility of the website.

The second user was told to play the role of a support hotline of the web shop. He was shown how to log into the system and wait for a request for live support. Furthermore, he was given information on how to assist the first user with the form, e.g. to let the user enter "0" in the house number field if the address does not have a house number. During the test, the user filling out the form had to describe his problem to the support hotline: The fields "first name", "last name" and "number" were required by the form, it was not possible to leave them empty. The hotline then gave give hints via the chat only, using text messages to guide the user towards completing the form.

The setup and behaviour of the monitoring feature did not present a problem to the users. In contrast to the first test above, the support hotline could not directly influence the page displayed by the browser, so much time was spent communicating via chat. Using the hints given by the online support, all users were able to complete the form. Unlike with the first scenario, users did not get confused – the text-based chat was a familiar form of communication for them, and they were fine with waiting for instructions on how to solve their problem. Later, eight of the 12 users mentioned in the questionnaire that they would use (or probably use) this kind of live support in practice, three were undecided and one would probably not use it.

This scenario is tailored towards the "server-side" mode of operation of UsaProxy 2: The proxy could be installed on a company's server and visitors of the company website could request help without any changes to their setup, not even a reconfiguration of their browser settings.

4.3 Teaching the Use of a Web Application (Shared Browsing)

The final task used the "shared browsing" mode of UsaProxy 2. Only one test user took part in the test, the second user's part was always played by an operator. The presence of a voice connection (e.g. via VoIP) between the two users was now assumed. The task was as follows:

Your friend has mentioned to you that he found a certain interesting publication with the topic of "why do people blog" or "blog communication". During a shared browsing session, search for it on the web. If you are unable to locate it, let your friend assist you in your search.

The test user first attempted to perform the task, but was unable to do so using the provided information. He then informed the other user (our operator) about this by talking to him. The operator demonstrated the correct solution in the shared browser session by visiting Google Advanced Search, and making the correct settings in the search form. Thus, the "teacher" (operator) was able to observe the mistakes of the "learner" (test user), react to them and guide him towards the right solution. On the other hand, the learner was able to see how the solution was reached. This scenario was well received by the participants, and several persons noted that they liked how they had been able to see all the actions that led to the desired search results.

5 Discussion and Conclusion

In this paper, a detailed concept for the shared use of web applications was introduced, together with a prototype implementation which allows two users to collaborate in web applications. We have successfully conducted an evaluation of the prototype, demonstrating that it is feasable at the technical level and that the browser session sharing features are appreciated by users. Compared to previous work, our approach is minimally invasive, as it does not require software installation on the client or server side, works with most existing web pages, allows shared browsing sessions to span across multiple domains, and does not change the behaviour of web pages beyond the addition of the collaboration features. With our work, we hope to provide the technical infrastructure for further work in the area of web-based collaboration.

The scenarios used in the experiment are only some examples for the potential of application sharing and monitoring. Many others can be imagined, such as annotation of pages (e.g. with virtual Post-Its) or special concepts for online learning and online gaming. The topic of using the monitoring data for usability testing, semi-automated improvements to websites, visualisation of their usage, marketing etc. has already been discussed in detail in previous work [2]. Finally, sessions could be recorded and played back at a later time, and the number of participants in a session could be raised beyond the current limit of two.

As monitoring can be made invisible to the user of a web page (e.g. by running the proxy transparently in front of the server), privacy issues arise. Our current implementation is not capable of hiding the fact that a monitoring session is active (the presence of the chat window makes this apparent at all times), but the technology can obviously be abused to monitor people's behaviour on web pages without their consent. It is the responsibility of anyone using the tracking code to inform users that it is being employed.

While we believe that we have addressed all security issues in our implementation, it should be noted that with the current state of the technology, using a shared browsing session will always make the participants of the session more vulnerable than if they were using the web alone. The most obvious attack vector is that a malicious user directs the shared session to a page which exploits known browser security vulnerabilities. Furthermore, session cookies are shared between the browsers, so "stealing" them is trivial. Thus, we recommend that shared browsing sessions should only be performed by users who trust each other.

In our experience, adding JavaScript on the fly to existing web pages is a very useful technique which has applications beyond those shown in this paper. Current systems and browsers offer enough performance and the needed capabilities for this approach.

The linking of interaction events like clicks to the DOM objects proved to be a very important design decision. Many problems related to different layouts on different browsers and systems are solved by this approach. However, many challenges are still ahead – in particular, fully supporting the sharing of complex AJAX applications, while conceptually feasible, is not easy to implement.

So far, we have decided not to include support for explicit protocols which support passing control over the shared browsing session from one user to another. This is due to the fact that we believe that audio connections are likely to be present in most collab-

orative settings, so social protocols will be the most efficient means for coordination. Thus, the system is not suitable for scenarios where users compete.

Acknowledgement. This work was funded by the BMBF (intermedia project) and by the DFG (Embedded Interaction Research Group).

References

1. Ahuja, S.R., Ensor, J.R., Lucco, S.E.: A comparison of application sharing mechanisms in real-time desktop conferencing systems. ACM SIGOIS Bulletin 11(2-3), 238–248 (1990)
2. Atterer, R., Wnuk, M., Schmidt, A.: Knowing the User's Every Move – User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In: WWW2006. Proceedings of the 15th International World Wide Web Conference, Edinburgh, Scotland (May 2006)
3. Cabri, G., Leonardi, L., Zambonelli, F.: Supporting Cooperative WWW Browsing: a Proxy-based Approach. In: Proceedings of the 7th Euromicro Workshop on Parallel and Distributed Processing, Madeira (P), pp. 138–145 (February 1999)
4. Esenther, A.W.: Instant Co-Browsing: Lightweight Real-time Collaborative Web Browsing. In: WWW2002. Proceedings of the 11th World Wide Web Conference, Honolulu, Hawaii, USA (May 7-11, 2002)
5. Gianoutsos, S., Grundy, J.: Collaborative work with the World Wide Web: Adding CSCW support to a Web Browser. In: Proceedings of the Oz-CSCW'96. DSTC Technical Workshop Series, pp. 14–21. University of Queensland, Brisbane, Australia (1996)
6. Greenberg, S., Roseman, M.: GroupWeb: A WWW Browser as Real Time Groupware. In: CHI 1996 Short Papers: Proceedings of the Conference on Human Factors in Computing Systems, Vancouver, British Columbia, Canada (April 13–18, 1996)
7. Han, R., Perret, V., Naghshineh, M.: WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing. In: Proceedings of the 2000 ACM conference on Computer supported cooperative work, Philadelphia, Pennsylvania, United States (2000)
8. Jacobs, S., Gebhardt, M., Kethers, S., Rzasa, W.: Filling HTML forms simultaneously: CoWeb – architecture and functionality. In: WWW1996. Proceedings of the 5th World Wide Web Conference, Paris, France (May 6–10, 1996)
9. Kobayashi, M., Shinozaki, M., Sakairi, T., Touma, M., Daijavad, S., Wolf, C.: Collaborative customer services using synchronous Web browser sharing. In: Proceedings of the 1998 ACM conference on Computer supported cooperative work, Seattle, Washington, United States, pp. 99–109. ACM Press, New York (1998)
10. Puglia, S., Carter, R., Jain, R.: MultECommerce: a distributed architecture for collaborative shopping on the WWW. In: Proceedings of the 2nd ACM conference on Electronic commerce, Minneapolis, Minnesota, United States, pp. 215–224. ACM Press, New York (2000)
11. Sidler, G., Scott, A., Wolf, H.: Collaborative Browsing in the World Wide Web. In: Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland (May 1997)