Integration of Mobile Devices in Collaborative Web Applications

Philipp J. Anger

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juli 2013

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, July 1, 2013

Philipp J. Anger

Contents

Kurzfassung

Der Markt wurde über die letzten Jahre hinweg mit einer Unzahl technologisch ausgereifter mobiler Geräte wie Smartphones und Tablets angereichert. Dieser Trend lenkte den Fokus des Web zunehmend auch in Richtung mobiler Geräte. Web-Entwickler stehen seither vor der Herausforderung, sich an diese florierende Umgebung anzupassen und die dadurch entstehenden Möglichkeiten zu nutzen. Dies führte zu gut an die mobile Umwelt angepassten Web-Applikationen, allerdings kaum zur Integration der mobilen Geräte in Web-Applikationen und die damit verbundene Ausschöpfung deren vollen Potentials. Web-Applikationen könnten signifikant durch die neuen Interaktionsmöglichkeiten verbessert werden, welche sich durch die Fähigkeiten der Geräte und die Anwendung mobiler Webtechnologien ergeben.

Neben der Einführung in den aktuellen Stand mobiler Web-Technologien stellt diese Arbeit einen konzeptionellen Ansatz zum Zugang und zur Verarbeitung multimedialer Kontextinformation innerhalb einer kollaborativen Web-Applikation vor. Insbesondere durch Verwendung traditioneller Web APIs basierend auf HTML, JavaScript und CSS. Mittelpunkt des konzeptionellen Ansatzes ist ein architektonisches Schichtenmodell zur Separierung der Applikationslogik und des geräteabhängigen Kontexts, welches eine plattformübergreifende Integration erleichtern soll. Zusätzlich wird der konzeptionelle Ansatz als Machbarkeitsnachweis in eine prototypische Implementierung integriert und evaluiert.

Die vorgeschlagene Integrationsarchitektur fördert dabei eine strukturierte und webbasierte Entwicklung von kollaborativen Web-Applikationen, um das volle interaktive Potential mobiler Geräte im Web zu nutzen.

Abstract

Intelligent mobile devices – like smartphones and tablets – have been rapidly emerging in the market within the last few years. Thus, they are gaining more and more importance to the ubiquitous Web and face developers with the challenge to adapt to this thriving environment and leverage its arising possibilities. This has led to well adapted Web applications rather than an integration of mobile devices in Web applications and therefore the utilization of their full potential. Web applications could be enormously enriched by the new forms of interaction constituted by mobile device capabilities and mobile Web technologies.

Along with an introduction into the state of the art of mobile Web technologies, this document introduces a conceptual approach for accessing and processing rich contextual information within a collaborative Web application. The integration thereby uses only traditional Web APIs based on HTML, JavaScript and CSS. Key aspect of the approach is a layer-based architecture to separate the application logic from the device's context in order to facilitate the cross-platform integration. In addition, the conceptual approach is integrated and evaluated as a proof of concept in a prototypically implemented application.

The proposed integration architecture leverages structural web-based development of collaborative Web applications to utilize the mobile devices' full interaction potential in the Web.

Chapter 1

Introduction

Mobile devices are becoming more and more important in todays society due to their rapid technological development. Smartphones, tablets and PDAs expose a noticeable demand for full accessibility, seamless functionality and instant information sharing across platforms. Developers therefore aim to satisfy these demands by developing native-, hybrid- or Web applications, whereby this document targets Web applications using traditional Web technologies. Especially collaborations profit from the integration of mobile devices in Web applications due to their need for accessibility and information sharing. Within this document, the state of the art of mobile device integration in Web applications is analyzed and a novel integration approach is presented and evaluated.

1.1 Problem Statement

As the mobile environment has barely been accessible from within a Web browser for a long time, native implementations have been the most powerful way to access mobile context information. Therefore, mobile devices could not be fully integrated into Web applications using Web APIs such as JavaScript and HTML. As a result of these circumstances, mobile devices were mainly handled as smaller screen environments with minor additional capabilities like sensing touch events or defining the device's location using its GPS sensor.

Recent developments in mobile Web APIs have provided the ability to access and use a wide range of sensors and interaction capabilities through mobile Web browsers. Although Web applications could be enormously enriched by the new forms of interaction constituted by mobile device capabilities and mobile Web features, most applications still use only fractions of the given possibilities. Particularly collaborative online applications could benefit from the cross-platform integration to overcome platform limitations and facilitate communication by rich-media content.

1.2 Goals

Based on the problems mentioned before, the lack of mobile device integration and consequently its use of interaction possibilities, this document should focus on answering the following scientific hypothesis:

> "How can mobile devices be integrated in collaborative Web applications considering their context information and their specific interaction capabilities?"

In order to be able to answer this scientific hypothesis, one essential goal pursued is to understand and reflect the currently given technological possibilities for developers to access and process mobile context information.

Understanding the current state of the art is also fundamental for achieving the main goal pursued with this document: introducing and evaluating a conceptual approach to utilize the full potential of mobile devices integrated in Web applications. This includes the presentation, discussion and in-depth analysis of the integration approach upon certain requirements.

Furthermore, a prototypical real-world application should be developed to evaluate the applicability of the proposed architecture and its feasibility from a developing point-of-view. In addition to the evaluation, future work should be shortly presented and discussed.

1.3 Structure

This document is structured into several chapters improving understanding and readability of the scientific work along with its presented approach. Starting with technical background information and state of the art, the integration approach is presented, followed by a description of the prototypical application and concluding with a detailed evaluation and discussion.

Chapter 2 provides some fundamental technical background information and terms of the research field. The information is divided into related sections for Web applications, mobile devices and communication.

Chapter 3 introduces and reflects the current state of the art in the research fields of cross-platform content adaption, integration and migration. Furthermore, some real-world applications and projects are presented.

Chapter 4 presents the proposed mobile integration architecture. The main approach is presented with its significantly important requirements for the approach to be applicable as well as by discussing each layer individually upon its functionality.

Chapter 5 discusses and illustrates the developed prototypical application including its architecture, implementation and user interface.

Chapter 6 evaluates the developed prototype and the conceptual approach upon its requirements and the applicability in the real world.

Chapter 7 concludes the document with outcomes of this work.

Chapter 2

Technical Background and Disambiguation

The World Wide Web – or simply the Web – is a thriving environment where new technologies and terms are frequently presented and known technologies are quickly outdated or updated. This chapter provides technical background information about relevant technologies and important terms, profounding for the understanding of this work. Even though, it is supposed that the reader has a certain understanding and knowledge of the technologies and processes in the ubiquitous Web.

2.1 Related Terms

As this work focuses on the specific research area of the Mobile Web, it is essential to provide a presentation of related terms before discussing the research topic in the following chapters.

2.1.1 Web Application

The term *Web application* describes an application that uses a Web browser, such as Chrome, Firefox, Opera, Safari or Internet Explorer, as a client[1] to make functionality accessible to users. This type of application differs from traditional *Web sites* on how and which tasks can be accomplished. Web applications emphasize dynamic tasks like buying products, transferring money or making reservations, whereas traditional Web sites are content oriented and facilitate browsing and consumption of static information rather than working with dynamic data [22].

[1] Clients are requesting resources from a server and represent the interface between an application and its users, for example a Web browser displaying Web pages.

2.1.2 Collaborative Web Application

A *collaborative Web application* (CWA) describes an application that provides a centrally shared space for multiple users to work together towards achieving a common goal. The user-base is often referred to as a community, working group or team. Compared to traditional Web applications, a CWA has higher accessibility regarding the extent to which read/write/create/delete permissions are extended to the users. Moreover, a CWA's content is driven by the user-base and the hosts primary function is to facilitate the information sharing process. Key aspects to classify a CWA are the barrier-to-entry (how a user needs to verify access to the application), accessibility (permissions to manipulate content) and moderation (how a certain standard of the shared information is upheld) [23].

2.1.3 Web-Based Integration

Web-based integration in the context of mobile device integration in Web applications describes the technologies used to access and process the device's information. Such technologies in this case are traditional Web development technologies such as JavaScript, HTML and CSS. This work is focused on the current state of the art and possibilities of existing Web standards and their technologies with regards on the accessibility of mobile sensors and rich-media content, such as video, audio and image files, as well as the sharing of content between multiple users across various platforms.

2.1.4 Client-Side Web Applications

Web applications have been developed on the server-side ever since due to a much higher capability, consistency and performance of enterprise-class[2] servers in comparison to Web browsers and devices on the client-side. Nevertheless, such applications depend on how fast communications between a server and its clients can be handled. This strategy is now changing. Over the past few years, browsers have started to improve their implementations of Web technologies (especially JavaScript engines like the V8[3]) and Web standards, which enables Web developers to build larger and more demanding applications on the client-side with less dependency on the servers. Web applications are sometimes mistakenly also referred to as *Web apps*, similar to mobile apps[4] which are more like robust stand-alone applications living outside the Web browser ([8], [11]).

[2]Widely open, compatible and reliable platform.
[3]More information on the V8 engine can be found on https://code.google.com/p/v8/.
[4]Software applications built to run on mobile devices.

2.1.5 Mobile Web

The *Mobile Web* refers to browser-based applications, developed especially for mobile devices. In these days of soaring numbers of mobile devices entering the market, Web developers are faced with a whole new kind of interactive devices. This challenging development opens up billions of new devices to the Web on the one hand, but also urges the need for new usability patterns, new knowledge and abilities on the other hand. The new innovative wireless devices are providing a higher quality of life for today's users, therefore the Web and its developers are focusing more and more on mobile Web applications ([11], [7]).

2.1.6 Mobile Device

A mobile device can basically be categorized as such if it is portable, personal, a companion, easy to use and connected to the internet [7]. The market for mobile devices and especially mobile phones is bursting, for example with approximately 5,962,000,000 mobile-cellular telephone subscriptions at the beginning of 2012[5] (about 85% of the world's population). A huge developing process goes along with this trend. Thereupon this work is focusing on the shining stars of this progress: Smartphones and Tablets. This section introduces relevant terms in the mobile world and new possibilities for developers.

2.1.7 Context-Awareness

According to Hsu in [12], the term context awareness was first mentioned in 1994 describing the user access on information about their current context, including location, time, identity, activity and preferences. Over the past few years, the Internet has greatly changed our way of sharing resources and information. Especially sharing information through mobile devices has become very popular. Mobile devices equipped with new technologies, such as location sensors, created the ability to develop multimedia systems by gathering the desired context information from the user's environment [4].

2.2 W3C Standards

> "The W3C mission is to lead the World Wide Web to its full potential by developing protocols and guidelines that ensure the long-term growth of the Web. ... W3C's vision of One Web."[6]

With this quote the *World Wide Web Consortium* (W3C) clearly describes its mission and why standards are so important for future developments.

[5] http://www.itu.int/
[6] http://www.w3.org/Consortium/mission

WebW3C's *Mobile Web Initiative*[7] is currently emphasizing developments for mobile Web applications and the adaption of Web applications to as many kind of devices as possible. Particularly a deepening of integration on hosting devices for Web applications should be granted trough specified standards and APIs. These processes are essential and trend-setting for the development of mobile Web applications. The following sections present some of the most relevant and already widely supported APIs published by this initiative, referring to [28].

2.2.1 HTML5 video element

This API is used for playing videos and audio files with captions. The `<video>` element has several attributes including poster (display a representative frame image when no video is playing), mediagroup (link multiple media elements) or controls (provide default media controllers). The element can load a file of the format MP4, WebM or Ogg via its `src` attribute. As long as not all browsers support the same video formats, it is recommended to use the cross-browser compatible version with alternative sources (`<source>`), as shown in the following example:

```
1 <video width="320" height="240" controls autoplay>
2   <source src="video.mp4" type="video/mp4" />
3   <source src="video.webm" type="video/webm" />
4 </video>
```

2.2.2 HTML5 audio element

The HTML5 `<audio>` element represents a sound or audio stream. It has similar attributes to the video element, such as mediagroup, controls and autoplay (automatically play audio as soon as possible). In addition to the traditional HTML tag initialization, as shown in the code example, the element can also be created and processed as a JavaScript object via the constructors `new Audio()` or `new Audio(src)`, where `src` is the URL to a valid audio file of the format MP3, WebM, Ogg, WAV or AAC.

```
1 <audio controls>
2   <source src="audio.mp3" type="audio/mp3">
3   <source src="audio.oga" type="audio/ogg; codecs=vorbis">
4 </audio>
```

2.2.3 HTML Media Capture

This API defines an HTML form extension of the `<input>` element that facilitates user access to a device's media capture mechanism, with the camera

[7] http://www.w3.org/Mobile/

or the microphone, from within a file upload control. Two attributes are introduced with this API to the input element: `accept` (MIME type) and the boolean `capture`. The following example captures any kind of audio file:

```
1 <input type="file" accept="audio/*" capture>
```

2.2.4 HTML Canvas 2D Context

The specification defines the 2D Context for the HTML5 `<canvas>` element, which provides a surface to draw and manipulate graphics. All objects have to be created and manipulated with JavaScript on the context object, which is returned by the `getContext()` or `getContext(''2d'')` method. The example shows how to draw a simple rectangle in the 2D context.

```
1 <script>
2   var canvas = document.getElementById("myCanvas");
3   var context = canvas.getContext("2d");
4
5   // x=10, y=5, width=120, height=120
6   context.fillRect(10, 5, 120, 120);
7 </script>
```

2.2.5 Touch Events Specification

This specification defines a set of low-level events for touch interaction with a touch-sensitive surface. These events handle one or more points of contact, as well as changes of contact. The most important events are `touchstart`, `touchmove` and `touchend`, with event action information, such as the target DOM element (`target`), touching fingers (`touches`, `targetTouches`) and basic touch information (coordinates, radius and angle). The following example drags a specific element on touch, if only one finger touches the element.

```
1 <script>
2   var objectToDrag = document.getElementById("draggableElement");
3   objectToDrag.addEventListener("touchmove", function(event) {
4     if (event.targetTouches.length == 1) {
5       var touch = event.targetTouches[0];
6       objectToDrag.style.left = touch.pageX + " px";
7       objectToDrag.style.top = touch.pageY + " px";
8     }
9   }, false);
10 </script>
```

2.2.6 Geolocation API

The Geolocation API provides access to geographical location information associated with the hosting device. The browser creates a specific `Position` object on every `Geolocation` object (`navigator.geolocation`) request, with possible values for latitude, longitude, altitude, speed etc. as shown in the

following example. Together with the Google Geocoding API[8], the location can then be refined to country, city or street names.

```
<script>
  navigator.geolocation.getCurrentPosition(success, error);

  function success(position) {
    alert("Altitude: " + position.coords.altitude);
  }

  function error(error) {
    alert(error.message);
  }
</script>
```

2.2.7 Device Orientation Event Specification

This specification defines three DOM events providing high-level data of the physical orientation and movement of a hosting device: `deviceorientation`, `devicemotion` and `compassneedscalibration`. The device orientation describes the rotation in a "East, North, Up" earth coordinate frame, which is implemented as the `DeviceOrientationEvent` or the `OrientationEvent`. The device motion describes the rotation as well as the acceleration in the same earth coordinate frame, implemented as the `DeviceMotionEvent`.

2.3 Mobile Web Browser

As a mobile device forms a new and utterly different environment compared to a desktop machine, Web browsers had to be adapted. Consequently, *mobile Web browsers* were introduced to meet the demands of smaller screen sizes and resolutions, bandwidths and user interactions. Nowadays, there are numerous of preinstalled or downloadable mobile Web browsers for smartphones and tablets. The following examples – furtherly described in [11] and [7] – are currently the most widely used mobile Web browsers on the market[9]:

- Mobile Safari,
- Android Browser,
- Opera Mobile,
- Google Chrome
- Mobile Internet Explorer,
- BlackBerry browser,
- Symbian and
- Firefox Mobile.

[8] https://developers.google.com/maps/documentation/geocoding/
[9] http://www.netmarketshare.com/

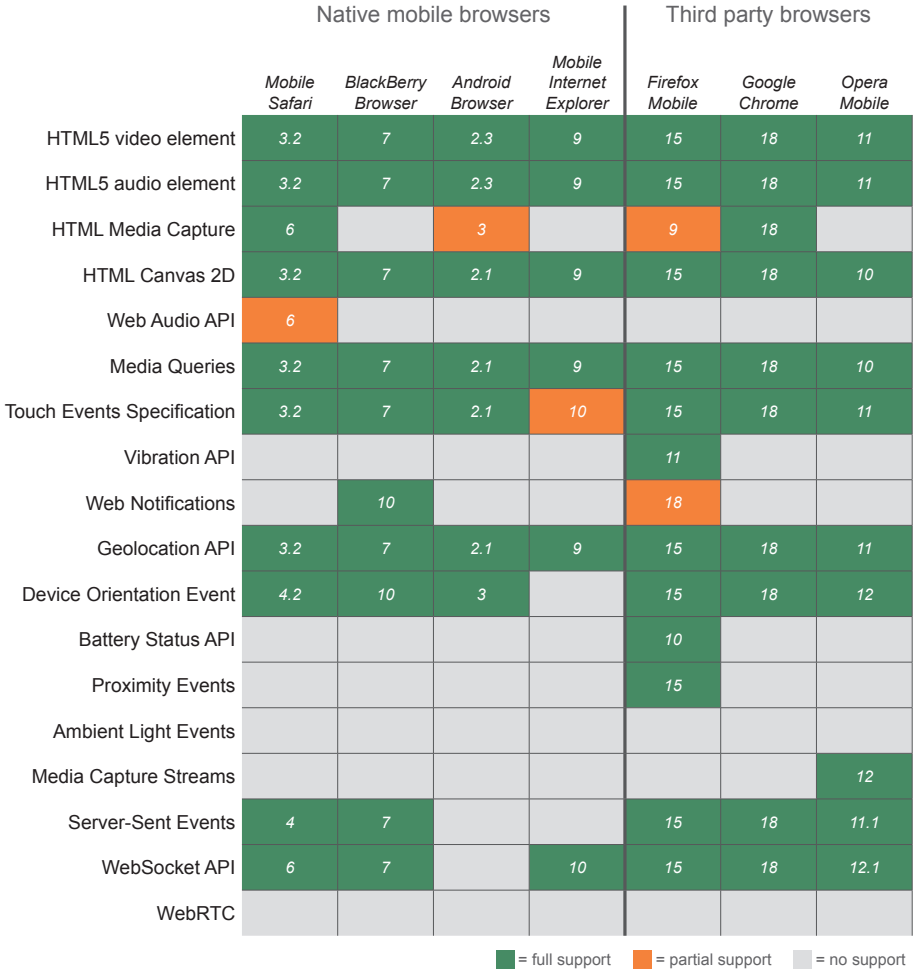

Figure 2.1: Browser support of native mobile browsers and third party mobile browsers for W3C's Mobile Web Application Standards (June 2013).

Figure 2.1 shows the the current state of browser support for the new mobile standards specified by the W3C group in [28]. The table focuses on the most widely spread mobile Web browsers – differentiated between native, preinstalled mobile browsers and third party browsers – and from which version on they support topic relevant APIs of W3C's standards for Web applications on mobile devices.

It can be seen that the Firefox Mobile browser is currently supporting the widest range of standards, shortly before Mobile Safari, Google Chrome, Opera Mobile and the BlackBerry Browser. Only the Android Browser and especially the Mobile Internet Explorer are lacking implementation of major and recently specified standards.

2.4 Communication Technologies

Real-time communication in the Web is one well-defined goal of the two major standardization bodies IETF[10] and W3C, as stated in [17]. Usually the communication between a server and client is built around the request/response model of HTTP, so the client has to explicitly ask the server for data changes each time. Real-time communication offers instant data exchange and notification on changes between server and clients. The following sections describe recent APIs that seek to accomplish this goal, as stated in [28], [16] and [18].

2.4.1 W3C APIs

As already described in 2.2, the World Wide Web Consortium is emphasizing on standardization of Web technologies. A particular focus is also on improving the communication between the servers and browsers by introducing features like WebSockets and Server-Sent Events.

WebSockets

The *WebSockets API*[11], also specified by W3C's Mobile Web Initiative, describes a bidirectional real-time connection between a server and a browser. This technology provides a persisting connection between two sockets over TCP (Transmission Control Protocol), which could only be accomplished before by hacks like long-polling[12] or browser plug-ins. The TCP connection greatly reduces bandwidth usage once the connection is established, because messages can then be sent back and forth without the overhead of HTTP headers.

Server-Sent Events

This W3C API is perfect for push-based functionalities in Web applications. The *Server-Sent Events*[13] offers a one-way real-time transportation of data with the `EventSource` object. Similar to WebSockets, this object is connecting the browser to the server and waiting for incoming messages to be processed within the client-side application. A major benefit of Server-Sent Events is its connection handling and message tracking. It remembers the last received message ID after the connection dropped, with which it can request the related backlog from the server.

[10] The Internet Engineering Task Force – http://www.ietf.org/
[11] http://www.w3.org/TR/websockets/
[12] Client constantly sends requests for possibly new available data to server.
[13] http://www.w3.org/TR/eventsource/

2.4.2 Node.js and Socket.IO

Node.js[14] is an event-driven, asynchronous server-side JavaScript Web server, powered by Google's V8 JavaScript engine. This technology makes the application's communication significantly faster compared to traditional server-side implementation. *Socket.IO*[15] is a transport library for Node servers. In addition to WebSockets, Socket.IO supports several other transport types among which it can choose at runtime, if a browser doesn't support WebSockets. This capability gives Socket.IO a brilliant browser support.

2.4.3 WebRTC

The *Web Real-Time Communication* (WebRTC) is an open source project that enables Web browsers to establish peer-to-peer connections via a simple JavaScript API. In contrast to WebSockets and Server-Sent Events, this technology sends data directly between two browsers over UDP (User Datagram Protocol) with the *Real-Time Transport Protocol*[16]. At the time of writing, there is only one experimental implementation of the WebRTC API in the Ericsson Bowser Browser[17]. Therefore this technology can't be used for mobile Web applications so far and will be left aside in this work. However, it will be highly interesting in future real-time connection developments.

2.5 Sensors and Interaction

Numerous new technologies come along with this rapid development for mobile devices. A sensor on a mobile device measures its environment and converts the data into a readable signal for the observer. Most of them are well-known and well-accessible, such as accelerometer, GPS or proximity. These sensors offer a broad range of new interaction possibilities for mobile applications as well as for Web applications. Collectively, these sensors enabled the development of applications for various new domains, such as healthcare, social networks, safety and transportation, which is being researched in the new area of mobile phone sensing. The research also focuses on processing raw data (coming from external devices) on mobile devices. The following list give an overview over currently available sensors, as described in [15], [14] and well documented in the Android SDK (Software development kit) Guidelines for Sensors[18].

[14] http://nodejs.org/
[15] http://socket.io/
[16] http://www.webrtc.org/
[17] https://labs.ericsson.com/apps/bowser
[18] http://developer.android.com/guide/topics/sensors

Accelerometer
: This sensor measures the acceleration in a three-dimensional scale, which is used to detect motions like shaking or tilting.

Digital compass
: The digital compass determines the direction in which the device is pointed to.

Gyroscope
: A gyroscope measures the orientation of a device directly and is used to determine the device's rotation.

GPS
: The Global Positioning System (GPS), allows the detection of a device's earth position by satellites. It specifies the position in longitude and latitude coordinates.

Microphone
: Microphones are used to sense sound in the device's environment.

Camera
: A camera takes pictures or videos of its environment.

Touch
: Modern displays can sense single and multi-touch actions of a user.

Proximity
: The proximity sensor detects objects near to it, for example an ear during a call.

Atmospheric pressure
: This sensor measures the environment's air pressure, which can be monitored on the device.

Ambient light
: Ambient light sensors are used to adjust the brightness of the screen according to the environment's light.

Humidity
: A humidity sensor determines the relative humidity value of the surrounding.

Ambient temperature
: This sensor measures the air temperature, which can then be monitored on the device.

2.6 Accessing device sensors

There are basically three ways to access sensor data on a mobile device. The primary approach is to use native operating system APIs for each platform (e.g. iOS, Android, Blackberry). Another approach is to access the context data via JavaScript APIs as they were already mentioned in section 2.2. Also, hybrid frameworks can be used, which are compiling the application's code

into numerous other preferred platform code. The following sections shortly introduce the three approaches according to [26].

2.6.1 Native APIs

A *native app* is directly built leveraging the native APIs provided by the operating system, such as iOS or Android. Every application is therefore developed in the operating system's (OS) specific language, like Objective C for iOS or Java for Android. The software runs directly on the device and gives the developer full access to the device's abilities. It can communicate with other devices or servers as well. Nevertheless, every implementation of an application has to be developed separately for each specific OS and users need to install the application manually on their devices.

2.6.2 JavaScript APIs

JavaScript APIs are used to create mobile Web applications. The provided features are implemented by the various kinds of mobile and Web browsers. Although not all browsers implement the same feature or same implementation of the features, these APIs increasingly provide seamless access to mobile device abilities.

2.6.3 Hybrid Frameworks

A *hybrid app* is basically a native app with HTML5 embedded in it. Hybrid apps aim to use the benefits of both approaches mentioned above, but to negate their disadvantages. Developing is a fast process in comparison to native apps or mobile Web applications, because hybrid frameworks reduce time and effort for implementation due to a simple and well structured development approach: implement once, build for several operating systems. As the implementation is mostly done with Web technologies like HTML5, such applications are also often referred to as *HTML5 apps*. This approach is easy-to-use, well documented and needs minor customization for uncomplex applications, but doesn't provide a real native app look-and-feel for complex applications. Examples of such frameworks are PhoneGap[19], Corona[20] or Titanium[21].

[19] http://phonegap.com/
[20] http://www.coronalabs.com/
[21] http://www.appcelerator.com/platform/titanium-platform/

Chapter 3

State of the Art

This chapter introduces some related work with unique approaches of cross device Web applications in the ubiquitous Web. All the described approaches are targeting the same research domain with different research aspects, such as adaption, integration and migration of Web applications for mobile devices. Following related work, some real world projects with similar or partially similar research aspects are presented and briefly analyzed.

3.1 Adaption

The adaption of Web applications (mostly of existing applications) to mobile devices has been an agile developing field since the first mobile devices with Web browsers entered the market. Over the past years, we have seen an increasing variety in mobile devices and therefore different screen sizes, resolutions and interface representations in mobile browsers. Particularly the user interface (UI) is in the focus of this adaption processes. As this work is concentrating on the integration of mobile devices in Web applications, this section focuses on the adaption of a Web application considering mobile dependent context-aware information and application structures instead of the user interface.

One novel generic approach to add context-aware features to an existing Web site is presented by Van Woensel, Casteleyn, and De Troyer in [20]. The approach leverages on-the-fly adding of such features, which means that the context-aware information is added asynchronously to a third party Web site while the user is browsing the Web site with a mobile device. This is achieved by extracting the Web site's semantic information, matching this information with the gathered user's context information and finally adapting the Web site according to the matches. An example for a context-aware feature in a collaborative Web application could be to show relevant information for a collaborating team when one of its members arrives at a geographical point of interest for the team's subject (e.g. a multiplayer game where users have

to visit locations in the real world). This shouldn't be added as a displaying feature but instead directly detectable and processable by the collaborative application logic.

The researchers of the Human Computer Interaction Group at HIIS Laboratory[1] (Human Interfaces in Information Systems Lab) present an entirely different adaption solution in [2], by using Automatic Reverse Engineering[2]. This approach focuses on the adaption of UI elements across platforms, by scanning the application's code and translating the core application into MARIA[3], an universal language for service-oriented applications in an ubiquitous environment. The resulting abstract concept includes the document's structure, its functionality and its visual representation, which can then be adapted to any kind of device and refined by specific, device-dependent languages. This solution enriches the adaptability of Web applications, but also requires great additional processing time for the reverse engineering and adaption process. Moreover, the system produces significant inconsistency in the resulting concept, as it can be seen in the paper's validation section.

3.2 Integration

This section introduces solutions for the integration of mobile devices from an architectural point of view, with attention to the user's and device's context information. Some of the following approaches are also capable of working with mobile sensor data and rich-media content.

One approach of providing context-aware information is presented in [13]. According to Kapitsaki, Kateros and Venieris, it is essential to separate the application logic from the context adaption in order to provide context-aware Web applications based on Web services. Thereupon, existing Web services are modified according to certain mobile context information, before the services are added to the Web application's presentation. This paper, published in 2008, considers an important architectural aspect for modern context information integration: the separation of the context adaption from the application logic.

An entirely different approach, concentrating on the integration of mobile devices in context-aware mobile Web applications, is presented by researchers of the Department of Computer Science in Oviedo in [6]. They focus on an architectural solution for handling a device's context information, by presenting a modular Web browser that is aware of specific context information XML tags. Any application could then define, in XML tags, which context information is needed for it to run properly. The modular browser is re-

[1] http://giove.isti.cnr.it/index.php

[2] Process to analyze and rebuild the application's structure, functionality and operations, based on [2].

[3] Model-based language for interactive Applications.

sponsible for detecting incompatibilities between a device and the specified requirements, asking the user for permission to use the context information and controlling the scheduled tasks. This is an interesting and expandable approach, which would state an enormous benefit for Web developers, but it still needs a lot of developing effort compared to the development using existing Web technologies.

I-Ching Hsu discusses a mechanism to facilitate the interoperability and reusability among heterogeneous context-aware systems and various mobile devices in [12]. He particularly focuses on the integration of Web 2.0 technologies[4] as a backbone in a Multi-layer Context Framework (MCF), in order to provide uniform access to context information. The MCF separates the context sensors/information/services and presentation, as well as the mobile devices and the context-aware applications. The described architecture is distinguished by its interoperability and the uniform access making it a reasonable approach for developing context-aware collaborative applications.

3.3 Migration

Nowadays, migration of Web applications across devices is experiencing increasing importance, as flexibility is a key necessity for modern multi-device users. Users want to switch from one device to another instantly with a seamless experience on all devices. The following approaches introduce some solutions to accomplish seamless migrations of the structure, state and sessions.

Ghiani, Paternò and Santoro from the HIIS Laboratory are presenting an approach in [10] to manipulate the Web pages in a way that it can be split up and shared across devices. Especially certain information at a time (e.g. form input data, search results, product information etc.) should be shared between multiple users across platforms. This approach is migrating existing interfaces to various kinds of devices and concentrates on DOM structure manipulation rather than device dependent context information.

The article of Bin Cheng [5] shows another good approach of cross-device collaboration and integration of mobile devices, by the use of a virtual browser. His research mainly focuses on how to seamlessly perform cross-device operations. Thereupon, he supposes to separate the logical workflow of an application from its presentation and to generate a single DOM with several sub DOM trees to be distributed to the corresponding devices.

The HIIS Laboratory group also performed research in the area of state persistency on Web application migration across multiple devices, as described in [3]. They particularly focused on how to preserve a Web application's state by its links, element ids and JavaScript objects. In this paper,

[4]Technologies which provide a medium for sharing and exchanging of resources, e.g. RSS, JavaScript, Ajax, SOAP, REST.

they present a novel solution by periodically scanning the current state, wrapping it in a JSON[5] object and responding to a client request with a migrated, state consistent Web page.

Furthermore, the researchers of the HIIS Laboratory group are also targeting an important migration issue: Security. In [9] they analyze the risks raised by such pervasive applications and present solutions to address them. Security risks can be the theft of private information or the intrusion of malicious versions of the migrated applications. Solutions presented by the research group are for instance to exchange data via the HTTPS protocol, run the application as a HTTPS connection, use a proxy to preserve correct data and to migrate the DOM form state between devices.

In addition to security issues, there are papers targeting the new given possibilities of accessing and migrating Web sessions with mobile devices. Alexandre Alpetite presents a basic way to migrate a Web session, with its session state and migration parameters, by dynamically creating a scannable 2D-barcode that embeds the information [1]. Another way of accessing restricted Web services by using biometrics is discussed by Carlos Vivaracho-Pascual and Juan Pascual-Gaspar in [21]. Biometrics is divided into physiological (e.g. fingerprint, face, iris, etc.) and behavioral (e.g. voice, handwriting, gait, etc.) categories and can already be partly used on mobile devices to identify a person and therefore grant access to Web sessions.

3.4 Related Projects

With the thriving development of mobile devices comes a whole range of real world projects aiming to support collaboration across multiple devices. Only a few of them are also focusing on the use of mobile device specific information and the new input and output possibilities. The following sections present some applications concentrating on the collaboration of multiple users within one application and/or with mobile devices. First, some basic collaborative applications are presented, then two real world projects are analyzed, which are targeting a similar research field as this work.

3.4.1 Collaborative Applications

Most of the currently available collaborative applications are natively built apps, desktop Web applications or a combination of both. The following projects are briefly analyzed and evaluated in contrast to the work's subject. They are targeting private, business and theoretical uses of such systems.

[5] JavaScript Object Notation is a compact and readable data interchange object.

Google Wave alias Wave in a box[6]

Google Wave is a suspended stand-alone application for instant content sharing and manipulation in collaborative *waves*. The project is currently under development as the open source project *Wave in a box* (WIAB) at the Apache Software Foundation. WIAB targets on an improved federation between multiple wave servers and its use as an open source project, rather than the integration of mobile devices into the application.

Scriblink[7]

This Web application offers a multi-user online whiteboard to exchange ideas and sketches with others. The owner also offers a business subscription to embed a customizable version of the service in other Web applications. Although it's a neat collaborative service, the application does not focus on any mobile adaption or integration so far.

Campfire[8]

Campfire is a web-based group chat tool for multiple users with secured chat rooms, where text messages and files can be exchanged. Furthermore, the application can be accessed and used via a native mobile app. The tool allows multi-device collaborative interaction, but only concentrates on desktop-oriented input/output possibilities.

aceproject[9]

This application is a project management, time tracking, human resources management and collaboration tool. The aceproject is especially designed for business use and offers a Web application as well as an adapted mobile Web application. Users only work together synchronously and the collaboration centers on user activities.

AgileZen[10]

AgileZen is a fully web-based project management tool for multiple teams to collaborate on multiple projects. This tool also integrates several collaborative applications like Campfire and github (collaborative code repository), but there is no explicit mobile integration considered for this tool so far.

[6] http://www.waveprotocol.org/
[7] http://www.scriblink.com/
[8] http://campfirenow.com/
[9] http://www.aceproject.com/
[10] http://www.agilezen.com/

clinked[11]

This Web application focuses on project management and data sharing for multiple users or teams. Clinked also offers mobile apps to always keep track of the collaborator's activities and easily share rich-media content. Particularly the asynchronous information and rich-media data sharing are well designed collaborative aspects, but the developers decided to implement native mobile apps for the purpose of use on mobile devices instead of a Web application.

Dazzle

Dazzle is a remote collaboration tool to improve face-to-face collaboration of a product designer team during meetings. The process should thereby be enriched by sharing rich-media content, such as video, audio and image files, across devices [19]. This paper presents a theoretical possibility to create a collaboration zone for better sharing, but without taking developing aspects into account.

3.4.2 Mobile Controllers

The integration of mobile devices in Web applications has always been a far vision from a Web application's point of view, until recent developments took place as introduced in section 2.2 and 2.4. There are two real world examples, as described in the following sections, concentrating on the aspects of these new possibilities to use mobile devices as controllers. These examples are simple games rather than collaborative Web applications. However, the applications are still relevant to this work, because multiple users are interacting with one application in one environment with various different devices at a time.

Unfortunately, the two games could not be tested at the time of writing, because the applications do not work on the tested platforms: LG Nexus 4 (Android JellyBean 4.2.2), Samsung Galaxy S (Android Eclair 2.1), Samsung Galaxy Tab 2 (Android 4.0.4), BlackBerry PlayBook (Tablet OS), iPhone 4 (iOS 5.1) and iPhone 5 (iOS 6.1.3). They were tested in all major mobile browsers with 3G network and WIFI connection.

Space Words

This game was created by GamesForLanguage[12], a company offering simple HTML5 games tailored for adults who want to learn foreign languages in a playful way. SpaceWords is a HTML5 game where up to 4 users can

[11] http://clinked.com/
[12] http://www.gamesforlanguage.com/

control spaceships with their smartphones, in order to collect graphical representations of foreign words in a space room. The game is available under http://spacewords.gamesforlanguage.com/.

It was developed to work as a Web application with at least two devices needed. One device, usually a desktop computer or laptop with a desktop Web browser, takes the role of the space room, in which the game takes place and the remotely controlled ships are shown. A Smartphone serves as a mobile controller for a space ship, where the interface shows touchable controlling buttons for left, right, front and back. Moreover, the application is also listening to the device's acceleration, which is the real purpose of use for controlling the ships. A representation of the desktop and mobile interface can be seen in figure figure 3.1. In order to access the Web session, this game provides a scannable QR-Code with an URL to the active session, so the camera of the smartphone can also be used for a better interaction with the Web application.

According to James Burke, developer of the game, accessing and processing the mobile device orientation with the new HTML5 standards is no longer difficult or time consuming. In combination with Node.js, Socket.IO and HTML5 mobile devices could be well integrated in Web applications [25].

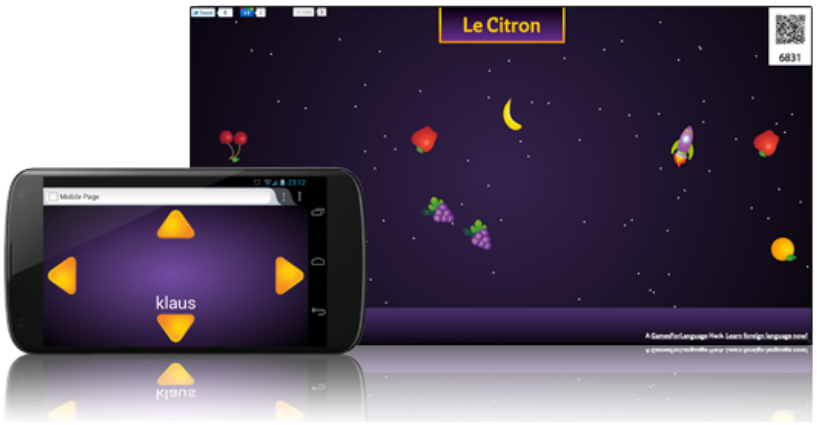

Figure 3.1: SpaceWords mobile and desktop view with remotely controlled space ships.

Shield Attack

Shield Attack was developed by UNIT9[13] as a multiplayer, cross-platform game, where the user can use the smartphone to throw shields into a game environment, by swiping over the smartphone's screen, on a Desktop or TV. The mobile application was developed using Adobe AIR with Flash, which

[13] http://www.unit9.com/

is a flexible solution, because the same source can be used for different platforms. The smartphone is seamlessly connected to the Web application, a 3D Unity[14] game, through a HTML5 WebSocket [27].

The interesting research aspect in this game is the use of a mobile and desktop or TV device working together as one application via HTML5 WebSockets. Besides this fact, the application basically uses native applications and no real Web application. This means that necessary plug-ins in the Web browser and the mobile apps for the smartphones need to be installed manually. Figure figure 3.2 shows the mobile's native application view and the Unity game environment view of a desktop or TV.

Figure 3.2: Shield Attack's mobile and desktop/tv view with throwing shields.

[14] A game development ecosystem.

Chapter 4

Mobile Integration Architecture

This chapter introduces the main conceptual approach of the research topic. The presentation of this approach is following an explicit listing of essential requirements before concluding with a detailed presentation of the architectural layers of the approach.

4.1 Integration Approach

As it has already been described in section 2.1.1, there are different kinds of Web applications. Either a Web application is developed from a web-centric, mobile-centric or a hybrid point of view. Therefore there are plenty of possibilities to access mobile device sensors to gather context information or rich-media content, as explained in section 2.6. Apparently, developing mobile native apps, mobile Web applications or hybrid apps are the common ways to interact with the Internet. Besides, most of the existing pages in the Web are not designed for mobile use or are "only" Web sites or Web applications adapted for use on mobile devices.

4.1.1 Application Type

Since mobile devices have seen enormous improvements in processor performance, memory capacity and network connectivity over the past years, Web applications are gaining more and more attention, as it can be seen in recently published figures about mobile internet access by the International Telecommunication Union (ITU). Another noticable reason for the increasing development of Web applications for mobile use is money. Developing mobile apps costs a lot of money, because the application has to be reimplemented for various platforms and mobile Web applications or hybrid apps also need a lot of adaption to run properly on mobile devices. On the contrary, Web

applications are following the "write once, deploy everywhere"[1] policy.

This work focuses particularly on Web applications developed with common Web technologies only, in order to facilitate a web-based integration of mobile devices. Such Web technologies, with focus on client-side integration in this work, are HTML, JavaScript and CSS. The recent versions of the markup language HTML (HTML5) and the style sheet language CSS (CSS3) have brought groundbreaking innovations for the adaption of Web applications to mobile devices. JavaScript is the most popular scripting language on the Web and revolutionized Web development by introducing asynchronous communication with the XMLHttpRequest object in 2006, the so called Ajax[2] technique. Moreover, JavaScript gains additional importance by the recent standardization process of the W3C for mobile Web applications, as already presented in section 2.2, and consequently represents the key aspect for the integration of mobile devices in Web application by Web technologies.

4.1.2 Architecture

Technologies for the mobile integration are present and mobile devices provide powerful platforms, but most of the presented architectures, for example in [7], [8] or [13], are focusing basically on the adaption to mobile platforms. On the contrary, some architectures presented in [6] or [5] do build on common Web technologies and mobile devices, but with the use of additional, custom services.

The architecture for this approach not only introduces a reasonable adaption for but also an integration of the mobile devices and their new capabilities. These capabilities, especially new interaction opportunities, should be facilitated in order to provide an enriched, seamless and quick collaboration between multiple users across devices. A central issue for the integration is the detection, access and processing of context information (e.g. device type, sensor capabilities, sensor data, user preferences).

As figure 4.1 on the next page shows, the approach basically consists of the four layers *Device Layer*, *Context-Aware Layer*, *Logic Layer* and *Communication Layer*, as well as a *Database* and a *Web Server*. These layers are separated from each other in order to provide an application running independently from its client devices. Above all, the Context-Aware Layer should act as an interface between the application and the interacting client devices. The Context-Aware Layer, the Logic Layer and the Communication Layer together illustrate the core Web application. Although the Web server might also be regarded as a main part of the Web application, it is more considered to be responsible for forwarding requests and the data exchange in this context rather than handling parts of the application logic.

[1] Slogan by Sun Microsystems for their Java language.
[2] Asynchronous JavaScript and XML.

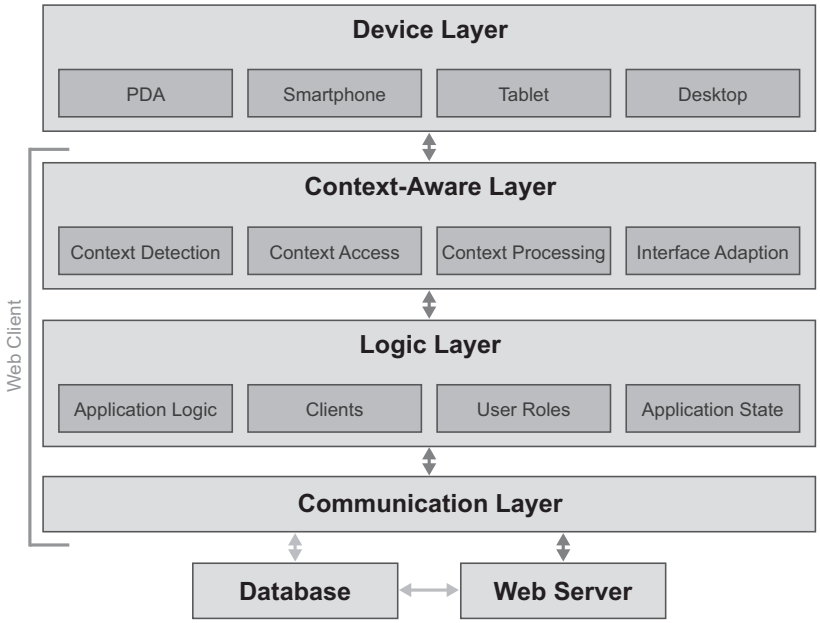

Figure 4.1: Mobile integration architecture for collaborative Web applications.

4.2 Requirements

The proposed system needs to conform to certain requirements in order to provide a satisfying functionality. These requirements are defined with attention to a seamless technical functionality for a collaborative Web application across multiple devices. Emphasized are cross-platform compatibility, multi-user capability, asynchronous communication and state persistence, whereas topics like security or device dependent interface policies are not focused on in particular.

4.2.1 Cross-Platform Compatibility

One of the central themes of the proposed system is the cross-platform compatibility. The application should be fully functional, accessible and interoperable across all modern types of smartphones, tablets, PDAs, desktops and laptops. This can be achieved by an appropriate detection of the device's context. Especially the handling of existing or non-existing device capabilities, required by the application, is crucial for its full functionality. It is of major importance for the compatibility, that the application is aware of the device's context and can deal with any possible platform. The context information should then be used to execute platform dependent code to access and process sensor data.

Another fundamental topic for a high cross-platform compatibility is the adaption of the user interface to the device's environment. The user interface is an essential aspect for an application to be compatible, because if the user can't cope with the application's graphical user interface on any device, the application cannot be considered fully compatible. Therefore the interface should be built and styled taking the given context information into account.

4.2.2 Multi-User Capability

As this document is dealing with collaborative Web applications, it can be assumed that the application is intended for multi-user usage. Thus, the system should be developed to support multiple active users at a time, with a state dependency upon all users' activities. Every user should be aware of other active users and their collaborative activities, because the application's state changes according to these activities. For this reason, users should always be associated with a certain state, coordinated by the application's logic and communicated to other client instances to preserve the application's interoperability.

It is not compulsory for a multi-user application to run with multiple users only. Certainly it would not make much sense in most cases for Web applications, because an application should be usable as soon as a page is requested, not just when another user joins the Web session. But still there are some cases this can make sense, for example in the game Space Words, presented in section 3.4.2, where one user opens up a space room (Web session) and waits for other users to join the session, before the game starts.

However, it is of major importance that the application is not restricted to the number of active users. It should be operable by a single user as well as multiple users, besides the fact that some services may require more users to work properly.

4.2.3 Asynchronous Communication

A collaborative application highly depends on its communication, like a project team depends on a comprehensive communication with the purpose to achieve its corporate goal. Collaborative applications can enrich and facilitate this communication between teams by sharing information, documents and rich-media content in one place, from everywhere, with everybody. Like in real life, a conversation in a collaboration group does not always work synchronously, because collaborators hear and see others while they are talking and not only when they finished talking.

Accordingly, the communication should be done asynchronously, to facilitate the conversations and prevent communication troubles. This means that data is sent intermittently between the server and its clients when data should be sent, instead of requesting and/or sending data in a certain time

interval. As soon as the user types words in a chat service or a shared document, all other users will see the changes "immediately", only with a small latency produced by data processing and transmission delay.

If the communication would be processed synchronously, the application would require a short time interval for steady requests or an open stream between client and server, which would cost more processing time and programming effort. Although asynchronous communication cannot ensure perfect real-time communication, synchronous communication additionally produces a higher risk of communication and document merging conflicts due to its lack of continuity.

4.2.4 State Persistence

Another essential requirement for a collaborative multi-user Web application is to preserve its state persistence. The application needs to be aware of all active clients and activities, as well as their individual states. Otherwise, the application's functionality can break due to invalid states or sudden state changes. Keeping track of its state as well as the clients' state is therefore essential for an application. The states can be saved and modified in a local storage (server and/or client device), offline storage or in a connected database. Each and every application is defined in different ways, therefore it is not always easy to filter out the needed information to preserve a state. The following list gives some examples of possible values to save for a client's or application's state:

- Unique Identifier (UID),
- Parent processes and/or child processes,
- Action state (e.g. ready, waiting, processing, stopped),
- JavaScript state,
- Form inputs and
- Socket ID.

State persistence is not only important to avoid errors while using the application, but also to provide seamless multi-platform use. If a user changes from one device to another, the state should be persistent on both devices in order to provide a satisfying user experience. Furthermore, if a user's state is invalid, other users might not be able to communicate properly with this user and therefore the collaboration within the team gets problematic or even fails.

4.3 Device Layer

This layer represents mobile devices like smartphones, tablets and PDAs as well as desktop-like devices such as laptops and standalone computers. In

the case of web-based technologies, the devices are only accessible from the application through its Web browsers. Consequently, only the Web browsers are responsible for the supply of context information and sensor data by their individual implementation.

Major desktop Web browsers, such as Mozilla Firefox, Google Chrome, Opera, Safari and Internet Explorer, are already well established in the Web and are not in particular focus of this work. Important mobile Web browsers were already introduced in section 2.3 and significant differences in the support of new W3C's mobile Web technologies are shown in figure 2.1.

4.4 Context-Aware Layer

The context-aware layer is the central layer for the mobile integration system, as it is the interface between the devices and the application logic. This layer transmits data between the logic layer as well as the devices and manipulates the representation on the devices, particularly the application's representation in the devices' Web browsers. The main reason for this separation is to provide a device independent application logic. Of course the application might offer services that are dependent of a device's functionality, such as orientation event data, but the context-aware layer is only responsible for context information and not for the logic.

First, the layer detects the device's capabilities, then it accesses the predefined device sensors and information and lastly it processes all the incoming and outgoing data between the devices and the application.

4.4.1 Context Detection

Context detection is the initial task of the context-aware layer, which is performed to create a full state of the available device's context information. Such context information states represent the accessibility of sensors and given environmental information. The first step for every mobile Web application should be the identification of the browser type, so if it is a mobile or desktop client. This identification is the first stage in the application's logic workflow. As soon as the identification was successful, the context detection detects all possible features implemented by the current browser and possibly even adds missing features.

Web Browser Type

Basically, there are two approaches for Web developers to identify the browser type. On the one hand, by parsing the `window.navigator.userAgent` object of the browser and on the other hand, by detecting supported features. The most common way for simple browser type detection, in order to differ between mobile and desktop browser, is to parse the user agent object. This can

be accomplished by several JavaScript plugins, like the open source project *Detect Mobile Browsers*[3], *Browser Detect*[4] and *Dojo's Browser (User Agent) Sniffing*[5], or by implementing a simple parsing process, as presented in the following code example.

```
<script>
  function isMobile () {
    userAgent = JSON.stringify(window.navigator.userAgent);
    if (userAgent.match(/Android/i)
     || userAgent.match(/webOS/i)
     || userAgent.match(/iPhone/i)
     || userAgent.match(/iPad/i)
     || userAgent.match(/BlackBerry/i)
     || userAgent.match(/Windows Phone/i)
     || userAgent.match(/IEMobile/i)
     || userAgent.match(/Opera Mini/i)
     || userAgent.match(/Opera Mobi/i)
    ) {
      return true;
    } else {
      return false;
    }
  }
</script>
```

Anyway, parsing the user agent is generally not the recommended way to detect the browser type anymore, because user agent information can be inaccurate due to *user agent spoofing*. User agent spoofing describes the process of Web browsers spoofing their identity in the user agent object, for example a Chrome browser pretends to be a Safari browser as well, in order to receive more server-sent information and therefore increase their compatibility. Furthermore, *user agent sniffing* – describing the process of parsing the browser type and eventually also the version – is also a threat to the application's functionality by possibly misinterpreting the client Web browser. However, only applications with specific browser-based implementations are confronted with these issues. In contrast, this approach presents a feature-based architecture, hence parsing the user agent for browser type detection is no threat to the application.

After the successful identification of the Web browser type, the context-aware layer needs to detect the supported technologies and context information on the device. Therefore it needs to check which APIs are provided by the browser, especially HTML5 and new mobile APIs. Again, there are different possibilities to detect and handle features by own standardized implementation or by the use of JavaScript libraries.

[3] http://detectmobilebrowsers.com/

[4] http://www.quirksmode.org/js/detect.html

[5] http://dojotoolkit.org/reference-guide/1.7/quickstart/browser-sniffing.html

Standardized Feature Detection

Basic feature detection can be done via the `window` or `window.navigator` object or DOM elements, as it can be seen in a few examples below. These examples show simple native JavaScript API object detection, HTML element tests and CSS module tests. Additionally, Opera 12.10 and Firefox Aurora already provide the native `@supports` rule for CSS detection, which tests requested CSS modules and provides return-blocks for further operations.

```
<script>
  function defineContext() {
    orientable  = window.DeviceOrientationEvent ||
                  window.OrientationEvent;
    streamable  = navigator.getUserMedia ||
                  navigator.webkitGetUserMedia ||
                  navigator.mozGetUserMedia ||
                  navigator.msGetUserMedia;
    vibrateable = "vibrate" in navigator;
    battery     = navigator.battery ||
                  navigator.webkitBattery ||
                  navigator.mozBattery;
    canvasable  = !!document.createElement('canvas').getContext;
    shadowable  = !!document.createElement('div').style['MozBoxShadow'];
  }
</script>
```

Library Feature Detection

Over the past years, several JavaScript libraries emerged for the front-end Web development community, providing feature detection functionality for cross-browser compatibility checks. Some famous examples are Has.js[6], which is a library focusing on tests for JavaScript features (e.g. Array, String, Object, JSON, console, ActiveX, XHR, etc.), or Detector[7], a PHP- and JavaScript-based browser- and feature-detection library. The most widely used feature detection library is *Modernizr*[8], which is focusing on HTML5 and CSS3 features, as well as other features like Geolocation, Touch Events and WebGL. The library improves development by its simple use and full list of detectable features. The following code example illustrates the significant difference in the usage compared to the above example with traditional native feature detection.

```
<script>
  function defineContext() {
    orientable  = Modernizr.deviceorientation;
    streamable  = Modernizr.getusermedia;
    vibrateable = Modernizr.vibrate;
```

[6] http://badassjs.com/post/1217357060/hasjs
[7] https://github.com/dmolsen/Detector
[8] http://modernizr.com/

```
 6     battery     = Modernizr.battery;
 7     canvasable  = Modernizr.canvas;
 8     shadowable  = Modernizr.boxshadow;
 9   }
10 </script>
```

Polyfills

polyfills is a modern technique to handle cross-browser feature compatibility. A polyfill is a code snippet that implements the required features if they are not available in the target browser. This technique adds functionality, especially HTML5 and CSS3, for older browsers or modern major browsers to ensure a cross-browser support experience in Web applications. Examples for libraries using polyfill or scripts providing the functionality are Socket.IO, HTML5Shim[9], Respond.js[10], MediaElement.js[11], Store.js[12] and also Has.js and Modernizr. Modernizr is one of the most popular cross-browser feature detection libraries, because it not only provides feature detection, but also polyfills for almost every modern API.

But this does not mean, that every missing API should be polyfilled, because this would result in a massive overhead. Polyfills should only be used for really necessary features. Modernizr provides this functionality to test and conditionally load polyfills by including the asynchronous resource loader yepnope.js[13].

In order to improve the adaption of the user interface, the layer can also detect the device's screen size and resolution. Additionally, the layer can detect device independent environmental context information, such as the user's location or the current weather for this specific location. The last, but most important task of the context detection is to save the gained information in states, as it can be seen in the above code examples. This represents the state of the device's available context information within the application and is of significant importance to the application logic. Feature detection libraries already provide state condition access.

4.4.2 Context Access

After the context was detected and the availability of the features saved in states, the context-aware layer provides access to the sensors and other features. This process starts by conditionally checking the availability state of each feature. If a feature (like camera sensor or vibration) is available,

[9] https://code.google.com/p/html5shim/
[10] https://github.com/scottjehl/Respond
[11] http://mediaelementjs.com/
[12] https://github.com/StevenBlack/store.js/tree/
[13] http://yepnopejs.com/

the layer should access the context and sense data as provided by the feature's API, which is also depending on the Web browser implementation. The sensed data is then provided to the context-aware layer for further processing.

Data Sensing

Data sensing is one of the main advantages of standardized Web technologies compared to native mobile technologies, because the standardized APIs provide single predefined objects and methods to access the desired context. On the contrary, accessing context with native technologies differs a lot, for instance the access to the camera between the Android- and iOS SDK. However there are still minor variations in browser implementations of the standardized APIs, especially between WebKit-based browsers (Chrome, Safari, Android etc.), Firefox Mobile and Mobile Internet Explorer. The W3C and the IETF are emphasizing a standardized implementation and use of features to reduce the differences between browser implementations.

Basically, data is sensed by event listeners perceiving state changes of features (e.g. accelerometer, gyroscope, touch, location, audio or battery) or by simple object methods provided by the API. Especially mobile features like sensors require a steady check for state changes. On the other hand, environmental context information sometimes may only be sensed once without an event listener for sufficient context information. An example is to provide the initially sensed location for further processing by the `navigator.geolocation.getCurrentPosition()` method. The data sensing process results in formatted values, specified by the implemented APIs.

4.4.3 Content Processing

As soon as data is available, the context-aware layer processes data for the application logic or for the view representation. The former process is called *Input Processing* and sends formatted data from the device to the application logic, whereas the latter process is called *Output Processing* and receives data from the application logic in order to add new content to the application view.

Input Processing

Once data is sensed, it can be processed by the sensing event listener or by manipulating the object with the valid sensed data. As the sensed data sometimes varies due to different API implementations, it is of high importance to process it and provide generalized as well as simplified readable data to the application logic. Hence, the logic layer does not have to deal with data validation and can mainly concentrate on the application's logic.

An example for different implementations of the same feature in Web browsers is the device orientation event. WebKit-based browsers and the

Gecko-based browsers do have the same W3C implementation of the API now, but they provided totally different formatted values for orientation changes until an update of the Gecko version in the end of 2011. Therefore it is important to process and map the data to a generalized format before it is used by the application.

Another example for using context processing to prepare sensed data is when additional services are required for proper information. For instance, the `navigator.geolocation.getCurrentPosition()` method returns geographical values for latitude, longitude, altitude and accuracy, but no information about the state, region or city. Especially for Web applications it might sometimes be more useful to work with human-readable information instead of numerical values. The Google Geocoding API[14] is one popular API providing this functionality and can be used to enrich the sensed location data, before sending it to the application logic.

Nevertheless, the use of external services and the implementation of long processing code is performance intensive and should only be used if really necessary. Additional services should not be used within event listeners, but only for initial processing or on seldom state changes.

The context-aware layer is also be able to process innovative types of interaction, such as scanning 2D barcodes or fingerprints. After the context was processed, the application logic is notified about the action, results, possible errors and sensed data, so it can decide what has to be done with the gained information rather than validating the input.

Output Processing

The layer is not only responsible for sensed device data, but also for processing incoming information from the application. Such incoming information could be actions to add any kind of new content, notify the user, update other users' activities or simply to change content. First of all, the context-aware layer needs to identify the executable action. Then it checks for support by the previously detected context states. If the Web browser supports the action, it can be executed on the client device.

The Vibration API, for instance, offers the ability to trigger a vibration event on a mobile device. If the application logic decides to send a vibration action to the device, the context-aware layer first identifies the action, checks its support and executes the action only if the API is supported by the Web browser implementation.

Another example for an action could be the adding of a new video file to the application's view. Therefore the context-aware layer only adds the new content, if the output processing process verifies support of the file type and action.

[14] https://developers.google.com/maps/documentation/geocoding/?hl=en

4.4.4 Interface Adaption

Interface adaption concentrates on the adaption of the graphical user interface and the application content to the specific device's environment. This approach assumes that the Web application is built from scratch and therefore the interface adaption does not have to cope with a complex desktop-oriented design, but can concentrate on a cross-platform design. A cross-platform design already considers a flexible layout for big and small screens, with only small basic differences predefined for mobile and desktop versions of the application.

Screen size/resolution

The adaption process mainly concentrates on the already detected screen size and screen resolution. This information about the target environment is essential for the arrangement of the content in the application's presentation layout. However, the application should not provide specific interface adaptions for several screen sizes and resolutions, at highest for mobile and desktop, because the cross-platform design should be flexible enough to display important content in any environment.

User Preferences

User preferences are gaining more and more popularity in Web applications, especially for small screen sizes, so the user can modify the representation according to his/her preferences. Web applications do not have to provide customizable settings for the user interface, except it improves the application's usability. The following list shows a few examples of feasable user preference settings for a customized user interface:

- Lock/unlock screen rotation,
- Font size,
- Button sizes,
- Enable/disable shadow effects and
- Background color.

4.5 Logic Layer

The logic layer is the brain of the system and controlling interface of the whole application. From content management to communication and view presentation, the logic layer specifies the resulting functionality. It is also responsible for handling the logical workflow within the application system, defined by the developer to follow a certain strategy of conditions. Figure 4.2 shows an abstract strategy for processing requests to actions.

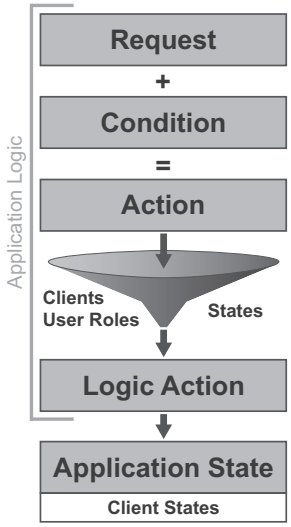

Figure 4.2: Representation of a logic strategy to process requests.

The strategy is defined by workflow conditions and is basically depending on the application's state, which is defined by actions the users take. Collaborative Web applications require the system to gain additional knowledge beside the application state. Especially knowledge about all clients connected to the application instance is important for collaborative Web applications to provide continuity in a multi-user environment. Furthermore, the logical workflow should be aware of specific user roles each user has, before processing the actions to other layers. After the action was processed and is sent to other layers, the strategy updates the application's and clients' states.

4.5.1 Application Logic

The application logic is controlled by the logic workflow. By the specification of the workflow, the developer indicates a certain strategy the application has to obey. This process interprets user actions as well as raised errors and decides which actions to take according to the given conditions in the workflow. As the context-aware layer is responsible for the interaction between application logic and client device, this process does not have to cope with any adaptions and can purely concentrate on the logical workflow.

On initialization of the system in a Web browser, the logic coordinates actions to be taken before the system is ready, such as the context detection, -access, -processing and interface adaption considering the user preferences. The strategy then defines the next action to take based on the corresponding condition, given clients, applied user roles and states. The resulting logical actions are finally communicated to the context-aware layer for output processing or the communication layer for input processing.

4.5.2 Clients

In collaborative Web applications, every client is regarded as a single collaborator. Every user is a client to the application system, no matter which platform is used. As every client has its own state, the user is able to switch between devices without significant usability changes. One difference may occur due to the dependency on the device's context, because not every device provides the same sensors and other context services. However, a client triggers state-changing actions which affect the application and other clients.

It is managed by the logic layer, as a client itself does not necessarily need to be aware of all connected clients in the application system. It heavily depends on the actions the clients can take, which information is considered to be necessary or not. For instance, if a collaborator wants to share sketches with all other collaborators, it is not necessary for the client to be aware of other connected clients, because the collaborator wants to share them with all clients anyway. But if a collaborator wants to start a video chat conference with another collaborator, it is of importance for the client to know which collaborators are currently connected and ready to accept incoming video chat calls. Thereupon, the application logic can decide, which state information should be sent along with action requests. Clients should only receive and work with information they really need, so the usability for users can be fully granted.

4.5.3 User Roles

User roles define permissions and/or roles for users within a collaborative system. The logic layer is responsible for filtering the actions for each concerned client based on its user role. Only if the user is allowed to receive or take the action, the logic layer will process it. The key aspect of user roles is to make the collaboration more administrative, which is of particular importance for collaborations within large teams or if the application's functionality requires different types of users.

The former is reflected by real life collaborations within hierarchical systems, such as a company or project team, where users have different permissions on how to act within the system and which actions they can take (e.g. manager, full-time employee or intern). This role separation implies permissions to edit, modify and create content, as well as to access restricted content or to share information.

The latter indicates another use of the collaborative system depending on the application's target functionality. An example was presented with Space Words in section 3.4.2, where one client is responsible for presenting the game room and other clients interact with the game room. Although this game has only one type of user role that actually interacts with the application system (the controlling smartphones), the space room does take

the user role of visualizing the playground. An extension to this functionality could introduce a second user role, where the user has to type the wanted words, whereas the other users have to catch the wanted fruit representations.

4.5.4 Application State

An application state indicates the state of the connected clients, shared content and application logic. It can change after each request and is essential for the application to work as expected. Every action has to be filtered and mapped with the application state before the logic layer can process the action. This ensures valid processing of requests and prevents invalid state changes, which could result in breaking the application's system.

4.6 Communication Layer

As the interaction with mobile devices is done purely on the client-side, the communication layer basically handles the communication between the client application and the server application. The former is representing the processes running on the client-side and therefore on the devices, whereas the latter is representing the Web server and the database.

This layer acts as an interface between the application and its data source and is responsible for establishing a connection to the server and possibly also to the database. It depends on the preferred strategy on how to store data (see section 4.8) to decide in which way the database should be accessed, being either directly by the communication layer or via a Web server. The following sections introduce significant characteristics of the communication layer for collaborative Web applications.

4.6.1 Communication Type

There are basically two ways to communicate: synchronously and asynchronously [24]. The communication types are the similar between the Web and the real world, depending on the expectation of a certain time to reply, urgency and importance of the message. Most communication in real life is done synchronously by talking and listening to other persons, whereas sending letters represents an asynchronous way of communication. The information is processed instantly when talking to a person, which cannot be expected for a letter.

Reflected to collaborative Web applications, synchronous communication needs to be almost real-time to achieve instant information sharing between multiple users. It should provide a seamless collaboration experience for instant messaging, phone calls, video chatting or simultaneous file editing. One major requirement for synchronous communication is the presence of all communicating parties, because the connection is exchanging data periodically

and needs a persistent connection between senders and receivers. This type is quite traffic consuming due to the number of needed requests to provide a persistent connection. Even if requests of a synchronous connection need only small traffic space because of the awareness of the other users.

On the other hand, asynchronous communication in CWAs can be used for tasks where no instant reply is expected or required, for example private messages, shared document editing or image sharing. It is also well suited for broadcasting, where one user sends a message to all listening receivers without expecting an immediate reply. Such tasks are popular, because the sending user does not have to wait for any response and can continue the interaction with the application. This type of communication leverages higher flexibility on the usage of the application, but also increases traffic space because it mostly sends and receives a full request on each transmission.

A collaboration application should provide a reasonable balance between synchronous and asynchronous communication, as the different tasks may demand different types. Overall, asynchronous communication can be favored due to providing more flexibility for users in a collaborative environment. They can work together with the community but also resolve own tasks in the same time.

4.6.2 Data Exchange

From a technological point of view synchronous communication is a typical request/response strategy, where the application logic is waiting for the transmission to finish before other tasks are handled. In order to provide synchronous communication in Web applications, the system needs to establish a persistent connection between users with methods including long-polling or streaming[15]. In order to keep communication traffic low, it is important to use synchronous communication only when reasonable.

Ever since AJAX was introduced, asynchronous communication gained enormous popularity in the Web, enabling small tasks on Web pages to be processed without a page reload. A user sends a request via the XMLHttpRequest[16] API to the receiver and doesn't wait for a response, but provides a specific callback function which will be called with the response. This progress has led to the development of fully asynchronous tools, Web services and even Web servers, for instance a Node.js server, as presented in section 2.4.2. Asynchronous transmission can be established using sockets and a TCP connection.

The communication layer should be able to transmit data upon its type of action synchronously or asynchronously. It needs to be capable of transmitting and receiving data simultaneously to facilitate a fast collaboration.

[15]Streaming is mostly achieved by sending packets (e.g. RTP, RTCP) over the User Datagram Protocol (UDP).

[16]API to send HTTP requests directly to a Web server with any possible request method.

Data should thereby exchanged in a lightweight format, such as XML or JSON. Furthermore, all requests need accurate error handling (passed to the application logic) to prevent incomplete and erroneous communication.

4.7 Web Server

The Web server is the most important part of a Web application because it provides access to the necessary source files, such as HTML, CSS, JavaScript or images, via a unique IP address and processes incoming requests from the World Wide Web. All application dependent processes on the Web server are controlled by a server-side script, which dynamically creates HTML pages and defines the response data. Web developers are faced with a broad variety of Web server implementations, but the most important ones nowadays are the *Apache HTTP Server* and the *Microsoft Internet Information Service*.

A modern alternative to common Web servers is *Node.js*, which was already briefly presented in section 2.4.2. This is a server-side Web server built atop Chrome's V8 JavaScript engine. The fact that this Web server is using a non-blocking I/O model and can be controlled with JavaScript makes Node incredibly fast and perfectly accessible for the client-side application.

It depends on the application system and the given hardware resources to decide for a type of Web server. For collaborative applications, the Node Web server might be the best solution due to its better performance and easier accessibility compared to other servers. This is the case especially because the integration of mobile devices is mostly done by client-side JavaScript. But when it comes to highly complex systems with the use of a lot of different libraries and a complex database, Node could possibly perform worse than other servers. Therefore it highly depends on the application system and its functionality, but Node is basically a reasonable choice for asynchronous collaborative applications.

4.8 Database

A database represents the collection of application dependent data. It is part of the back-end application, handling non-user specific tasks such as data manipulation and data storage. Nowadays, there are numerous of different ways to store data and it is up to the Web developer which type to choose. When choosing the type and technology to store data it basically depends on the Web technologies used to develop the application system. Moreover, it depends on the targeted functionality and the application requirements. The following sections present the two main possibilities for storing data: on a server or locally on the clients.

4.8.1 Server Storage

Each server database type uses its own query language, structure and logic and can run on the Web application server or on an external server. Efficiency and scalability are one major benefit of using server storage, because the hardware can be utilized as the developer wants and is not dependent on any other machines. There are three main types of server databases: relational, key-value and document-oriented.

Relational Databases are the most used and popular type of databases in the Web. These databases can handle almost every complexity of relational data management structures and usually provide a good administration tool for developers. The Structured Query Language (SQL) thereby defines the core of all relational databases. Some of the most important server systems nowadays are Microsoft SQL Server, Oracle and MySQL.

A key-value database stores key-value entries in a non-relational collection table. This form of database is a so-called NoSQL database, because it is not using SQL as a core feature. One popularity gaining key-value storage database is *Redis*[17], which is using the main memory of the server machine, in other words the random-access memory (RAM), to store data. Manipulating data in the main memory is really fast compared to the disk memory, which is still common for server databases. It only writes stored key-value pairs to the disk memory in a predefined period of time or after a certain amount of database manipulations.

Document-oriented databases are also NoSQL systems, basically storing data in documents similar to relational databases, but less rigid. The databases can store files in different encodings (e.g. JSON, XML, YAML) and the entries can have different schema. Popular examples for document-oriented databases are *MongoDB*[18] and *CouchDB*[19].

4.8.2 Client Storage

Client storage, also called local storage, is storing data on the client-side rather than on the server-side, hence directly in the requesting local client browser. Server data always needs to be sent form the server to the clients before it can be represented, whereas locally stored data on the client can be rendered as soon as the user is requesting a page and recognizes already stored data.

Native applications are quite powerful in this area for mobile devices, because a native operating system usually provides a specific layer for application data to be stored on the devices. Web applications have not supported local storage for a long time. Over the past decade, browser plugins

[17] http://redis.io/
[18] http://www.mongodb.org/
[19] http://couchdb.apache.org/

and adapted cookie versions were developed to improve the old fashioned way of local storage with cookies. A Cookie is stored in a text file with a maximum size of 4 KB, which is then sent along with every server request and therefore slows down the application performance. Since a few years, the W3C has been working on the *Web Storage*[20] specification, often referred to as DOM storage, which is the first natively implemented local storage API in Web browsers, having almost full cross-browser support. It revolutionized modern data handling by enabling persistent local offline data storage of key-value pairs in data tables on the Web browser. In contrast to cookies, Web storage data is not automatically included with every server request and the system offers much more storage space. The storage can be accessed and manipulated with JavaScript via the `window.localStorage` object, where data is stored without an expiration date in a specified file, or the `window.sessionStorage` object, which is storing data only for the duration of one session. Furthermore, Web Storage fires events on data changes, which is very useful for applications running in multiple windows. There are already numerous of different libraries available to provide cross-browser support by the use of different browser implementations. Popular examples are dojox.storage, jStorage, AmplifyJS or Store.js.

IndexedDB[21] and *Web SQL*[22] are further alternatives offering even more memory space and can store data using indexes, which is especially useful for large databases. However, W3C is purely concentrating on the specifications for Web Storage and IndexedDB instead of Web SQL, because Web SQL has a lack of independent implementations needed to continue the recommendation process.

Generally speaking, a collaborative Web application seeks for a database system with plenty of available space, high efficiency and scalability, because of its need to handle a large amount of shared data and multiple user accesses. From a collaborative point-of-view, client storage is dangerous, because every client has its own local database. Due to asynchronous communication and a large data exchange rate, this could result in a non-persistent data collection for shared data and therefore inconsistency in the application.

A database for collaborative applications should guarantee multiple users to manipulate data simultaneously, which is why server databases could be preferred over client storage systems for storing shared data. Although client storage can enormously improve an application's performance and flexibility and can be highly useful for client dependent tasks. Such tasks could be to store user preferences, login credentials, backup saves, static content or to provide offline use.

[20] http://www.w3.org/TR/webstorage/
[21] https://developer.mozilla.org/en-US/docs/IndexedDB
[22] http://www.w3.org/TR/webdatabase/

Within the presented integration system, the database can be accessed by the communication layer as well as by the Web server. It depends on the application system whether it uses a server storage or/and client storage system.

Chapter 5

Prototype

As this document introduces a theoretical approach for the mobile device integration in Web applications, it is crucial for the proposed system to test its practicability in a real-world example. Therefore, this chapter presents a prototypical application that was developed as a proof of concept.

The prototypical application introduces the *Rich Media Collaboration Tool* (RMCT)[1] that concentrates on the technological state of the art of collaborative cross-platform and multi-user Web applications, especially built with traditional web technologies. It should reflect the given technological possibilities for developers to access and process mobile context data via Web APIs. The main goal of the application's functionality is to facilitate online rich-media collaboration within communities, teams or project groups.

At first the the used application model will be presented, followed by the technological architecture and implementation and finally an illustration of the user interface.

5.1 Single Page Model

The *Single Page Model* is a programming approach for the Mobile Web. As the name already implies, this means that the entire content of a web application is combined in one single page, as stated by Hales in [11]. Thereupon, every JavaScript- and CSS-file is included in a single HTML page, which also contains the complete required DOM tree. The body of the DOM is thereby usually structured in segments of pages, tools or separate functionalities. Whenever a user navigates to another page, for instance to a certain interaction page in the RMCT, the other page's markup is being hidden and the requested page markup is being shown by the application logic. These segments should basically be enclosed by a `<div>`, as you can see in the following abstract structure of the prototype's markup:

[1] The repository can be found on https://bitbucket.org/Aaang/richmediacollaboration.

```
<body>
  <div id="authorized_wrapper">
    ....
    <div id="menubox">
      ...menu to choose which content to display
    </div>
    <div id="contentbox">
      ...interaction pages (e.g chat, image gallery, etc.)
    </div>
  </div>
</body>
```

A single page application has primarily one major benefit for interactive Web applications: it reduces additional HTTP requests. HTTP requests are often causing performance problems due to additional HTTP requests for every user navigation. The single page model has only initial requests on the first page load. Furthermore, this model provides a better mobile-like look and feel due to its faster page loads and possible smooth transitions between the pages.

5.2 Architecture

The prototype was built as a single page application based on a Node.js Web server and the Socket.IO communication API. It consists of one Web server and multiple client sockets that represent the application's core functionality. The architecture also depends on its source files, which are usually found on the hosting server. Although databases are of significant importance for most Web applications, the RMCT prototype has not integrated any database, because data storing is no key aspect of the research topic. Even though data storing should be essential to provide a proper functionality for long-time collaborations within the RMCT. Figure 5.1 on the next page illustrates the given technical architecture of the pototypical application, which will be described in more detail in the following subsections, following a workflow presentation.

5.2.1 Web Server

As described in section 2.4.2, Node.js is an event-driven, asynchronous server-side JavaScript server powered by the V8 engine in order to provide faster communication in the Web. Especially the asynchronous event-driven communication makes request handling extremely fast, because the server handles every request and I/O by an infinite and non-blocking event-loop, as it can be seen in figure 5.1. This enables clients to execute code after firing an event to the server and therefore while they are waiting for the server response, which is called an event callback.

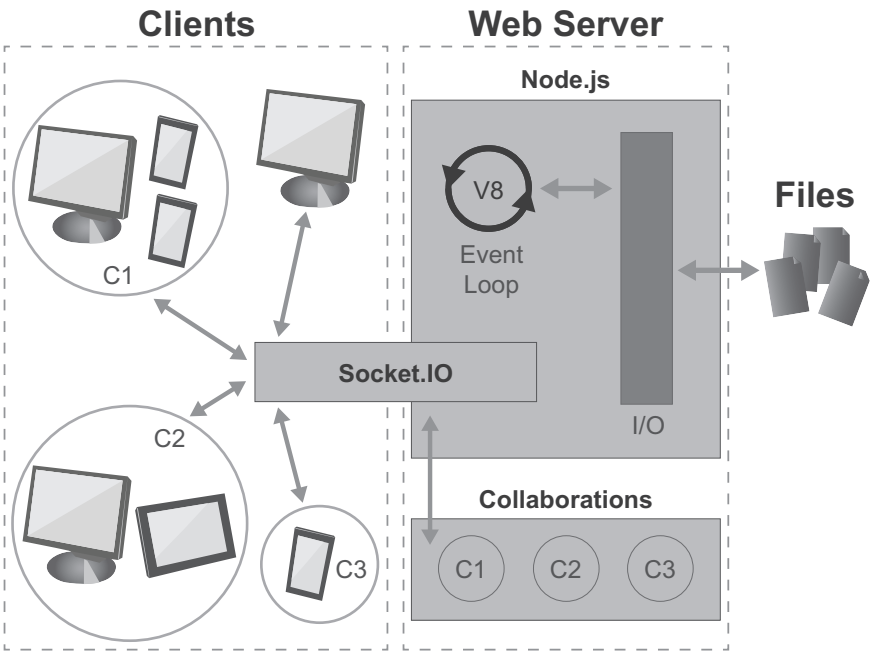

Figure 5.1: Technical architecture of the pototypical application.

The Web server in the RCMT creates an HTTP-server which sends the HTML pages to the requesting page. As soon as the document has been loaded, the client script fires and creates a socket connection between the Web server and the client. This connection is established with the Socket.IO framework and then the server listens to events from all connected client sockets. These listeners are representing the communication layer between client and server, but inside the listeners, the application logic needs to decide which actions to take on specific incoming events.

Server-Side Logic

In figure 5.1 it can be seen that the clients communicate with the Web server, but it would be much more efficient for client-centric applications, such as the RMCT, to communicate directly to other clients via their sockets. This is called a peer-to-peer connection – as presented in section 2.4.3 – which is not supported yet in most Web browsers and therefore not integrated in this prototype. Consequently, the system requires logic to be placed on the server in order to assure persistent data synchronization.

The server-side logic decides upon certain conditions within the socket listeners and information about the collaboration's state. It explicitly reacts upon the collaboration's state, because this is the platform for every inter-

action between the clients and the application. A soon as the user submits a valid name for the collaboration, the logic creates and saves a collaboration instance and redirects the user to another page, where a new socket connection is being established. A collaboration is aware of all its connected client sockets and collaboration data, such as text messages, images, videos and so on. It can add and remove content and represents the key aspect for the server-side logic. Additionally, it also broadcasts events to all of its clients and handles its own state until deallocation.

5.2.2 Clients

As already mentioned in section 2.1.1, clients can be considered as Web browsers – the environment where the user interacts with the application. A Client in the RMCT is defined by its state, logic and socket connection to the server. Sockets represent identified endpoints in Internet communication processes and can be used to establish a faster, bidirectional connection between a client and its server. Particularly WebSockets, as presented in section 2.4.1, provide a good API for this purpose. In order to support access to this technology for the prototypical application, the Socket.IO transport library for Node servers was integrated in the system. As briefly described in section 2.4.2, Socket.IO decides at runtime which of its six possible transport types to use for transmitting the data. However, the best result in socket communication is achieved by using Web browsers and a hosting server that provides an implementation of the WebSocket API.

Basically, every client connecting to the application creates a socket which is then a client socket, but not every client socket is being added to the application system. For instance, the desktop client without a circle and a collaboration instance in figure 5.1 is probably on the landing page and in the process of creating a new collaboration. A client socket is only added to the system if the user is connected to a valid collaboration and has submitted a valid name. As soon as the user is connected to the collaboration, the user's socket is listening to events which are then processed by the client-side logic.

Client-Side Logic

The client-side logic is responsible for controlling the interaction between server and client. It is aware of the socket connection to the server, the presented view and the client's state, including user information and context information. Although the client is the key aspect for the client-side logic, it highly depends on the collaboration's state, which is controlled by the server-side logic. The arrows from the clients to the Socket.IO layer and forth to the collaborations in figure 5.1 illustrate this logical dependency of the clients to their collaboration instances. The client socket, the client-side logic and the view presentation collectively represent the client Web application.

5.2.3 Logic Workflow

Some aspects of the basic workflow for the collaborative application have already been introduced in the previous sections. The sequence diagram in figure 5.2 on the following page illustrates the workflow from client connection to collaboration creation and finally message sending. For an easier understanding of this workflow the actions are readably named rather than following the exact naming in the implementation.

The first step in the workflow is always the HTTP page request to the server, which will return a HTTP response with an HTML file in the body. After the client has requested and loaded all necessary scripts and styling files, the client-side logic creates a socket and sends an initial HTTP request to the target server. If the server accepts the connection, it creates a so-called handshaken-connection and responds with the session-id that signalizes the successful bidirectional connection between the two sockets.

The next step of the logic is to verify the target collaboration, assuming that the user is requesting an URL with a valid collaboration id. Therefore the server filters the list of collaborations and returns a valid response when it exists in the list. If not, the server will respond with an error message and the client logic will ask the user to change the id or create a new collaboration.

As soon as the client is connected to a valid collaboration, the logic prompts the user to input a name visible to other users. This request will then again be filtered on the server within the list of clients of a valid collaboration. If the name is available – no other user has the same name – the client shows the full page content in the browser, hence the user is fully connected to the system. Afterwards the user can send a message – as demonstrated in figure 5.2 – which will be stored on the server and broadcasted to all connected client sockets.

5.3 Implementation

This section emphasizes on a concise presentation of the prototypical application implementation. The architecture and logical functionality have already been presented in the previous sections and will therefore not be discussed in detail again in this section. Moreover, it provides accurate code examples for essential functionalities of the application system.

The presented code examples are not executable and minor important lines of code might be missing, which are illustrated as dots (. . .) within the code blocks. Furthermore, the code could somewhat differ to the attached project source code in order to improve the readability and understanding.

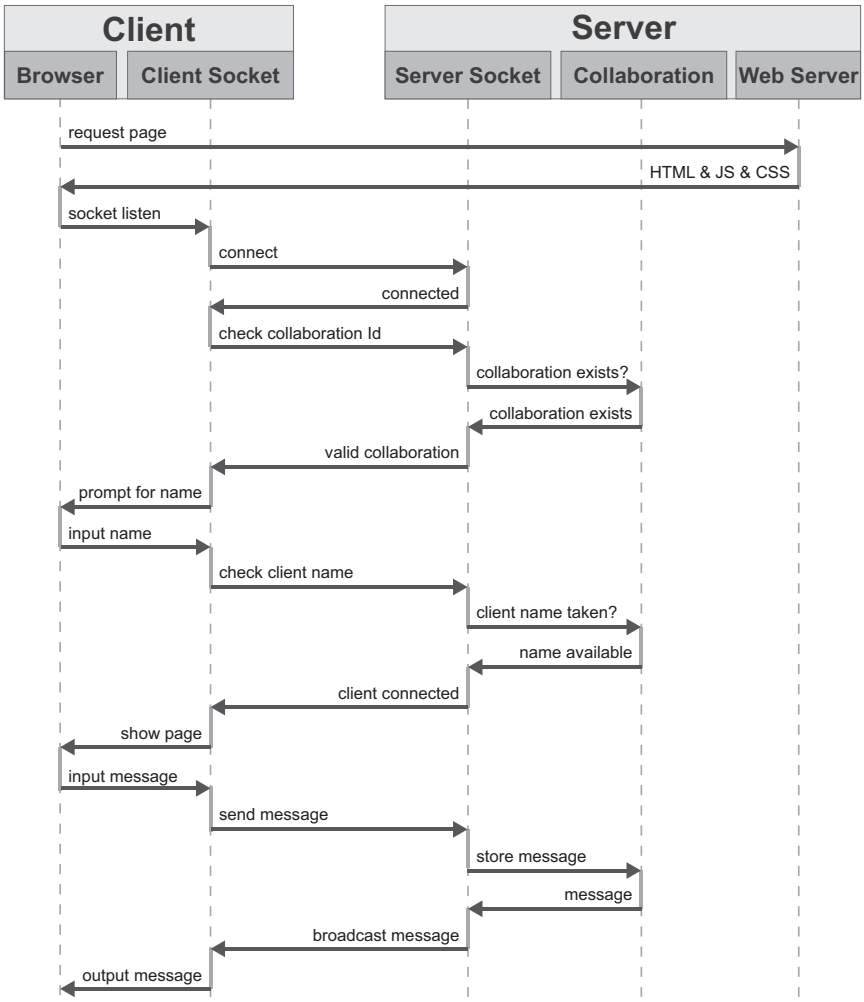

Figure 5.2: Sequence diagram of the basic application workflow.

5.3.1 Node.js Web Server

As already described in section 5.2.1 the Web server was built with the Node framework. The configuration of the node application, which is the basis for the server, can take some effort for proper error handling, template engine integration and data storing. Nevertheless, using this framework facilitates the creation of the server for Web developers, because only a few lines of code are necessary to create and handle a simple server-side JavaScript Web server, as demonstrated in the following code block.

```
1 var application = require("./app").getApplicationInstance(),
2     http        = require('http'),
3     server      = http.createServer(application.app),
4     io          = require('socket.io').listen(server);
```

```
5
6 server.listen(PORT, HOST, function() {
7   console.log("Server is listening on port"+server.address().port);
8 });
9
10 process.on('SIGINT', function () {
11   console.log("Shutting down server...");
12   server.close();
13   process.exit(0);
14 });
```

5.3.2 Connection

Generally, a client connects to the server by the Socket.IO API, but this does not automatically connect the client to the collaboration. In order to identify and validate the desired collaboration, the client-side logic parses the location URL of the `window.location` object, which is looking like this: `http://collaborationtool-anger.rhcloud.com:8000/collaboration/h3Se93Wi`. The URL path contains two important parameters: the application type (`collaboration`) and the collaboration ID (`h3Se93Wi`). This collaboration ID can then be used by the application logic to identify the collaboration and connect the client to it.

RMCT's connection process basically consists of three steps: socket connection, collaboration verification and client validation. At first the client script creates a socket and tries to connect to the Web server with the given location host and port, which can either response with connected, disconnected or a connection failure. If the client is successfully connected to the node server, the client-side logic sends a request with the parsed collaboration ID, as described above. Lastly, after the collaboration verification succeeded, the user is prompted to input a valid name visible to other users, which will then be sent to the server and validated for presence. If the username does not yet exist for the desired collaboration, the client is fully connected to the application system. The following lines of code show the socket event listeners for RMCT's client connection process.

```
1 client.socket.on('connect', function() { ... });
2 client.socket.on('disconnect', function() { ... });
3 client.socket.on('connect_failed', function() { ... });
4
5 client.socket.on('valid_collaboration', function(data) { ... });
6 client.socket.on('invalid_collaboration', function() { ... });
7
8 client.socket.on('client_already_exists', function() { ... });
9 client.socket.on('client_connected', function(data) { ... });
```

One possible improvement for the client connection could be to remember the user's name on a page reload, so the application could reuse a user after a page reload. Currently the client is being disconnected and deleted

from the collaboration after a page reload, which means that the user has to create a new user. In order to recognize a user after a reload in a single page application, the user's name could be saved and read with the URL's hash object. This information is accessible via the `window.location.hash` object, which does not trigger a page reload on manipulation [18].

5.3.3 Context Information

Context information is a collection of information about a device's environment, including available sensors, device type and size, accessible Web standards and the user. Most of the information is gathered by accessing Web standard APIs implemented by the presenting browser. A detailed list of relevant APIs was introduced in section 2.2 and their support in the currently most widely spread mobile Web browser is illustrated in figure 2.1 on page 9.

Usually these APIs are accessible via JavaScript, which is predestinated for applications fully developed in this scripting language, such as the RMCT. First, the context needs to be detected, for instance the device's Web browser type and its supported features, as already presented in detail and with code examples in section 4.4.1. Afterwards, the detected context needs to be set up and accessed, as shown in this code block:

```
1 function setupContext () {
2   if (!client.fileable) {
3     $(".file_input").addClass("not_supported").hide();
4   }
5
6   if (!client.clipable) {
7     $("#videos_button").parent().addClass("not_supported").hide();
8   }
9
10  if (!client.soundable) {
11    $("#audio_button").parent().addClass("not_supported").hide();
12  }
13
14  if (client.drawable) {
15    initDrawing();
16  } else {
17    $("#drawing_button").parent().addClass("not_supported").hide();
18  }
19
20  if (client.streamable) {
21    initVideoStream();
22  } else {
23    $("#videostream_button").parent().addClass("not_supported").hide();
24  }
25
26  if (client.orientable) {
27    initDeviceOrientationListener();
28  } else {
```

```
29     $("#orientation_button").parent().addClass("not_supported").hide();
30   }
31 }
```

This code block only shows the custom initialization function called to set up the context, but the actual access functionalities would be too much code to visualize and are mostly well documented in their related APIs.

As soon as the context is set up and the interface is adapted accordingly, the application logic initializes the context event listeners by calling the function `initInputListeners()` in the client-side script. This call starts the defined listeners – which are mostly accessed via JavaScript's jQuery framework – and therefore also initializes the context processing. The following listener shows a simplified example code for the access and processing of an uploaded sound file:

```
1 jQuery(document).on("change", "#sound_input", function(e) {
2   if(this.files.length > 0) {
3     var file = this.files[0];
4     var name = file.name.substr(0, file.name.lastIndexOf("."));
5     var mimeType = file.type;
6
7     if(isValidAudio(mimeType)) {
8       var reader = new FileReader();
9       reader.onload = function(e) {
10        var data = {type: mimeType, src: e.target.result, name: name};
11        client.socket.emit("send_sound", data);
12      };
13      reader.readAsDataURL(file);
14    } else {
15      console.log("Choose a valid audio format: ogg, wav, mp3, mpeg");
16      if (client.vibrateable) vibrate();
17    }
18  }
19 });
```

The above example shows the processing of the data to a readable object format as well as the emission of the information to the Web server. Furthermore the logic chooses to send a vibration event to the device on an error, if the client device supports the vibration feature.

5.3.4 Communication

The communication within the application system is based on the Socket.IO framework, as already mention in previous sections and seen in various code examples. Every client socket connects to the server and is then capable of transmitting data without the additional HTTP headers, which makes the connection significantly faster. As the implementation has already been presented in previous code examples it will not be visualized in this section.

As the application might sometimes handle data broadcasting itself, the RMCT collaboration object provides a method to broadcast data to every

connected client. In order to leverage transmission filtering, an additional parameter for an exception was added, which excludes the filtered socket from the emission. The method can be seen in the following code example:

```
1 collaboration.broadcast = function (command, data, exception) {
2   for (var i=0; i < clients.length; i++) {
3     if (!exception || clients[i].id != exception) {
4       clients[i].emit(command, data);
5     }
6   }
7 }
```

5.4 User Interface

The user interface is the graphical representation of a Web application and the interactive platform for users to communicate with the system. Despite the fact that the focus of this research topic is on the theoretical approach to integrate mobile devices in Web applications rather than on the UI, it is of significant importance for cross-platform applications to provide a responsive graphical interface. Therefore the RMCT provides a simple responsive UI, as it can be seen in figure 5.3 on the next page, which aims to support a satisfied interaction with the Web application and to still provide full functionality.

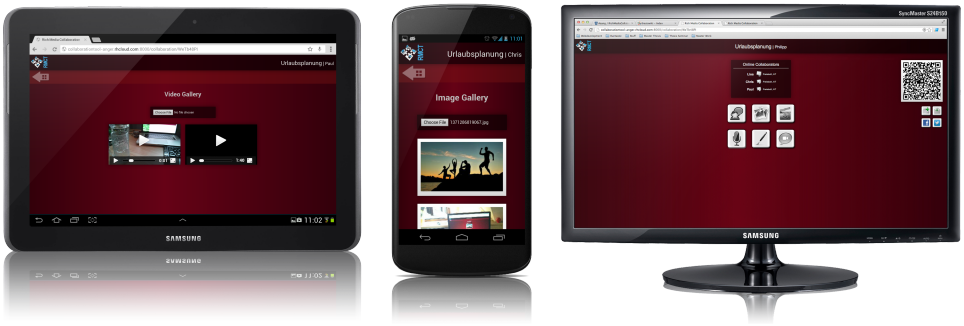

Figure 5.3: Screenshots of the cross-platform collaborative Web application *Rich Media Collaboration Tool* on various devices.

5.4.1 Basic UI elements

As this work is based on a theoretical approach, graphics design was a secondary objective, but the user interface still must provide crucial elements for supporting the collaborative purpose of the application. During research and project processing, it became clear which elements are necessary to support collaborative functionality. Besides the content interaction elements these elements are the connector, session information, collaborators box and interaction menu.

Connector

The *connector* is probably the most important element on the application page, because it is used to share the collaboration with other users. As it can be seen in figure 5.4, the connector provides four kinds of connection sharing options. Most notably besides these options is the QR-code image, which is individually generated by client-side code for each collaboration and comprehends the URL of the given collaboration. Nearly every modern smartphone or tablet has an integrated camera, which can scan and read QR-codes. Therefore this is a highly comfortable way to access an existing Web session by simply scanning the required code from an application page, as stated in [1].

Furthermore the connector provides the possibility to open a new tab in the Web browser window with the collaboration URl, which is intended for a developing purpose to test and validate the application. It also provides an email invitation button, which opens the preferred mail service on the client's device and contains an invitation message with a link to the collaboration in the message's body. The other two options – share on facebook and twitter – are only for demonstration purposes to be able to share the collaboration on social networks as well. They are have not been implemented in the prototype so far.

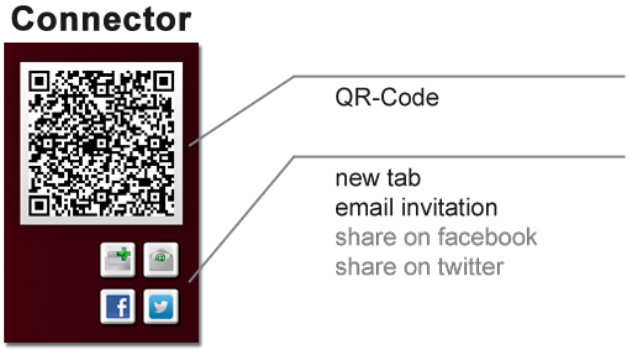

Figure 5.4: The collaboration connector with the QR-Code and sharing buttons.

Session Info

Another UI element is the *session info*. This element, as displayed in figure 5.5 on the next page, is situated at the top of the page and provides the user with basic information about the collaboration title and the user's name visible to other users. This can especially be important if a user has multiple active collaborations at a time.

Figure 5.5: Interface details about the essential UI elements: session info, collaborators box, interaction menu.

Collaborators Box

The *collaborators box* is only displayed on the home page to connected users. It visualizes the currently connected users/collaborators to this collaboration and provides information on the users' name, device type and location. This functionality is particularly important from a collaborative point-of-view, because collaborative groups/teams need to be aware of the online users for fulfilling certain tasks and facilitate the overall communication.

Interaction Menu

Collaborative applications offer numerous kinds of content interaction and content sharing. These options are displayed in the *interaction menu* on the home page of the application. It depends on the device's capabilities on which menu buttons are being displayed and therefore provided for the user. Figure 5.5 shows the basic menu for a modern smartphone with several interaction possibilities, such as chat, image gallery, video conference and whiteboard.

5.4.2 Interaction UI

The *interaction UI* represents the interface for all different kinds of content interaction types. Basically, these pages are structured similarly to the home page, but with focus on the content area. Each type provides input elements and an output view, as it can be seen in the screenshots of the four different interaction pages in figure 5.6.

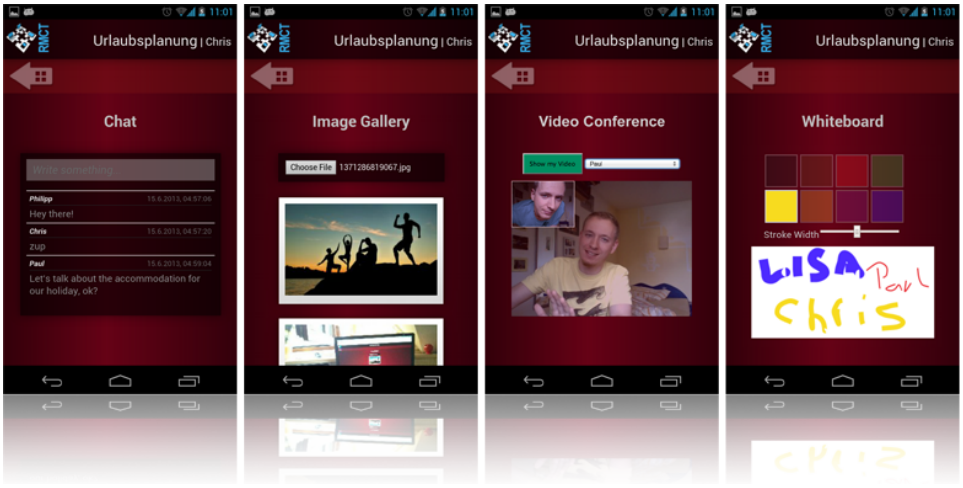

Figure 5.6: Examples for the interaction UI: chat, image gallery, video conference and whiteboard.

Chapter 6

Evaluation

Within this chapter, the evaluation of the proposed system on the basis of certain requirements as well as the realized pototypical application will be discussed. At first, the practical prototype implementation will be evaluated and described in detail upon its used technologies, predefined requirements and the conceptual approach. Following this prototype evaluation, the proposed integration architecture will be analyzed and discussed concentrating on the acquired knowledge about key aspects of the research topic. To round off the evaluation, to the results of the previous evaluation and the current state of the art. To round off the evaluation, the applicability of the integration architecture based on Web technologies will be discussed and put in contrast to the current state of the art.

6.1 Prototype

As described in chapter 5, the Rich Media Collaboration Tool (RMCT) was developed as a proof of concept for the proposed mobile integration architecture as well as to leverage new arising possibilities with standardized mobile Web technologies, as presented in section 2.2 on page 5. Since these standards are best compatible with the Firefox Mobile and the Google Chrome for Android browsers, the prototype was evaluated on those two Mobile browsers. The following sections concentrate on evaluating the developed application upon the used technologies, the requirements defined in 4.2 and most notably the conceptual approach.

6.1.1 Technical Evaluation

One emerging trend used on developing the application is the single page model, which is – as presented in section 5.1 – supposed for fully client-side applications. The RMCT perfectly conforms the requirements of this approach, because most of the application workflow is encountered at the

client and no complex contents or actions need to be processed. Especially the mobile representation of the application benefits from this model due to its never-redirecting capability which creates a more native-like look and feel for the mobile user. This feeling could even be enriched by the use of smooth animations for page swipes or menu hiding and showing at a side of the screen. One pitfall of a single page application is the fact that it breaks the page history by not reloading the page on navigation. This could produce a decrease in the user satisfaction while browsing through the application, but is basically avoidable by a convenient user interface. Furthermore, a collaborative Web application could have a lot of content that would be initially loaded on connection and therefore results in high traffic costs.

Another benefit and pitfall at the same time is the use of JavaScript libraries and jQuery plugins. On the one hand, the Web offers a platform for developers to share open source code with the World, which is an enormous profit for developing Web pages and Web applications. On the other hand, a lot of external code decreases the code's readability and increases the page's overall traffic.

6.1.2 Cross-Platform Compatibility

Generally, the application is cross-platform compatible with most tested browsers, except the BlackBerry Browser and the desktop and mobile Internet Explorer. This is basically due to their lack of Web standard implementations, different implementations or problems caused by the JavaScript libraries used in the application. The project does not concentrate on creating workarounds to match those individual environmental requirements but rather investigates the possibilities of the given Web standards. Therefore those minor cross-platform compatibility issues can be resolved by further concentrating on this topic in future developments.

6.1.3 Multi-User Capability

One significant difference to the integration architecture is the lack of user roles, which were not part of the evaluation goal for the prototype. User roles are necessary in order to provide full collaborative functionality in future projects.

Nevertheless, the prototype is capable of accepting and managing multiple users at a time by its fast socket communication and especially by the event-driven server. Each user is thereby equally handled by the application logic.

6.1.4 Asynchronous Communication

The prototype uses Node.js and Socket.IO as basic framework technologies, which saves time and effort to set up a simple server and socket connec-

tion. Furthermore this combination is extremely fast compared to traditional server implementations and gives the application excellent cross-browser support for the socket communication. But it is still quite difficult to find a suitable Web hosting provider fulfilling the requirements, such as WebSockets.

6.1.5 State Persistence

As the prototype is a non-database-driven system, the server is mainly aware of the stored data and states. The integration architecture wants the server to be only responsible for the event communication controlling between the connected clients, which can be achieved by integrating a database to the application system, guaranteeing long-time data storing and therefore state persistence.

State persistence is guaranteed within the developed prototype due to the broadcast exchanging routine on state and content changes. Although collaborations will be closed after a certain period of inactivity and, hence all content data and saved states will then be deleted.

6.1.6 Conceptual Approach

The proposed conceptual approach presents a layer-based integration architecture to improve the mobile integration in Web applications. These layers – described in section 4.1 – basically separate the application into multiple layers for the Web server, database, communication, logic and context-awareness. Consequently, the prototype's architecture oriented towards this structure from a software engineering point of view.

Concentrating on this layer system, the main application logic was situated on the client-side as well as the context totally separated from the client-side logic. The separation of the client logic and the context-aware layer was thereby implemented in two different JavaScript classes. The client logic class is aware of all the states and follows certain conditions, but also the context-aware layer needs to be aware of state information in some rare cases when it comes to input processing. One example in the prototype is video streaming, where the state of the stream is changed from within a context listener, but this code could also be extracted to the client-side logic as well as all socket event emits. Overall the logic and context can be successfully and reasonably separated.

It takes some additional programming effort to strictly separate the client-logic from the context adaption and processing, but the implementation significantly improves readability of the application code and the workflow. Additionally, this separation leverages code encapsulation which is reasonable for an automated cross-platform module like the context-aware layer.

6.2 Integration Architecture

The integration architecture is the key aspect of this approach, because it provides is significantly important to concentrate on standardized Web specifications in order to facilitate W3C's vision of One Web. Section 2.1.3 already presented the term *traditional*, which stands for Web technologies such as JavaScript, HTML and CSS. This means accessing context information on mobile devices without any native (implemented with a mobile SDK) or hybrid (implemented in HTML5, compiled into other platform application code) solutions. The following sections list some benefits and pitfalls of the web-based integration compared to native- and hybrid integration, as well as basic Web application approaches for building collaborative applications. This information was acquired during the research process, work on this document and the practical realization of the system as a prototype.

6.2.1 Context Handling

With the use of the Web standard APIs, context information can be handled according to certain standards and processed as preferred in the application logic. The integration architecture provides the context-aware layer for detecting, accessing and processing context information, as described in 4.4. The presented layer separation enables context handling independent from the application's logic. Particularly the readability of the context implementation is significantly improved by this technique, because every possible action the logic can take is provided as a function call within the structured target area of context detection, context initialization/access, input processing, output processing and interface adaption.

Different situations may require different context handling, for instance if an application logic needs an uploaded image file to be processed in a certain way before transmitting it to the application logic, it would be necessary to add some logic to the the context-aware layer as well. The prototypical application has shown that it is reasonable in some situations to add logic to the context-aware layer in order to save developing effort and traffic costs.

Generally, separating the application logic and the context-awareness is a reasonable supporting aspect for the mobile device integration. Another opportunity arising from this approach is to develop the context handling functionality as a module, which will be described in the following section.

6.2.2 Interaction Features

As already discussed and presented in figure 2.1 on page page 9, the feature support for devices' interaction capabilities is insufficient for a wide range of technologies, but quite good for some basic technologies including HTML5 audio element, HTML5 video element, HTML Canvas 2D and Touch Events.

Mobile Web standard technologies represent an evolving and thriving developing area which is more and more accepted by browser developers. Furthermore, the Mobile Web experiences a steady growth in technological possibilities by emerging developing processes in the mobile device industry. Especially hardware improvements on mobile devices and the broadcasting environment are increasing accessibility, performance and usability. This trend facilitates future Web development and is expected to be utilized in a broad variety of future Web projects.

API support is a crucial topic particularly for collaborative Web applications which should provide the best possible interaction. Thereby, the integration architecture is responsible for the automated adaption of the system to the given environment while retaining the application's core functionality. Of course, a collaborative Web application heavily depends on its interaction possibilities, which are sufficiently supported for primal interaction types including touch, geolocation, device orientation, device direction, images, canvas sketches, messages and most audio and video files.

Nevertheless, it still takes a lot of development effort and workarounds to suit every environment with the current state of the art of the APIs. Even if the all browsers have implemented the same standards, there can still be major differences in other areas affecting those technologies, for instance the file type support for audio and video files in the various browsers. Figure 6.1 illustrates the different file type support tested in some of the most widely spread desktop and mobile browsers (Android and iOS). As shown in the figure, iOS browsers did not support any of the tested file types. Moreover,

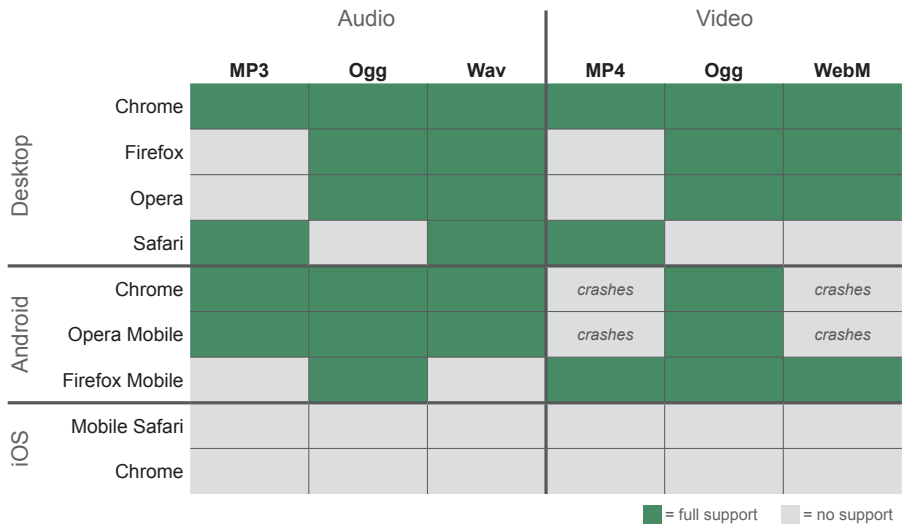

Figure 6.1: Support of audio and video file types as tested in different browsers through the pototypical application.

playing apparently supported files – types and codecs are being tested for each and every file before it is rendered and processed on the page – will also result in the application to crash in the mobile versions of Google Chrome and Opera, as shown in the figure.

6.2.3 Benefits

Research and work on the proposed integration architecture pointed out some significant benefits for the use of Web technologies to integrate mobile devices in collaborative Web applications. Benefits for developers as well as for users are pointed out in the following list:

Autonomy
: The system is able to autonomously detect and adapt to the given environment by its context information. Hence, the application is not dependent of the Web browser for the core application to be executable.

Code Readability
: Separating the logic and context-awareness implementation creates a more readable code. This especially benefits Web developers when working with JavaScript for the logic processes and jQuery to adapt and manipulate the context in the view representation.

Accessibility
: The Web application can be accessed from anywhere and from any device. It does not matter where and which device the collaborator is accessing the application from.

One application
: The application is developed once and runs everywhere, while native applications need to be developed separately for each platform and hybrid applications may require a lot of additional adaptions for other platforms.

One language
: It is possible to build the whole application system in one language. This prevents language compatibility issues and leverages faster development. The pototypical application RMCT is completely implemented in JavaScript.

No installation
: As it is a Web application, no explicit software installation and updates are needed. Every modern mobile device provides a built-in Web browser and the application is updated directly on the hosting server.

Speed
: Client-centric applications built on Node.js and Socket.IO considerably increase the communication speed compared to standard Web applications.

6.2.4 Pitfalls

On the contrary, there are also various crucial pitfalls of this integration architecture for Web applications. Again emphasized on development aspects as well as on a usability point of view, the following list gives an overview of some pitfalls:

JavaScript
: JavaScript is essential as a scripting language to access the Web APIs implemented by the Web browser, but it can be manually disabled by the user effecting the application to not work properly or brake.

Internet connection
: Web applications usually need a persistent connection to the Internet. It is possible to store data locally on the client device with Web standard APIs, but these technologies are not as efficient as native local data storage.

Performance
: Client scripting needs more processing time than native applications, because the Web browser interprets the commands rather than the native platform. However, V8 is a great performance booster for Web applications.

Browser support
: Unfortunately the interaction possibilities of users with the Web application and vice versa are still heavily dependent on the used Web browser. Hence, a lot of browser-dependent workarounds are necessary.

Programming
: It needs advanced JavaScript skills for developers to implement readable and separated code, because JavaScript is not an object-orientated programming language. Therefore it is difficult to apply advanced software architectures to such a system.

6.3 Applicability in the Real World

In this section, the integration system will be evaluated according to its applicability in the Real World. The evaluation will particularly focus on the applicability from a developing point-of-view as well as the usability for a collaborative system. The evaluation is based on the current state of the art and the developed prototype.

6.3.1 Developing costs

When concentrating on the use of Web standard APIs to create collaborative Web applications, it may result in higher developing costs compared to

native or hybrid application development depending on the APIs used. Evaluating the state of the art and the proof of concept pototypical application shows that it can require costly workarounds to achieve full cross-platform compatibility. In future, when Web standards are more widely supported, the integration architecture can provide a simple engineering approach to structure the application's client-side implementation and thereby save developing costs.

6.3.2 Functional Scalability

This paragraph concentrates on the functional scalability of the integration system, which describes the possibility to easily add functionality to the existing system. Considering the layer-based architecture, it should be easy to add additional functionalities to a specific layer, as the communication, logic and context are separated from each other. For instance, adding Web APIs can be done in the context-aware layer with functions provided for the logic layer. Another example could be the integration of an additional tool using the same context information as an existing tool, without the need to change the context-aware layer but only add the functionality to the logic layer. Taking a closer look at the prototype implementation shows that the single page model additionally facilitates functional scalability in the view by its recommended view presentation in a content-oriented structure.

6.3.3 Collaborative Usability

As the integration approach not only focuses on the use of traditional Web technologies to integrate mobile devices in Web applications, but also on the integration in a collaborative application, the usability of this system in a collaborative environment needs to be evaluated. Collaborative applications – as stated in section 2.1.2 – provide a centrally shared space for multiple users to work together. Therefore the system needs to conform to certain requirements, which will be stated and analyzed according to the integration system in the following list:

Multi-user capability

As the name already implies, collaborative applications need to be capable of handling multiple users for a session at a time. The integration architecture provides a certain structure for collaboration and client instances in order to support multi-user access. Technologies and the hosting server should be chosen wisely to support safe and quick access to the stored data. Another useful functionality for a group or team is the information sharing of the collaborators state, used as the collaborators box in the prototype (described in section 5.4.1).

Accessibility
: The application should be accessible from anywhere and from any device. Web applications basically provide this functionality unless the user is not connected to the Internet. Especially collaborative applications profit from the increasing accessibility of mobile devices, because they can be enriched by rich-media content.

Continuity
: Collaboration sessions and their contents should be saved in a database as long as the session is needed. The collaboration state and content is synchronized by the logic layers in order to provide the application's continuity for each client.

Live
: The communication needs to simulate a live face-to-face communication as much as possible. Therefore, instant communication is crucial for a satisfying collaboration. The architecture simplifies this process by the separation of the communication layer and the use of modern technologies such as Node.js and Socket.IO.

6.4 Future Work

Various possible improvements emerged from research process and work along this document. The following two examples can be considered for future work in this research field.

6.4.1 Using Polyfills

In section 4.4.1 of the proposed system, the possibility of using polyfills to detect and process context information was introduced. Polyfills are code snippets that implement required features if they are not supported by the target Web browser. This technology can be used to reduce the previously described compatibility issues to a minimum. Although it should only be a temporary solution, because Web standards are aiming to a standardized Web where no alternative scripts – which additionally increase traffic and maintenance costs – should be needed to achieve cross-platform compatibility. The developed pototypical application did not use any polyfills in order to be able to evaluate the current state of the art of Web standards.

6.4.2 Module Pattern

As context handling is extracted from the application logic, it can also be implemented as a module in other applications to support a standardized cross-platform access to context information by Web technologies. The central theme of module context handling is to leverage the mobile device integration in Web applications. This module pattern was partly introduced by

the Rich Media Collaboration Tool, but combined with interface adaption according to processed context. In order to create a fully modular context handling, the module needs to focus only on the communication with the browsers' feature API implementations. View presentations and logic processes always depend on the specific application system and should not be part of the context-aware module.

However, including interface adaption within the context-aware layer turned out to be an acceptable solution for a fully satisfied environment where the developer is aware of the DOM structure and the application logic. This saves programming costs and still provides a certain separation between the logic and its context.

Chapter 7

Conclusion

Mobile devices has been emerging on the market with rapid technological development for the last few years. Smartphones and tablets expose a noticeable demand for full accessibility, seamless functionality and instant information sharing across platforms. This trend has led to the development of several approaches in the field of Mobile Web, such as Web applications, hybrid apps and native apps. Web developers are thereby faced with the challenge to adapt to this new environment.

This trend has mainly led to well-adapted Web Applications rather than a mobile device integration due to its lack of API support in mobile Web browsers. Although recent developments in the Mobile Web introduced numerous Web features mobile devices are still mainly handled as smaller screen environments with minor additional capabilities like sensing touch events or defining the device's location.

Based on this research area, this work provided an overview of the given possibilities and the current state of the art for mobile device integration in Web applications, with a particular focus on online collaborations. The analysis and discussion of the presented technologies and approaches revealed the insufficient cross-platform compatibility and the focused interface adaption and migration as the main reasons for the lack of mobile device integration.

To address this issue, a layer-based mobile integration architecture was presented to facilitate access and processing of rich contextual information. Each layer was discussed by its functionality and necessity within the application as well as by its particular utilization for online cross-platform collaborations.

In order to analyze the practicability of the proposed approach, the prototypical collaborative application *Rich Media Collaboration Tool* was developed and published on a hosting service. The prototype particularly focuses on an autonomic integration of multiple devices and the utilization of as many interaction capabilities as possible using Web technologies.

As a result of the prototypical implementation, the application was able

to deal with the lack of cross-platform compatibility by automatically detecting the device-dependent context information and adapting the application accordingly to provide the application's basic functionality. Thereupon, numerous interaction capabilities were integrated in the prototypical application and offered fast rich-media collaboration between multiple devices and users.

Nevertheless, full cross-platform compatibility cannot be provided with the current state of the art and the given capabilities of mobile devices. This may not be achieved in near future, but the approach provides an architecture to build fully functional applications adapted to their environment and capabilities. The evaluation also showed the complexity and the additional development effort necessary for implementing the proposed architecture. Consequently, the structure sometimes had to be adjusted to provide the desired functionality.

Additionally, some further improvements to the approach – like using polyfills to achieve full cross-platform compatibility or creating the context layer as a module pattern for autonomy – were presented for future work.

Integrating mobile devices in Web applications is being facilitated by recent and ongoing developments of Web standardization institutions, but the full integration is still highly limited by the target platform and therefore far from full cross-platform compatibility. Still these APIs already provide the ability for developers to utilize a noticeable number of the devices' interaction capabilities and thereby to enrich online collaborations and other Web services.

In conclusion, mobile devices can be integrated in collaborative Web applications by autonomously detect the device's capabilities and adapt to its environment by using its context information as well as separating these processes from the application logic.

Appendix A

Contents of the DVD-ROM

Format: DVD-ROM, Single Layer, ISO9660-Format

A.1 PDF files

Pfad: /

Anger_Philipp_2013.pdf Master thesis

Pfad: /Online_Sources

AsynchVsSynch.pdf . . [24]
SpaceWords.pdf . . . . [25]
HybridVsNative.pdf . . [26]
ShieldAttack.pdf . . . . [27]
W3CStandards.pdf . . . [28]

A.2 Source Code

Pfad: /Code

RMCT . . . . . Rich Media Collaboration Tool – prototypical implementation of the mobile integration architecture as described in chapter 4

References

Literature

[1] Alexandre Alapetite. "Dynamic 2D-barcodes for multi-device Web session migration including mobile phones". In: *Journal of Software* 14.1 (2010), pp. 45–52.

[2] Federico Bellucci et al. "Automatic reverse engineering of interactive dynamic Web applications to support adaptation across platforms". In: *International Conference on Intelligent User Interfaces, Proceedings IUI*. (Lisbon). New York: Association for Computing Machinery, Feb. 2012, pp. 217–226.

[3] Federico Bellucci et al. "Engineering JavaScript state persistence of web applications migrating across multiple devices". In: *Proceedings of the 2011 SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2011*. (Pisa). New York: Association for Computing Machinery, June 2011, pp. 105–110.

[4] R. Bezerra Braga et al. "A Context-Aware Web Content Generator Based on Personal Tracking". In: *Web and Wireless Geographical Information Systems. Proceedings 11th International Symposium, W2GIS 2012*. (Naples). Berlin: Springer Verlag, Apr. 2012, pp. 134–50.

[5] Bin Cheng. "Virtual browser for enabling multi-device web applications". In: *Proceedings of the International Workshop on Multi-Device App Middleware 2012, Multi-Device 2012 - Co-located with ACM/IFIP/USENIX 13th International Conference on Middleware*. (Montreal). New York: Association for Computing Machinery, Dec. 2012.

[6] Jordan Pascual Espada et al. "Extensible architecture for context-aware mobile web applications". In: *Expert Systems with Applications* 39.10 (2012), pp. 9686–9694.

[7] Maximiliano Firtman. *Programming the Mobile Web*. 1st ed. Sebastopol: O'Reilly Media, 2010.

[8] Adam Freeman. *Pro JavaScript for Web Apps*. 1st ed. New York: Apress Media, 2012.

[9] Giuseppe Ghiani, Lorenzo Isoni, and Fabio Paterno. "Security in migratory interactive web applications". In: *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia, MUM 2012*. (Ulm). New York: Association for Computing Machinery, Dec. 2012.

[10] Giuseppe Ghiani, Fabio Paterno, and Carmen Santoro. "Push and pull of web user interfaces in multi-device environments". In: *Proceedings of the 2011 SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2011*. (Capri Island). New York: Association for Computing Machinery, May 2007, pp. 10–17.

[11] Wesley Hales. *HTML5 and JavaScript Web Apps*. 1st ed. Sebastopol: O'Reilly Media, 2012.

[12] I-Ching Hsu. "An architecture of mobile Web 2.0 context-aware applications in ubiquitous Web". In: *Journal of Software* 6.4 (2011), pp. 705–715.

[13] Georgia M. Kapitsaki, Dimitrios A. Kateros, and Iakovos S. Venieris. "Architecture for provision of context-aware web applications based on web services". In: *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*. (Poznan). Piscataway: Institute of Electrical and Electronics Engineers, Sept. 2008.

[14] Wazir Zada Khan et al. "Mobile phone sensing systems: A survey". In: *IEEE Communications Surveys and Tutorials* 15.1 (2013), pp. 402–427.

[15] Nicholas D. Lane et al. "A survey of mobile phone sensing". In: *IEEE Communications Magazine* 48.9 (2010), pp. 140–150.

[16] Bruce Lawson and Remy Sharp. *Introducing HTML5, Second Edition*. 2nd ed. Berkeley: New Riders Press, 2011.

[17] Salvatore Loreto and Simon Pietro Romano. "Real-time communications in the web: Issues, achievements, and ongoing standardization efforts". In: *IEEE Internet Computing* 16.5 (2012), pp. 68–73.

[18] Alex MacCaw. *JavaScript Web Applications*. 1st ed. Sebastopol: O'Reilly Media, 2011.

[19] Lora Oehlberg et al. "Dazzle: Supporting framing in co-located design teams through remote collaboration tool". In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*. (Seattle). New York: Association for Computing Machinery, Feb. 2012, pp. 183–186.

[20] William Van Woensel, Sven Casteleyn, and Olga De Troyer. "A generic approach for on-the-fly adding of context-aware features to existing websites". In: *HT 2011 - Proceedings of the 22nd ACM Conference on Hypertext and Hypermedia* (2011), pp. 143–152.

[21] C. Vivaracho-Pascual and J. Pascual-Gaspar. "On the Use of Mobile Phones and Biometrics for Accessing Restricted Web Services". In: *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)* 42.2 (2012), pp. 213–22.

[22] Pawan Vora. *Web Application Design Patterns*. 1st ed. Burlington: Morgan Kaufmann Publishers, 2009.

[23] A.G. West et al. "Trust in collaborative web applications". In: *Future Generation Computer Systems* 28.8 (2012), pp. 1238–51.

Online sources

[24] *Asynchronous vs. Synchronous Communication*. URL: http://academictech.doit.wisc.edu/ideas/otr/communication/asynchronous-synchronous (visited on 06/29/2013).

[25] James Burke. *Using Node.js and your phone to control a Browser game*. 2011. URL: http://cykod.com/blog/post/2011-08-using-nodejs-and-your-phone-to-control-a-browser-game (visited on 06/29/2013).

[26] Ata Sasmaz. *Hybrid vs. Native Mobile Apps*. 2013. URL: http://www.optimum7.com/internet-marketing/web-development/hybrid-vs-native-mobile-apps-html5.html (visited on 06/29/2013).

[27] *Shield Attack*. 2012. URL: http://www.unit9.com/project/shield-attack (visited on 06/29/2013).

[28] *Standards for Web Applications on Mobile: current state and roadmap*. URL: http://www.w3.org/2013/02/mobile-web-app-state/ (visited on 06/29/2013).