Pers Ubiquit Comput (2014) 18:1201–1211
DOI 10.1007/s00779-013-0729-0

ORIGINAL ARTICLE

Real-time collaboration through web applications: an introduction to the Toolkit for Web-based Interactive Collaborative Environments (TWICE)

Oliver Schmid · Agnes Lisowska Masson · Béat Hirsbrunner

Received: 14 January 2013 / Accepted: 25 July 2013 / Published online: 16 October 2013

Abstract The widespread availability of personal mobile devices, combined with the increasing availability of stationary public devices such as large interactive displays, creates new opportunities for computer-supported collaborative work. In particular, these two factors enable the emergence of collaborative scenarios, whether planned or spontaneous, in any location, and previous obstacles to such collaborative settings such as limitations on the number of devices available for use and infrastructure costs can be overcome more easily. As hardware restrictions diminish, the need for software toolkits that simplify the development of distributed collaborative applications allowing for device heterogeneity, true multi-user interaction and spontaneous emergence increases. In this article, we describe the Toolkit for Web-based Interactive Collaborative Environments whose aim is to address these issues. This is done using current standard web technologies extended for real-time application (and structured using specific development guidelines) while ensuring compatibility with the manifold new evolutions in the currently ongoing development of open web platform (HTML5, websockets, etc). While our own work has mainly focused on synchronous co-located collaborative systems (same place/same time), our solution, the technologies used, as well as the concepts that are introduced are easily extendable for remote and/or asynchronous collaboration.

Keywords CSCW · Ad hoc collaboration · Web technologies · Real time · Device heterogeneity

1 Introduction

The widespread availability of powerful user-owned mobile devices, the broad availability of wireless networks and the increasing availability of devices such as large interactive displays installed in public (and especially semi-public) locations bring new opportunities for computer-supported collaborative work, which has traditionally been carried out in rather static and controlled environments in terms of available hardware and software infrastructure (e.g. meeting rooms).

Although, given the above advances, the hardware requirements for spontaneous collaborative scenarios are now much more easily fulfilled, the many potentially heterogeneous devices and device types involved imply several issues in terms of software engineering that need to be addressed. The Toolkit for Web-based Interactive Collaborative Environments (TWICE) was created to support and explore some of the new collaborative interaction possibilities that are emerging and to overcome some of the more complex issues that developers of applications for such scenarios would be faced with. In addition to the usual issues related to ad hoc networks and distributed computing (network latencies, synchronization of application states, etc.), developers would be faced with supporting a heterogeneous set of devices and technologies, and making

This paper is an extract of the PhD thesis in [1] and presents the main concepts of this work.

O. Schmid (✉) · A. Lisowska Masson · B. Hirsbrunner
University of Fribourg, Blvd de Pérolles 90,
Fribourg, Switzerland
e-mail: Oliver.Schmid@unifr.ch

A. Lisowska Masson
e-mail: Agnes.LisowskaMasson@unifr.ch

B. Hirsbrunner
e-mail: Beat.Hirsbrunner@unifr.ch

spontaneous collaborative scenarios using personal devices more accessible and acceptable for end-users by providing minimally invasive installation and configuration while still providing a secure "walk-up-and-use" solution. To help facilitate these tasks, it would be useful to have a toolkit that reduces syntactical differences when developing multi-device and multi-user applications (Fig. 1) compared to their standard single-user counterparts. Our toolkit proposes a solution addressing these issues.

2 Context

The TWICE toolkit evolved from our work on exploring the possibilities of collaboration in multi-user, multi-display and multi-device environments in semi-public settings, and in particular in cases where the collaboration occurs spontaneously. Given this context, we assumed that users would bring and use their own devices for interaction, minimal infrastructure should be expected (due to the spontaneity), and the interaction should not only be collaborative, but also concurrent (users interacting with the same object at the same time).

As already mentioned, many people these days carry with them at least one mobile device (smart phone, tablet, laptop etc), and most, if not all, of these devices are powerful enough to provide the functionalities that would be necessary for complex interaction. Moreover, using personal devices can also have a positive effect in terms of user experience. First, users are familiar with the device, allowing us to leverage the notion of habituation, "When one uses an interface repeatedly, some frequent physical actions become reflexive […]. The user no longer needs to think consciously about these actions. They've become habitual" [2, p. 15], which can be especially important for novice technology users since they would not have to worry about learning new features of a device. Second, the device settings and preferences are already configured to suit the user's needs and interaction style (e.g. user-specific dictionaries for text input corrections), making interaction smoother and more comfortable for them. Third, it allows users to share private content (e.g. pictures, documents, music, etc) easily and directly by transferring it to the collaborative system, and it allows them to directly save any artefacts generated during the collaboration onto the device, making them immediately accessible to the user in the future. Finally, a personal device can act as a natural barrier between public and private space, helping users maintain an awareness of which data are shared within the collaborative session and which is not.

The use of personal devices for interaction also contributes to minimizing the amount of infrastructure necessary, and the cost of providing and maintaining that infrastructure for the owner of the collaborative environment. The fact that an increasing number of semi-public (and public) settings provide wireless networks and many also include some form of large (sometimes interactive) display also contributes to this, and we wanted to leverage these "free" infrastructure resources as much as possible in order to maximize the degree of spontaneity that our system could ensure.

However, several significant technical and usability issues have to be resolved in order to maximize the opportunities that personal devices and minimal infrastructure provide. Given that personal mobile devices vary in their hardware and platform capabilities, heterogeneity of devices can be expected, and thus needs to be supported—otherwise, owners of certain types of devices would be unfairly excluded from the collaborative environment. Moreover, the specificities of the different devices themselves (e.g. interaction possibilities, screen size, external buttons etc) need to be considered, and some device-specific adaptations of functionalities (e.g. providing appropriate layouts depending on the screen size or reacting to device-specific input modalities) may be necessary. Additionally, the use of personal devices immediately implies the need for addressing privacy issues since "careful users concerned about their privacy have to make a tradeoff between the functionality offered by an app and its potential for compromising their privacy" [3, p. 315]. This means that when connecting to the collaborative system, the user should not be obligated to download or launch any custom-built applications (since they might be doubtful of the security and privacy that the application provides) and the user should be assured that their private data will remain private and will not be accessed by or through the collaborative application without the user's explicit permission.

Finally, since in our scenario the infrastructure does not dictate the type or quantity of devices, a system that allows the integration of personal devices should handle scaling issues easily and allow all users, independent of their number, to interact with the system simultaneously. While in some cases simultaneous interaction can be considered as many people connected to a single system but only one being able to directly affect the system at any given moment (even though all connected users could potentially affect the system), the case we wanted to consider in our work was one in which many users can interact with and affect the system at the exact same time (for example in a collaborative editing application where one person is writing the end of a paragraph and another is starting the paragraph that immediately follows, or even editing the just-written paragraph).

As the next section will show, we were unable to find any existing toolkits that we could use which fully satisfied

the requirements of our particular context, which is why we decided to develop the TWICE toolkit.

3 Related work

Over the years, many different toolkits have been created to support the development of collaborative applications. However, most of these toolkits, libraries and APIs were developed to address specific problems that were being explored at the time—communication between devices (e.g. GT/SD [4], Kevlar [5]), the synchronization and conflict handling of events by cloud-based real-time back-end solutions (e.g. Google Drive Realtime API [6], Firebase [7]), the distribution of user interfaces (e.g. GrafiXML [8], Toolkit for Peer-To-Peer Distributed User Interfaces [9], WebSplitter [10]), the extension of legacy applications (e.g. mighty mouse [11], CollabWiseTk [12]), support of multiple users and multiple input devices (e.g. MID [13], MPX [14])—and therefore very few generic toolkits which try to establish a code base for more general support of collaborative scenarios can be found. Examples of more generic toolkits are GroupKIT [15] and GaiaOS [16], although these toolkits, and those like them, tend to depend on specific types of devices or other infrastructure requirements or offer limited flexibility in the types of interaction or collaboration possible. Moreover, many of these task or problem-oriented toolkits handle different technologies or are written in different programming languages, making them difficult to integrate, either with one another solution or into new ones, and since many of them were written to handle pre-defined devices in a specific environment, they are seldom able to adapt sufficiently to dynamically be able to integrate personal devices easily and without the need to install specialized software. In 1996, Roseman and Greenberg found that "Virtually all toolkits […] are just prototypes used to explore different ideas, abstractions, and architectures." and unfortunately, this statement still tends to be valid today.

Although contributing a lot to advanced technologies for collaborative systems, commercial collaborative applications such as Google Docs often fulfil very specific purposes and are usually restricted in their extensibility and therefore in their generalizability for other use. Additionally, most such solutions are often built for telepresent collaboration and usually assume a "one user, one device" scenario in which they allow multiple such users to interact with the same content on a shared platform in real time.

Further complicating the rapid prototyping and development of collaborative applications and systems is the lack of mature software development tools such as integrated development environments (IDEs), testing environments and coding guidelines, like those that exist for prototyping and development of single-user applications. As Roseman and Greenberg point out "Groupware toolkits still have a long way to go to catch up to their single-user counterparts. We look forward to the day when all toolkits, […], will incorporate multi-user features. When that day comes, the artificial distinction between constructing single and collaborative systems will disappear" [15]. Most of the existing approaches to simplify the development of collaborative systems are a trade-off between the complexity needed and customizability, since the abstractions needed for simplification often involve default behaviour which is either impossible or very difficult to customize.

Approaches for "Single Display Groupware" (SDG) (cp. [17]) using a single shared device (e.g. a shared screen) are the focus of several toolkits (e.g. [18]). However, our work necessitates the use of multiple displays and interaction devices, and as a result, we face a distributed system where the applications executed on the different devices have to be kept in a consistent state due to their influence on each other. This type of distributed collaborative system therefore has to handle issues such as application state synchronization, event ordering and messaging, coordination of the devices involved (e.g. to distribute user interfaces) as well as balancing functionalities between devices. It therefore has to coordinate the load of executed code between the different devices while taking into account the differences in their capabilities (performance, suitability of the device for a specific component, etc.).

Finally, most standard software engineering toolkits are oriented towards the development of single-user systems and applications, and in particular the most advanced graphical toolkits are only aware of a single user interacting at a time, limiting their suitability for development of collaborative applications: "Unfortunately, modern window systems are tied to the notion of a single cursor, and application developers must go to great lengths (and suffer performance penalties) to implement multiple cursors" [15]. Only a few specific solutions exist that address this issue (e.g. Windows MultiPoint Mouse SDK [19], MAUI Toolkit [20]), but they are platform dependant, which limits their use in handling a potentially heterogeneous set of devices.

Although research and development of technical solutions for collaborative systems has a long tradition and many researchers have contributed a lot of interesting and fundamental work in this field, we were not able to find a solution which fully suited our requirements—to support dynamic integration of personal devices into the overall system, to allow easy configuration and installation for a real "walk-up-and-use" environment, to support as many different platforms and devices as possible without (significant) adaptation of code and with as little reprogramming as possible, to enable adaptability to specific devices

and their specificities (screen size, input modalities, etc.), as well as to simplify development while maintaining fine-grained customizability of functionalities if needed, in order to meet application-specific needs.

We therefore decided to develop a generic and extensible toolkit which fulfils the above-mentioned requirements or at the very least allows them to be fulfilled by third-party developers.

4 Approach

To create a toolkit which fulfils our requirements, we had to choose an appropriate technology and decide on the distribution of the load of functionalities between the devices for the special case of unpredictable performance capabilities of the devices involved. Additionally, we had to extend existing software engineering concepts or introduce new ones and had to add functionalities to support collaboration using the technology we chose. In the following sections, we present our approach to addressing the main issues we encountered.

4.1 Technology

After evaluating different possible technologies, we found standard web technologies to be the most appropriate technology stack for our solution. Web applications are supported by almost all modern devices which could be considered as useful in a collaborative setting (smartphone, tablet, laptop etc) thanks to their built-in web browsers and wireless network capabilities. They also provide a true "walk-up-and-use" functionality since they can be launched by simply accessing the correct URL, while applications written in native programming languages have to be installed on a device before they can be executed. Standard web technologies have a natural privacy protection mechanism. Since the application runs in a restrictive security and privacy-protected browser sandbox, it is easy to make users aware that when they give permission to the possibly unknown and therefore untrusted collaborative system, and they do so in the same way as they would if they were navigating to a previously unknown web page. For the collaborative system to access extended functionalities such as location information, video and audio resources, etc. users have to explicitly give permission, meaning that they are in full control of what the system can and cannot access.

Despite these advantages, web technologies unfortunately also imply several disadvantages. Due to the security and privacy restrictions of the web application, the set of accessible functionalities is limited. It is therefore not possible to use all potential functionalities of the device when the application is executed within the context of a standard web browser. Other execution environments, and therefore less restrictive sandboxes, like operating systems that are able to execute web applications natively (e.g. Firefox OS) or native wrapper applications (e.g. PhoneGap) are able to extend the set of functionalities of web technologies and can therefore provide ways to work around the mentioned restrictions.

One of the most difficult disadvantages of web technologies to overcome is the reduced set of communication channels. While new web technology specifications provide solutions for true bidirectional communication between servers and clients (through web sockets) and between clients in a peer-to-peer manner (through data channels), legacy browsers require workarounds to establish bidirectional communication through standard HTTP connections such as long polling, iframe streaming, etc. known under the collective term "comet". Although they are not as powerful as natively bidirectional communication protocols, Gutwin et al. [21] have shown that such workarounds can provide sufficient message rates for most collaborative applications and share our belief in "[…] a strong need for better tools in this area—e.g., groupware toolkits that use Web technologies, and development environments for Web applications" [21].

4.1.1 Reuse of APIs

Web technologies, and especially JavaScript, provide functionalities which can be used to overcome the "artificial distinction between constructing single and collaborative systems" [15]. Because of JavaScript's weak typing, existing standard structures such as standard events can be freely extended by introducing additional fields. It is therefore rather easy to provide backward compatibility and to extend the functionality of this language, all while reusing standard APIs.

An example of how this backward-compatible extension of standard APIs works is the way in which multiple mouse pointers can be handled. When introducing multi-pointer functionality, it is important to be able to separate mouse events triggered by the different individual pointers. In JavaScript, this can be done by extending the standard mouse event with an additional attribute "deviceId" which contains a unique value for the device which fires the event and therefore allows the event handler logic to react appropriately. Thus, a standard mouse event handler can be registered which receives events originating from any of the available pointers and the "deviceId" field of the received event can be checked within the handling mechanism.

This behaviour not only allows the reuse of standard APIs such as the mouse event object and the mouse event handler,

but also lets the developer distinguish between native and remote pointers and to react differently to them (e.g. by processing a "click" event on a button only if the event originates from a specific pointer, thus preventing others from selecting the button). Additionally, legacy web applications can be extended for multi-pointer scenarios rather easily because their registered mouse event handlers can be invoked by the additional pointers as well and will therefore react the same way as they would if the native pointer had interacted with them. This enables the application to support all simple mouse events (although more complex mouse gestures such as drag-and-drop which need additional device information to work properly in multi-pointer scenarios might cause unexpected behaviours—cp. [22, 23]).

This reuse of standard APIs allows developers to make use of APIs with which they are already familiar (thus lowering the learning curve of the toolkit) while profiting from new functionalities by using additional methods and fields. Another big advantage is a simplified development and testing process. Since the native mouse pointer behaves in the same way as any other additional pointer, the development and testing for a multi-user environment can be executed just as in a single-user mode using only the default mouse pointer. Multiple pointers can be simulated in one of two ways—by manipulation of the value of the field "deviceId" at runtime (e.g. in debug mode) or by the definition of mode switches (e.g. mouse events which are extended with a specific device identifier when they are triggered by pressing down on a specific keyboard button, simulating the use of another pointer). Thus, the complex set-up of a special development environment with multiple devices and network connections becomes unnecessary, greatly simplifying the testing process.

The example of multiple mouse pointers was chosen to demonstrate the idea of API reuse—the power of dynamic extensions of standard JavaScript objects, and consequently a reduction in the need to introduce new concepts to cover additional functionalities, is one of the central concepts of development using the TWICE toolkit.

4.1.2 *Supporting heterogeneous devices with "deferred binding"*

Although the choice of technology defines the programming language (JavaScript) and the markup language (HTML) in combination with CSS for the visual representation, there was still a choice to work with pure JavaScript, third-party libraries such as jQuery or more structured concepts such as the Google Web Toolkit (GWT). All of these options have the same capabilities, and therefore, our goals could have been achieved using any of them. However, the Google Web Toolkit facilitated the realization of several specific features. In particular, GWT allows the developer to write code in Java and the toolkit then transforms it in a compilation step into JavaScript for the client side. The advantage is not only that the same programming language can be used for the client as well as for the server side (if the backend is written in Java as well) but also that the toolkit provides different versions of JavaScript to handle browser-specific characteristics and interpretations of code.

The separation of code for different browsers is achieved through a mechanism called "deferred binding". This mechanism allows to define different implementations of a specific functionality. The toolkit then creates different versions ("permutations" in GWT terminology) of the application code at compile time. Specified JavaScript logic then decides at runtime which implementation should be loaded and executed. Although originally created to overcome different interpretations of JavaScript by the main web browsers, this mechanism can be reused by defining custom decision logic (cp. Listing 1) and dynamically replacing implementations (cp. Listing 2) as part of a standard GWT module descriptor file.

```
<define-property name="deviceType" values="cursor,
    touch"/>
<property-provider name="deviceType">
  <![CDATA[
      if ("ontouchstart" in window.document.
          documentElement) {
        return "touch";
      }
      else{
        return "cursor";
      }
  ]]>
</property-provider>
```

Listing 1 Definition of a deferred binding property which checks if the device supports touch as an input modality

```
<replace-with class="ch.unifr.twice.CursorComponent">
    <when-type-is class="ch.unifr.twice.MyComponent"
        />
</replace-with>

<replace-with class="ch.unifr.twice.TouchComponent">
    <when-type-is class="ch.unifr.twice.MyComponent"/
        >
    <when-property-is name="deviceType" value="touch"
        />
</replace-with>
```

Listing 2 Replacement of the class MyComponent with CursorComponent or TouchComponent respectively

If the class `MyComponent` is now instantiated with the factory method `GWT.create(MyComponent.class)`, the return value is either `CursorComponent` or `TouchComponent` if the defined property is set to the value "touch" and the device therefore supports touch input.

Because any information which is accessible by JavaScript (e.g. screen resolution, user agent, URL request

parameters, support of HTML5 features, etc.) can be used to distinguish and therefore replace implementations at runtime, this functionality is perfectly suited for fulfilling our requirement of being able to handle device heterogeneity since different implementations of a specific functionality can be substituted as needed.

4.2 Load balancing

The balancing of the executed functionalities over the different devices involved has several effects on the stability of the system, the performance and the network traffic. If a functionality is spread over many devices, the failure of a single device usually has little impact on the stability of the overall system—especially if the devices are mainly executing their own functionalities (and therefore do not execute calculations or other tasks for other devices). We have therefore decided to offload functionalities to the clients and therefore to execute the application code within the web browser. This allows the relaxation of requirements for web server(s) since they have to execute fewer functionalities and can focus on their core tasks such as providing application code and resources (e.g. images, stylesheets, etc.) and acting as a message gateway since the (not yet) possible direct communication between clients, as well as potential management and coordination functionalities (e.g. control over the distribution of user interfaces), depend on the requirements of the specific application. This reduction in functionality is especially important if the system is not executed on powerful standard web server infrastructures as part of huge computer centres but rather on one or multiple dedicated devices in an ad hoc scenario.

The decision to execute the bulk of the functionalities on the client side was also influenced by the improved responsiveness that the device can provide. If an action is triggered on the device by the user, the software can respond immediately without having to send a request to the server first. On the other hand, this requires more complex synchronization requirements between the devices since all devices have their own application states and have to be kept updated about the changes on other devices. The reduction in executed code on the server side implies the execution of more code on the client, which can be an issue since the client devices may be low-performance devices and therefore may not be able to execute complex logic. A separation between different implementations depending on device specificities is therefore not only necessary to address user interface aspects and to support different input modalities, but also to provide variations of implementations which differ in complexity and can be executed on low-performance devices with reduced functionalities while medium- and high-performance devices can profit from their additional computational capacities and provide more advanced functionalities. If the complexity of a functionality cannot be reduced and therefore is not suited for execution on some devices, it should be possible to exclude it from the overall execution without affecting the general integration of the device. We have therefore developed a modular system in which components and their implementations can be replaced or excluded at runtime. The system thus allows the integration of devices into a collaborative setting even if they are not capable of executing all of the potentially available functionalities.

4.3 Eventing and messaging

Because GWT was developed for standard web applications and therefore is not prepared for distributed eventing mechanisms, we have extended the already existing concept of the `EventBus` with remote eventing capabilities. A new implementation of the event bus (which is usually applied as a singleton and on which elements all over the application can register event handlers through which events can be fired) has been developed which takes care of the set-up of bidirectional communication (either through web sockets or comet), serializes the events sent through the event bus, distributes them over devices registered in the same collaborative session and deserializes the remote events which are received from other devices. Additionally, it contains an event ordering mechanism which ensures consistency between the different devices involved.

The provided conflict management by TWICE is designed to handle conflicts on the client side (within the web browser) because of the design goal to minimize the requirements for the servers. The clients make use of the system clock of their providing server to obtain a common time base and therefore to define a global order of the events on which every device agrees. Thus, conflicts (unordered events) can be recognized and are addressed depending on their type. For events which cannot be undone (e.g. because they affect third-party systems), the system applies locking strategies to prevent conflicts from arising (e.g. delays the processing of the event until a conflict-free execution is guaranteed). Undoable events are rolled back and discarding events, which fully replace others of their kind (e.g. mouse pointer position updates), are simply ignored if newer information has already been received. The development of a custom-distributed event handling strategy was necessary, since solutions such as Google Drive Realtime [6], Firebase [7] and others are usually based on cloud infrastructures and are therefore not suitable for collaborative scenarios including devices which are not connected to the internet (but might instead be connected only to a local ad hoc network). Since it is an explicit design goal of TWICE to address low-capability and low-infrastructure scenarios, we have decided to

provide our own eventing and messaging mechanism with fewer requirements on the infrastructure. However, infrastructure-supported services can easily be integrated into the toolkit by creating alternative implementations of the `EventBus` which can replace the eventing mechanism provided with TWICE through the "deferred binding" mechanism at runtime whenever the collaborative scenario and its devices fulfil the service's requirements.

4.4 Multi-user support

One of the main features that a collaborative system has to provide is simultaneous shared access to a single resource for multiple users and multiple devices. This functionality is often established by providing multiple pointers, for example on a shared screen, or as remote text input possibilities.

4.4.1 Multiple pointers and remote text input

To enable multiple pointers within web technologies, different problems have to be solved for both the shared device (the one the pointers are actually living on) and the controller instances (the input devices used to move the pointer, to click, etc.). The implementations of the pointer controller instances differ depending on the input modalities of the device.

Touch-capable devices are equipped with a touch-sensitive area which works in a way that is similar to a touch pad on a notebook. Here, the pointer can be controlled by dragging the finger over the area (cp. Fig. 2 device B). Clicks are triggered by simple tapping on the screen, and drag-and-drop can be achieved by holding the finger on the screen for a short moment, until the display of the device indicates that it is in dragging mode. A draggable element on the shared screen can then be repositioned by moving the pointer and can be released by a single tap.

In contrast, on cursor-oriented devices, the relative position of the device's standard mouse is captured when

Fig. 1 TWICE—a toolkit for the development of collaborative systems based on web technologies

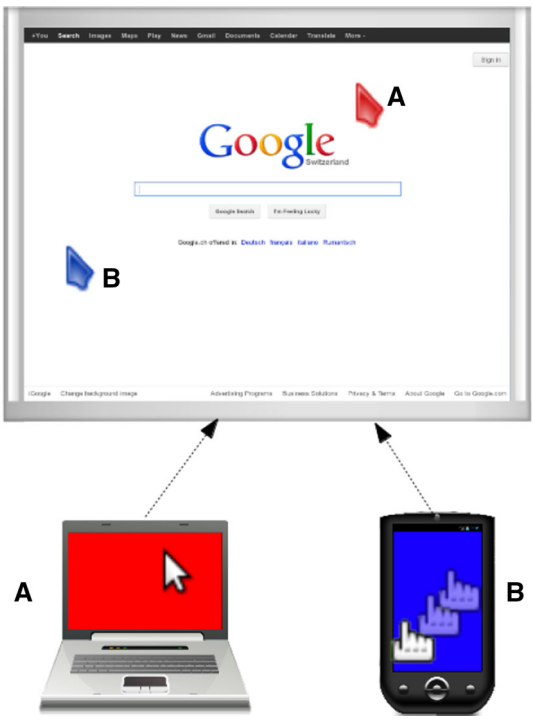

Fig. 2 Remote pointer control on a shared screen with a pointer (*A*)- and a touch (*B*)-based device

hovering over the sensitive area (cp. Fig. 2 device A) and is translated to the absolute coordinates depending on the screen resolution of the shared device. In addition to mouse movements, clicks and mouse down and mouse up events are captured and sent to the server which then pushes the information to the shared device through a web socket connection.

The shared device receives the command messages from its clients, assigns a mouse pointer (rendered as an HTML element), executes the action according to the commands and fires standard mouse events extended with the information about the mouse pointer (by adding a specific device identifier to the event).

Different implementations are also required for the text entry mechanism. Since most touch devices (such as smart phones and tablets) do not have a physical keyboard, the software keyboard needs to be triggered using only web technologies. This was not a trivial issue to solve since web applications do not have permission to request the native keyboard. Therefore, a little trick had to be used—a standard HTML textbox was redesigned to look like a button. When this textbox is touched, the focus is set on this widget, which triggers the appearance of the software keyboard (because the user is expected to enter text). By hiding the textbox (moving it outside of the visible area), we were able to provide a mechanism which is indistinguishable from native triggering of the keyboard (from the user's perspective), and the key press events can be redirected to the shared device.

4.4.2 Multi-focus widgets

Last but not least, multi-user support involves the issue of multi-focus. Because standard web browsers use standard widgets (e.g. buttons, text boxes, checkboxes, etc.) of the graphical toolkit of the programming language they are implemented in, none of them is prepared for multi-user use. The simultaneous focus of multiple devices on the same widget (e.g. multiple text cursors in the same text box or in different textboxes within the same application) is an issue that requires special handling at the functional level and also at the level of the visual representation. We have therefore implemented multi-focus replacements, for example for a text box based on the HTML5 canvas (although proof-of-concept implementations have been developed with legacy HTML elements as well), which contain the handling logic and are able to manage the visual representation of the text cursors so that the current entry position of the different devices can be identified (cp. [23]).

4.4.3 Collaborative web browsing

Based on the mechanisms of multiple pointers and multi-focus described above, we were able to provide a solution for simultaneous collaborative web browsing on a shared screen (cp. [23]). Here, we injected additional JavaScript functionalities into a third-party web page (e.g. http://www.google.com) when accessing it through a forward proxy server. This allowed us to control the web page with multiple pointers and to replace standard components (such as textboxes) of the original web page with our own multi-focus components to provide a full set of multi-user functionalities. Thanks to the backward compatibility achieved with the reuse of standard APIs, we were even able to dynamically extend third-party web pages which were developed for single-user use and support all of the standard functionalities provided by these "legacy" web applications without needing to make any code changes. Tests with major web portals (e.g. Google, Yahoo and Wikipedia) showed that we could extend single-user web applications dynamically for multi-user scenarios with default behaviours and therefore were able to noticeably diminish the differences between these modes in terms of software engineering.

As discussed in [23], the use of a proxy server involves security issues when interacting with web pages which are accessed through the secured HTTPS protocol: since the proxy server mediates the secured connection to the real server, the ensurance of confidentiality between the client and the back-end server is broken. Aware of this problem, we are convinced that the context of a collaborative web browsing session relativizes this issue because we can not see a realistic use case of transferring sensitive information to a web page on which multiple users are interacting simultaneously. Nevertheless, reasonable actions can be taken to make the user aware of the potential security threat by displaying warning messages whenever a web page is accessed with HTTPS through the proxy server. Additionally, since our forward proxy solution does not require configuration of the client, users can still interact with web pages through secured connections alongside of the collaborative session by simply accessing the original web-page (without the forward proxy prefix) in another browser tab or window.

4.5 Dynamic layouts and managed modules

Adaptive-distributed user interfaces which are required for multi-display groupware have to solve the issue of splitting up a user interface into multiple parts, distributing these parts across the available devices and ensuring that the communication between the different parts allows them to interact with and influence each other. In addition to the distribution logic which defines which part is sent to which device, issues of dynamic layout systems and lifecycle management have to be considered to ensure optimal resource use. Since it is difficult to predict which and how many components will be executed on a specific device, the main layout should be scalable and flexible enough to integrate a varying number of sub-elements. The design of such a layout mechanism depends on the actual device specificities. While a bigger display (such as a notebook screen) could easily run multiple components side by side in a split layout, it is less comfortable to have multiple elements visible on a small screen such as that of a smart phone. We therefore developed different layouting mechanisms for small- and large-screen devices. While on larger screens, a new tab (which can be rearranged by drag-and-drop (cp. Fig. 3 upper right)) is created for every component, the small screen version contains a single component on the screen at any one time and contains an additional menu button on the top left which, when selected, shows a list of available components (cp. Fig. 3 lower left).

To be able to separate the code and to save resources, especially on low-performance devices, we have extended the module concept of GWT. Every (user interface) component is defined as a single module which contains—inspired by OSGi—the necessary methods for the lifecycle management of the module (by the provision of start and stop methods). This extension of the module concept of GWT allows us to improve the resource use of the application. The code of the components is loaded in a lazy manner, meaning that it is only loaded (and therefore only occupies resources) at the moment when it is

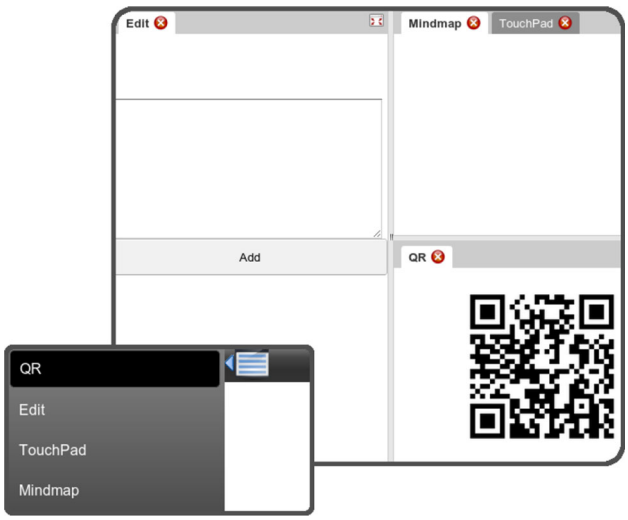

Fig. 3 Different dynamic layouts for big (*upper right*) and small (*lower left*) screens

accessed for the first time. Additionally, if the user navigates away from a component (e.g. switches to another component in the case of the small screen component or the tab becomes deactivated in the large screen layout), the stop method of the component is invoked and occupied resources can be released. These mechanisms allow us to reduce the complexity and occupied resources at runtime and therefore optimize the performance of the executed application.

5 Evaluation

The theoretical and implemented concepts we have described have been evaluated in a number of different situations (cp. [1]). The toolkit was used to develop multiple experiments to explore usability and group dynamics in collaborative scenarios (in collaboration with psychologists), and the software developed with our toolkit has proven to be reliable and stable, and shows the feasibility of integrating a very heterogeneous set of devices.

In open interviews with other developers who have used the toolkit, we examined how easy the newly introduced functionalities were to use. In particular, the reuse of existing GWT APIs was mentioned as lowering the learning curve for familiarizing oneself with the toolkit, and the possibility of implementing an application as if for a single-user context and then extending it for multi-pointer use with only a few lines of code was seen as a big advantage.

To ensure functionality in real-world use and as a practical scalability test, we developed a mind-map application with remote pointer and text input capabilities and ran an experiment in the context of a high school class with 13 students and one teacher. The students were asked to bring their own mobile devices with them, leading to a set of expectedly heterogeneous input devices. Although most of the devices were different versions of iPhones (from 3G to 4S), Android-based phones from different manufacturers and in different versions were also used, as were tablets. The client devices were all interconnected through a standard wireless LAN provided by a low-budget WiFi router and a server running on a standard notebook.[1] We experienced no performance issues within the network, even when all participants controlled their remote pointers simultaneously (we measured a maximal network throughput of 85 KB/s which is far below the capacity of a standard wireless LAN). Nor did we remark any performance issues at the server level. Therefore, we believe that the limit of the number of simultaneous users, even in a very simple setup, is far above 14. These performance results were confirmed by the impressions of the users who indicated (through a survey) that they did not feel that the system had delayed reactions to events.

6 Discussion

Although our development focused on synchronous co-located scenarios involving a stationary large display and an architecture with a standard wireless network and a single pre-installed server providing the application for a varying number of heterogeneous devices, our toolkit as well as the technologies involved can also be used in other settings such as purely ad hoc cases involving no infrastructural support (e.g. with only user-owned devices), asynchronous and/or remote collaboration and different (more redundant) system architectures (e.g. server clustering, dynamically added server resources from the cloud, etc).

In addition to standard devices such as computers, smart phones and tablets, we were also able to show that the system works with other types of devices such as e-readers and game consoles (cp. [1]). This gives a good indication that TWICE can support any device which is able to run a web browser with JavaScript and consequently that new generations of TV sets and photo cameras, as well as future device types with internet connections, will also be supported.

TWICE provides the basic infrastructure to build collaborative applications involving multiple users and multiple devices. The complexity of such applications is indisputably high and the adaptation of the toolkit to rapid and ongoing technological advancements is a huge

[1] HP Compaq 6730b, 4GB RAM, Intel Core2 Duo CPU P8800 (2 × 2.66 GHz) running Ubuntu with Kernel 3.2.0.

challenge. Some components of the toolkit might become obsolete due to new approaches and concepts originating from industry and academics, while others might gain in importance if new requirements arise or the toolkit is applied in other contexts than it was originally intended for. One of the main ideas of TWICE is to extract the commonalities of collaborative applications, to name them and to hide their complexity behind generic interfaces to make the manifold existing and not yet existing approaches and solutions replaceable and interchangeable and capable of adapting the application to the most appropriate approaches available for a specific type of collaborative setting at runtime.

Since TWICE applications are based on web technologies, they also suffer from the technologies' disadvantages: drawbacks such as the lack of direct hardware access, reduced performance as well as the required adaptation to a very heterogeneous set of devices are applicable to TWICE applications just as they are to any other web application. However, increasing performance of web browsers, ongoing standardization of APIs (video camera, audio, sensors, etc.) and the integration and extension of web technology functionalities as part of operating systems (Chromium OS, Firefox OS) give positive indications for further development and increasing capabilities of this type of technology. Although TWICE tries to address the main issues that arise when developing collaborative systems, there are still many issues which have not yet been solved, and the provided solutions could be further improved. Some of the provided components are well-performing proof-of-concept implementations which should be optimized in terms of performance, features, as well as visual appearance for productive use. Additionally, further evaluation of the validity of our technology choice, architectural concept and developer guidelines would provide more founded knowledge about the current shortcomings of the toolkit and give hints where further improvements would be most beneficial.

However, we are convinced that there is a need for a toolkit like TWICE which provides basic functionalities for the development of collaborative systems and which allows to focus on very specific topics in this interesting and rapidly growing area. We strongly believe that the big complexity involved in the area of real-time multi-user and multi-device systems can only be handled if experts from different fields can focus on their specific main topics while profiting from continuously integrated achievements and improvements coming from other research areas, as well as from the standard functionalities provided in the toolkit itself. With the release of the source code of TWICE under a very permissive license, we hope to improve and to extend the toolkit through the creation of an active community of users and contributors.

7 Conclusion

Although heterogeneity of end-user devices is more widespread than ever, we were able to show that it is possible to integrate most of these devices into a distributed collaborative system using currently available technologies. In our solution, we not only addressed the technical feasibility of multi-user systems but were also able to integrate privacy protection mechanisms and "walk-up-and-use" functionalities. Although the use of standard web technologies for real-time collaborative applications involves several issues such as the lack of bidirectional communication channels (unless websockets are supported by the browser), we have shown that most of these issues can be overcome and that the advantages of web technologies (widespread support, configuration and installation free use, etc.) outweigh the restrictions involved. We have shown concepts to reduce the complexity of development of multi-user applications compared to their single-user counterparts, ways of extending legacy and even third-party web applications with multi-user capabilities, introduced structures to be able to address device-specific characteristics by the dynamic partial replacement of code ("deferred binding") and to simplify the messaging and eventing between devices, and presented basic functionalities (multi-user support, dynamic layouting) required for the development of multi-display collaborative applications. We were also able to successfully evaluate our toolkit in many different scenarios, using it for the development of exploratory applications as well as applications used in real-world conditions applied in a classroom setting with 14 users.

Although we do not claim that our toolkit solves all issues involved in the development of collaborative applications, we believe that we have built a solid base for further development of solutions for specific issues arising in this domain. We strongly believe that all of the important work that has been done over the years by many researchers in the field of collaborative systems needs to be integrated into an open structure based on standardized, future-safe, extensible and platform-independent technologies to simplify the development of collaborative applications and therefore to improve the way in which traditional collaboration can be supported by technical means in general.

References

1. Schmid O (2013) TWICE - a toolkit for web-based interactive collaborative environments. PhD. thesis, University of Fribourg

2. Tidwell J (2010) Designing interfaces. O'Reilly Media, Inc., Sebastopol
3. Chia PH, Yamamoto Y, Asokan N (2012) In: Proceedings of the 21st international conference on World Wide Web, pp 311–320. http://dl.acm.org/citation.cfm?id=2187879
4. de Alwis B, Gutwin C, Greenberg S (2009) In: Proceedings of the 1st ACM SIGCHI symposium on engineering interactive computing systems. ACM, New York, EICS '09, pp 265–274. doi:10.1145/1570433.1570483
5. Huang Q, Freedman DA, Vigfusson Y, Birman K, Peng B (2010) In: Proceedings of the ACM/IFIP/USENIX 11th international conference on middleware, Springer, Berlin, Middleware '10, pp 148–168. http://dl.acm.org/citation.cfm?id=2023718.2023729
6. Google Drive Realtime API. http://developers.google.com/drive/realtime/21June2013
7. Firebase. http://www.firebase.com/21June2013
8. Michotte B, Vanderdonckt J (2008) In: Fourth international conference on autonomic and autonomous systems, ICAS 2008, pp 15–22. doi:10.1109/ICAS.2008.29
9. Melchior J, Grolaux D, Vanderdonckt J, Van Roy P (2009) In: Proceedings of the 1st ACM SIGCHI symposium on engineering interactive computing systems. ACM, New York, EICS '09, pp 69–78. doi:10.1145/1570433.1570449
10. Han R, Perret V, Naghshineh M (2000) In: Proceedings of the 2000 ACM conference on computer supported cooperative work. ACM, New York. CSCW'00, pp 221–230. doi:10.1145/358916.358993
11. Booth KS, Fisher BD, Lin CJR, Argue R (2002) In: Proceedings of the 15th annual ACM symposium on User interface software and technology. ACM, New York, UIST '02, pp 209–212. doi:10.1145/571985.572016
12. Lavana H, Brglez F (2000) In: Proceedings of the 7th conference on USENIX Tcl/Tk, vol 7. USENIX Association, Berkeley, CA, USA, TCLTK'00, p 4
13. Hourcade JP, Bederson BB (1999) Architecture and implementation of a java package for multiple input devices (MID). In: Technical Reports from UMIACS. http://hdl.handle.net/1903/100
14. Hutterer P, Thomas BH (2007) In: Proceedings of the eight Australasian conference on user interface, vol 64. Australian Computer Society, Inc., Darlinghurst, Australia, AUIC '07, pp 39–46. http://dl.acm.org/citation.cfm?id=1273714.1273721
15. Roseman M, Greenberg S (1996) ACM Trans Comput Hum Interact 3(1):66–106. doi:10.1145/226159.226162
16. Roman M, Hess C, Cerqueira R, Ranganathan A, Campbell R, Nahrstedt K (2002) IEEE Pervasive Comput 1(4):74. doi:10.1109/MPRV.2002.1158281
17. Wallace JR, Scott SD, Stutz T, Enns T, Inkpen K (2009) Personal Ubiquitous Comput 13(8):569–581. doi:10.1007/s00779-009-0241-8
18. Tse E, Greenberg S (2004) In: Proceedings of the fifth conference on Australasian user interface, vol 28. Australian Computer Society, Inc., Darlinghurst, Australia, AUIC '04, pp 101–110. http://dl.acm.org/citation.cfm?id=976310.976323
19. Website for the Windows MultiPoint Mouse SDK. http://www.microsoft.com/multipoint/mouse-sdk/13September2012
20. Hill J, Gutwin C (2004) The MAUI toolkit: groupware widgets for group awareness. CSCW 13(5–6):539–571. doi:10.1007/s10606-004-5063-7
21. Gutwin CA, Lippold M, Graham TCN (2011) In: Proceedings of the ACM 2011 conference on computer supported cooperative work. ACM, New York, CSCW'11, pp 167–176. doi:10.1145/1958824.1958850
22. Bowie M, Schmid O, Lisowska Masson A, Hirsbrunner B (2011) In: Proceedings of the ACM 2011 conference on Computer supported cooperative work. ACM, New York, CSCW'11, pp 609–612. doi:10.1145/1958824.1958926
23. Schmid O, Lisowska Masson A, Hirsbrunner B (2012) In: Proceedings of the 4th ACM SIGCHI symposium on engineering interactive computing systems. ACM, New York, EICS'12, pp 141–150. doi:10.1145/2305484.2305508