

Department of Informatics  
University of Fribourg (Switzerland)

# TWICE

A Toolkit for Web-based Interactive Collaborative Environments

THESIS

PRESENTED TO THE FACULTY OF SCIENCE OF THE UNIVERSITY OF FRIBOURG  
(SWITZERLAND) IN CONSIDERATION FOR THE AWARD OF THE ACADEMIC GRADE OF  
DOCTOR SCIENTIARUM INFORMATICARUM

BY

OLIVER SCHMID

FROM

WINTERTHUR ZH, SWITZERLAND

**Thesis No: 1790**

UniPrint  
May 2013

Accepted by the Faculty of Science of the University of Fribourg (Switzerland) upon the recommendation of

- Prof. Dr. Ulrich Ultes-Nitsche, University of Fribourg, Switzerland (Jury president)
- Prof. Dr. Béat Hirsbrunner, University of Fribourg, Switzerland (Thesis supervisor)
- Prof. Dr. Harald Reiterer, University of Konstanz, Germany (Expert)
- Dr. Agnes Lisowska Masson, University of Fribourg, Switzerland (Expert)
- Dr. Michèle Courant, University of Fribourg, Switzerland (Expert)

Fribourg, 05-14-2013

Thesis supervisor



Prof. Dr. Béat Hirsbrunner

Dean



Prof. Dr. Fritz Müller

In memory of my father who introduced me to the fascinating world of technology and inspired me by his drive for innovation. All of my career in academics and industry, and especially this thesis, would never have been possible without him sharing his passion for information technology with me.



# Acknowledgements

While working on this thesis, I have met many people which have influenced my thoughts, inspired me, challenged me, supported me in moments of doubts and shared the joy and enthusiasm when things worked out. Many thanks to all these people who contributed a lot to the final result as it is presented in this thesis.

My special thanks go to my supervisor Béat Hirsbrunner for giving me the opportunity to work on this thesis as well as giving me the freedom to explore an interesting topic. Special thanks go to Agnes Lisowska Masson who was always available for questions and discussions and who always found new ideas to challenge me. I also want to thank Apostolos Malatras, especially for his appreciated criticism which always drove me to do better. Thanks to Muriel Bowie who contributed a lot to the basic ideas of the toolkit, as well as to Michèle Courant for her support as well as her feedback. Many thanks also go to Prof. Dr. Harald Reiterer for taking part in the jury and sharing his expertise on this subject. I also want to thank my other friends and colleagues at the Department of Informatics in Fribourg for their support during these years. In no particular order, my gratitude goes to Denis Lalanne, Daniel Ostojic, Pascal Bruegger, Caroline Voeffray, Roman Baeriswyl, Maurizio Rigamonti, Fei Peng, Ammar Halabi, Benjamin Hadorn, Patricio Lerena, Fulvio Frappoli, Amos Brocco, Mohamed Momouh Khadraoui and Nicolas Juillerat.

For their appreciated collaboration, I want to thank Andreas Sonderegger from the Department of Psychology in Fribourg and David Weibel from the Department of Psychology in Bern as well as their students, who were involved in the collaboration. Special thanks go to Lorenz Roten who made it possible to do experiments in the Gymnasium Neufeld in Bern and who supported me by his contagious enthusiasm. My gratitude also goes to my co-workers at Puzzle ITC and EWB who supported me whenever they could in the job I was working in parallel doing my thesis and especially to Mark Waber, Thomas Weber and Ruedi Hofer for providing me with all the flexibility needed in terms of holidays and working hours to make the combination of the PhD thesis and industrial work possible.

Thanks also to my family who supported me through the whole time and who were always there when I have needed them. Last but not least my very special thanks go to my girlfriend Franziska for her love and her tolerance of the nights spent in front of the computer and my mood regularly switching from frustrated to enthusiastic. Without her support, this thesis would probably not exist.

# Abstract

The number of available personal devices (e.g. smart phones, tablets, game consoles, laptops) has increased rapidly in the last years. Their increasing capabilities enable complex interaction which might – in combination with their mobility and networking functionalities – lead to spontaneous computer supported collaboration in non-predefined locations. Additionally, already existing and potentially infrastructurally supported (by the availability of big screens and wireless networks) collaborative environments (e.g. meeting rooms) can be extended through the integration of personal devices. The heterogeneity as well as the spontaneity of such novel collaborative systems implies multiple challenges in terms of software engineering which have not been fully overcome yet. Therefore, we developed a toolkit for the development of collaborative systems based on standard web technologies, which allows to develop applications which are executable on any device that provides a network connection and contains an installed web browser without the need to install or configure anything on this device.

In addition to providing solutions for the fundamental issues of the development of real-time web applications (e.g. missing bi-directional communication) our toolkit includes concepts, structures and functionalities for the simplified development of multi-user systems (e.g. multi-pointer and multi-focus extensions) and enables dynamic adaptation of software functionalities depending on device-dependant specificities (e.g. screen size, input modalities, capabilities). Additionally, the toolkit simplifies development and testing by presenting concepts to reduce differences in the development process between single- and multi-user applications. The toolkit therefore not only provides necessary functionalities to execute extensive research in dynamic collaborative systems, but also builds the base for the development of fully functional end-user systems which can support collaboration in everyday situations.

Although during specification and development of the toolkit our focus was on colocated synchronous collaboration, we made sure that the technologies and the concepts introduced are applicable for remote and/or asynchronous collaboration as well.

**Key words:** Computer supported collaboration, web technologies, real-time web application, distributed system, multi-user, ad-hoc collaboration

# Zusammenfassung

Die Anzahl der verfügbaren persönlichen Geräte (z.B. Smart-Phones, Tablets, Gamekonsolen, Notebooks) ist in der letzten Zeit rasant angestiegen. Der zunehmende Funktionsumfang dieser Geräte ermöglicht komplexe Interaktionen, welche in Kombination mit der Mobilität und Netzwerkfähigkeit zu spontaner computerunterstützter Kollaboration an nicht vordefinierten Orten führen kann. Zudem können bereits existierende, u.U. infrastrukturell unterstützte (Vorhandensein von grossen Bildschirmen und kabellosen Netzwerken) kollaborative Einrichtungen (z.B. Meetingräume) durch die Integration persönlicher Geräte dynamisch erweitert werden. Mit der Heterogenität sowie der Spontanität mit welcher diese neuartigen kollaborativen Systeme auftreten können gehen diverse Herausforderungen für die Software-Entwicklung einher, welche bis anhin nur partiell gelöst sind. Aus diesem Grund haben wir ein Toolkit für die Entwicklung von kollaborativen Systemen auf der Basis von Standard Web-Technologien entwickelt, welches es erlaubt Applikationen zu entwickeln, welche installations- und konfigurationsfrei auf jedem Gerät mit Netzwerkverbindung und installiertem Web-Browser ausgeführt werden können.

Neben Lösungen für grundlegende Probleme bei der Entwicklung von Echtzeit-Webapplikationen (z.B. bi-direktionale Kommunikation) stellt unser Toolkit Konzepte, Strukturen und Funktionalitäten für die vereinfachte Entwicklung von Mehr-Benutzer Systemen zur Verfügung (z.B. Multi-Pointer und Multi-Fokus Erweiterungen) und erlaubt es, Funktionalitäten der Software dynamisch an gerätespezifische Eigenheiten (z.B. Bildschirmgrösse, Eingabemodalitäten, Leistungsfähigkeit) anzupassen. Zudem vereinfacht das Toolkit die Entwicklung und das Testing eines kollaborativen Systems indem es Konzepte aufzeigt um die Unterschiede im Entwicklungsprozess zwischen Ein- und Mehr-Benutzer-Applikationen zu reduzieren. Das Toolkit stellt somit nicht nur notwendige Funktionalitäten zur Verfügung um umfangreiche Forschung in dynamischen kollaborativen Systemen zu betreiben, sondern bietet auch Grundlagen zur Entwicklung von voll funktionsfähigen End-Benutzer-Systemen, welche Kollaboration in Alltagssituationen unterstützen können.

Obwohl der Fokus während der Spezifikation und der Entwicklung des Toolkits auf orts- gleiche synchrone Kollaboration gesetzt wurde, haben wir sichergestellt, dass die verwendeten Technologien und Konzepte auch bei entfernten und/oder asynchronen Kollaborationen angewendet werden kann.

**Schlüsselwörter:** Computerunterstützte Kollaboration, Web-Technologie, Echtzeit Webapplikation, verteilte Systeme, Mehr-Benutzer, Ad-Hoc Kollaboration

# Table of Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Zusammenfassung</b>	<b>iii</b>
<b>1. Introduction</b>	<b>1</b>
<b>2. State of the art / Related work</b>	<b>6</b>
2.1. Computer supported collaboration . . . . .	7
2.1.1. General challenges . . . . .	7
2.1.2. Types of collaborative systems . . . . .	8
2.1.3. Ad-hoc vs. infrastructure-based collaboration . . . . .	11
2.1.4. Devices involved . . . . .	11
2.1.5. Problem space . . . . .	13
2.2. System architecture of distributed systems . . . . .	15
2.2.1. System dynamics . . . . .	15
2.2.2. System availability . . . . .	18
2.2.3. System scalability . . . . .	19
2.3. Platform- and device-heterogeneity . . . . .	19
2.4. Eventing and synchronization . . . . .	22
2.5. Security and privacy . . . . .	25
2.6. Interaction with shared devices . . . . .	29
2.7. Aspects of user experience . . . . .	30
2.7.1. User awareness . . . . .	30
2.7.2. Distribution of user interfaces . . . . .	30
2.8. Toolkits and solutions for collaborative applications . . . . .	32
2.8.1. Multi-user / multi-device support . . . . .	33
2.8.2. Extension of legacy apps . . . . .	34
2.8.3. Communication . . . . .	34
2.8.4. Distribution of user interfaces . . . . .	35
2.8.5. General toolkits . . . . .	35
2.9. Discussion . . . . .	36
<b>3. Specification</b>	<b>38</b>
3.1. Context of the work . . . . .	39
3.2. Requirements . . . . .	39



---

3.2.1. Technology . . . . .	39
3.2.2. Software architecture . . . . .	42
3.2.3. Basic functionalities . . . . .	43
3.3. Choice of technology . . . . .	45
3.3.1. Priorization of requirements . . . . .	46
3.3.2. Technology candidates . . . . .	47
3.3.3. Technology decision . . . . .	58
3.3.4. Implications and challenges . . . . .	62
3.4. System architecture . . . . .	65
3.4.1. Overview . . . . .	65
3.4.2. Work load distribution . . . . .	67
3.4.3. Server side functionalities . . . . .	67
3.4.4. Client side functionalities . . . . .	69
3.4.5. Communication . . . . .	70
3.4.6. Set-up . . . . .	71
3.5. Discussion . . . . .	72
<b>4. Toolkit</b>	<b>73</b>
4.1. General concepts . . . . .	74
4.1.1. Modules . . . . .	74
4.1.2. Coding conventions and concepts . . . . .	78
4.2. Basic functionalities . . . . .	81
4.2.1. Distributed eventing mechanism . . . . .	82
4.2.2. Security . . . . .	85
4.2.3. Device grouping . . . . .	87
4.2.4. Layouting . . . . .	87
4.2.5. Multi-user support . . . . .	90
4.2.6. Easy access . . . . .	93
4.3. Software modules . . . . .	94
4.3.1. Drag and drop . . . . .	94
4.3.2. Remote mouse controller . . . . .	96
4.3.3. Remote keyboard . . . . .	98
4.3.4. Extended widgets for multi-user and multi-device contexts . . . . .	99
4.3.5. Collaborative web browsing . . . . .	102
4.4. Use of the toolkit in practice . . . . .	103
4.4.1. Module development . . . . .	104
4.5. Comparison with standard GWT . . . . .	105
4.6. Discussion . . . . .	106
<b>5. Evaluation and real world use</b>	<b>108</b>

---

5.1. Technical evaluation . . . . .	109
5.1.1. Performance . . . . .	109
5.1.2. Heterogeneity . . . . .	114
5.1.3. Scalability . . . . .	115
5.2. In-use evaluation . . . . .	116
5.2.1. Initial experiments . . . . .	116
5.2.2. A Fitt of distraction . . . . .	118
5.2.3. Distributed user interface experiments . . . . .	119
5.2.4. Computer supported brain storming . . . . .	121
5.2.5. Usability experiments . . . . .	122
5.2.6. Multi-Zoom . . . . .	123
5.3. A modular mindmap application . . . . .	124
5.3.1. The latest version of the mindmap application . . . . .	125
5.3.2. Non-integrated components . . . . .	127
5.3.3. Modularity in practice . . . . .	128
5.4. Real-world experiment: Use in an educational scenario . . . . .	128
5.5. Developer evaluation of the toolkit . . . . .	133
5.6. Discussion . . . . .	136
<b>6. Conclusion</b>	<b>138</b>
<b>A. Acronyms</b>	<b>143</b>
<b>B. Resources and extended code extracts</b>	<b>145</b>
<b>C. Website of the Project</b>	<b>152</b>
<b>Bibliography</b>	<b>153</b>
<b>Referenced Web Resources</b>	<b>162</b>
<b>Curriculum Vitae</b>	<b>165</b>

# List of Figures

2.1. The CSCW matrix – based on [96] . . . . .	8
2.2. The main issues of computer supported collaborative systems and their impact depending on the characteristics of a system . . . . .	14
2.3. Separatable GUI components in the example of the image editing software GIMP [99] . . . . .	31
3.1. Comparison matrix for technology candidates . . . . .	59
3.2. System architectures: Client-server with optional cloud resources (A), peer2peer (B) and hybrid peer2peer (C) . . . . .	66
3.3. Direct communication between clients is not (yet) possible with web technologies and therefore has to be rerouted by the server . . . . .	68
4.1. The IDE Eclipse (left) as a template for the design of the dynamic layout for big screen and cursor oriented devices (right). . . . .	88
4.2. The mobile layout of Google+ (left) and the TWICE toolkit’s dynamic layout (right): The menu button (A) is positioned on the top left and when pressed, the menu bar (B) appears from the left. . . . .	89
4.3. Control of the mouse cursor: A, B and C send the mouse control information to a server which reroutes them to the shared device. Within the response, the clients get information about the pixel size of the target screen and about the color of the pointer which they are controlling . . . . .	97
4.4. Activity diagram of a multi-focus textbox with two devices (A=red and B=blue)	101
5.1. Initial experiments . . . . .	117
5.2. A Fitt of distraction – Point-and-select tasks for distraction measurement . .	119
5.3. Distributed user interface experiments . . . . .	120
5.4. Brain storming application . . . . .	121
5.5. Usability experiments . . . . .	122
5.6. Multi-Zoom . . . . .	123
5.7. Shared mindmap screen . . . . .	126
5.8. Mindmap of the students’ opinions about nuclear power plants . . . . .	129
5.9. Load of the WLAN interface on the router during the execution of the real-world use experiment (involving text addition and moving objects) . . . . .	133
B.1. Results of the original questionnaire in its original version . . . . .	149
B.2. Translation of the results of the original questionnaire . . . . .	150
B.3. Results of the user evaluation of the system in the real world use case . . . . .	151

# List of Tables

5.1. SunSpider JavaScript benchmark results of selected devices (total values – lower values are better) . . . . .	112
5.2. Devices tested for basic support of the toolkit . . . . .	115

# List of Code Extracts

4.1. Definition of the property for distinction of implementations in deferred binding	76
4.2. Use of a property for the choice of a device specific implementation . . . . .	77
4.3. Declarative widget description in GWT . . . . .	81
4.4. Binding of the declarative widget description by annotation in GWT . . . . .	81
4.5. A custom remote event . . . . .	86
4.6. Creation of a remote event and firing through the event bus . . . . .	86
4.7. An example of a draggable widget . . . . .	95
4.8. A GWT module descriptor for libraries . . . . .	104
B.1. An example POM-file for library components . . . . .	145
B.2. An example POM-file for module components . . . . .	147



# 1

## Introduction

The introduction of new devices and device types on the electronics market has continuously accelerated in recent years and personal consumer devices are now introduced on an almost daily basis. In Switzerland, in March 2012 48% of the population between 15 and 74 years of age owned a smartphone (cp. [119]), which is comparable to other industrial countries (e.g. the USA – cp. [120]). For collaborative settings, this development has a rather big impact. In ad-hoc scenarios, where people meet spontaneously, they are now equipped with powerful devices which can support collaborative work. In coordinated collaboration (e.g. in a meeting room), infrastructural requirements can be relaxed since most of the participants can be expected to bring their own devices and therefore the hardware infrastructure can scale dynamically to the amount of users involved if the system supports personal devices.

In addition to the increase of available personal devices, we also find public electronic devices in many locations. In particular, public screens (mostly used for advertisement) are all around us and – together with wireless network infrastructures which are also widespread – could also potentially be used for building ad-hoc collaboration.

All the hardware requirements needed for enabling very spontaneous collaboration are therefore fulfilled – what is still lacking is the software and the concepts which enable developers and especially end-users to make proper use of these new opportunities.

The research on computer supported collaborative work has a long tradition and a lot of important and fundamental work has been contributed by many researchers around the globe. A lot of their findings, ideas and concepts can be directly applied to the new situation which has arisen in the last few years (e.g. conflict management strategies, multi-user support, etc.). Others have to be extended or rethought such as the concept of traditional window-based user interfaces, solutions for privacy and security, handling of different input modalities, ways of data representation, etc.

## Focus of the thesis

In light of the new opportunities for spontaneous collaborative work, we have set up an interdisciplinary project to carry out research on how this type of collaboration could be supported from a technical perspective (aspects of Software Engineering), how the usability of such systems can be improved (aspects of UX/HCI) as well as what effects the extension of collaborative situations with electronic devices has on the collaborative process and users themselves (aspects of Psychology). This thesis focuses on the technical perspective and therefore has as its goal to provide a technical solution for the easy development of collaborative systems.

For our context, as well as for supporting novel types of collaboration in general, we think that the following three challenges are the most important to overcome and therefore they will be the primary focus of our solution.

*Device heterogeneity* is implied when integrating personal devices into a collaborative system to ensure that no users are excluded from participation because of the lack of support for their devices. Besides general support (the possibility to connect to and interact with the system), device heterogeneity also implies challenges in terms of adaptability to device specificities such as screen sizes, input modalities and capabilities. To provide appropriate solutions to overcome device heterogeneity, we therefore not only address the integration of as many devices as possible into our collaborative system but also provide means to adapt the software to the specificities of the different devices and/or device-types involved.

Because users might not be willing to follow complex installation and configuration procedures when they start spontaneous collaborations, *“walk-up-and-use”* capabilities are essential for collaborative systems. Since in spontaneous situations no pre-installation of specific applications on personal devices can be assumed, nor can the installation of them be enforced, true walk-up-and-use can only be established by relying on default installations and functionalities of the devices. Our solution therefore will allow the execution of collaborative applications without installation and/or configuration and therefore support the spontaneity of ad-hoc collaborations.

As a third main goal, our solution will address the *complexity of multi-user and multi-device application development*. With development guidelines and best practices, we will show how additional complexity of applications involving multiple users and devices can be diminished and how their development, debugging and testing can be simplified. By reducing the complexity we will not only be able to flatten the required learning curve for novice developers but also increase the efficiency of experienced developers and improve software quality.



Since our research group’s project is meant to be long term, future safety and longevity of the developed code as well as applicability in real-world scenarios are important aspects that need to be considered. We do not want to build a solution that is only suited for developing software for research experiments. Our solution should be able to produce applications that are stable and complex enough for real world use, since this is the most effective way to validate the usefulness of our solution – by applying it in real world cases and getting feedback from collaborators about their experiences with it in daily life situations.

In this thesis we evaluate potential technology candidates and then show that development of collaborative applications using web technologies is both appropriate and feasible, and propose a toolkit for developing collaborative applications that is based on web technologies.

We present our approach and our concepts to reduce the complexity of the development of distributed real-time applications, and show how we can profit from code and API reuse. We also provide – in addition to implementations of the fundamental components of the toolkit – proof of concept implementations of components useful for typical use cases that arise when developing collaborative systems. These implementations can be taken as examples of how further components could be realized, thus extending the functionality of the toolkit.

We do not claim that our solution is complete, but rather that the programming guidelines and development concepts it provides establish an extensible and improvable modular code base that can be used to develop collaborative applications for a wide range of scenarios. While our own work focused on colocated synchronous collaboration, our solution is easily extensible for remote and/or asynchronous collaboration as well.

## **Contributions**

In addition to the code for the toolkit itself, our main contribution lies in the concepts for overcoming issues related to heterogeneity and the differences between single- and multi-user software development. We show why we think that web technologies are the only currently available technology stack which is flexible enough to provide “walk-up-and-use” functionalities to support even very spontaneous ad-hoc collaborations and provide a proof of concept which shows that even more complex collaborative applications can be implemented with today’s web technologies, which fulfill realistic scalability requirements and therefore are applicable in real world use cases. We also present concepts related to how code can be replaced dynamically based on device specificities and/or user preferences and how our extended module concept makes it easy to add, remove and replace components and implementations for customizability of collaborative settings.

## Deliverables

In addition to this thesis, the actual code of the toolkit is available at

<http://olinux.github.com/twice/>.

This project web page contains the different, elaborated modules as well as the basic structure of the toolkit, and also includes a documentation section where the technical API specification (JavaDoc) can be found as well as further tools for bug tracking, discussion and feedback.

## Outline of the thesis

The thesis is divided into four main parts. In the “state of the art” section (cp. 2), the broad concept of computer supported collaborative work as well as the issues that such a collaborative system implies are presented by giving an overview of the related work that has been done in this field. A concluding discussion examines the elements that are still lacking to achieve truly spontaneous collaboration.

In the “specification” section (cp. 3), we present the context of our project and the requirements that guided the work during this thesis. We present the arguments which motivated us to base our toolkit on web technologies and the system architecture analysis that lead us to structure the code and the distribution of functionalities across the devices involved in the way we have.

The “toolkit” section (cp. 4) explains the actual realization and the concepts and structures that are provided by our toolkit. Additionally, we present the basic functionalities, and examples for additional toolkit components which can be seen as examples for the development of modules to extend the toolkit and to establish even more complex functionalities.

In the “evaluation” section (cp. 5), we present the different actions we have taken to evaluate and therefore validate the different functionalities and requirements that were defined in the specification phase. We present technical evaluations, as well as different user experiments that were performed with our system, and a case study application which provides the functionalities for a collaborative mind map solution that was evaluated in a real-world experiment in an educational setting in a high school class.

## Terms and definitions

To improve readability of this thesis we will use the – in our opinion well argued – terminology of Kaufman et al.: “*We often refer to things involved in a conversation by name, for instance, **Alice** and **Bob**, [...]. This is a convenient way of making things unambiguous with relatively few words, since the pronoun **she** can be used for Alice and **he** can be used*

for Bob. It also avoids [...] arguments about whether to use the politically incorrect **he**, a confusing **she**, an awkward **he/she** or **(s)he**, an ungrammatical **they**, an impersonal **it**, or an incredibly awkward rewriting to avoid the problem.“. [42, p. 5f]

Terminological uncertainties are either explicitly clarified within the text or in footnotes. Citations are surrounded by double quotes (“ ”) and written in *italic style* and code sections are either presented in a separated listing or by the special code font style to separate them from standard text.

All trademarks in this document are held by their respective owners.

# 2

## State of the art / Related work

---

<b>2.1. Computer supported collaboration</b> . . . . .	<b>7</b>
2.1.1. General challenges . . . . .	7
2.1.2. Types of collaborative systems . . . . .	8
2.1.3. Ad-hoc vs. infrastructure-based collaboration . . . . .	11
2.1.4. Devices involved . . . . .	11
2.1.5. Problem space . . . . .	13
<b>2.2. System architecture of distributed systems</b> . . . . .	<b>15</b>
2.2.1. System dynamics . . . . .	15
2.2.2. System availability . . . . .	18
2.2.3. System scalability . . . . .	19
<b>2.3. Platform- and device-heterogeneity</b> . . . . .	<b>19</b>
<b>2.4. Eventing and synchronization</b> . . . . .	<b>22</b>
<b>2.5. Security and privacy</b> . . . . .	<b>25</b>
<b>2.6. Interaction with shared devices</b> . . . . .	<b>29</b>
<b>2.7. Aspects of user experience</b> . . . . .	<b>30</b>
2.7.1. User awareness . . . . .	30
2.7.2. Distribution of user interfaces . . . . .	30
<b>2.8. Toolkits and solutions for collaborative applications</b> . . . . .	<b>32</b>
2.8.1. Multi-user / multi-device support . . . . .	33
2.8.2. Extension of legacy apps . . . . .	34
2.8.3. Communication . . . . .	34
2.8.4. Distribution of user interfaces . . . . .	35
2.8.5. General toolkits . . . . .	35
<b>2.9. Discussion</b> . . . . .	<b>36</b>

---

The development of computer supported collaborative applications is still a big challenge. To simplify the development process and to reduce the effort it takes for a developer to get in touch with this complex but interesting field of software engineering, we propose a holistic and dynamic toolkit. After presenting the general issues which are involved when implementing computer supported collaborative systems, we examine the different aspects (*synchronous vs. asynchronous* and *colocated vs. remote*) these systems could have as well as the implications that these aspects have on implementation and of which a developer has to be aware. After the definition of two additional dimensions (*ad-hoc vs. infrastructure-based collaboration* and *devices involved*), an overview of the issues found is presented. The different issues and their related work are presented before we look at existing toolkits for collaborative systems. At the end of this chapter we discuss the findings of the state of the art and we show what is currently missing in order to be able to develop any type of collaborative system in a simple and highly customizable way.

## 2.1 Computer supported collaboration

*Computer supported collaborative work (CSCW)* is the subject of long term research and has been examined by many different scientific disciplines in terms of its technical as well as social implications[22].

Most commonly, collaborative systems can either allow multiple users to interact with a single device (e.g. a standalone digital whiteboard), rely on a single user to control the system as a representative of a collaborating group (cp. “backseat driver” in [60]) or support multiple, fully autonomous and non-related devices that are used together for collaboration (cp. “divide and conquer” and “brute force” in [60]). Although rather simple to realize technically, such collaborative systems involve several drawbacks in their effectiveness due to either their lack of parallelism or because they do not allow for an awareness of the other users actions (cp. [60]). To overcome these restrictions, many advanced collaborative systems allow the integration of multiple devices and consequently build a distributed system that allows simultaneous interaction using multiple devices (virtually) executing a single collaborative application.

### 2.1.1 General challenges

When developing a distributed collaborative system, many challenges have to be overcome. In addition to well-known (mostly technical) problems from the field of distributed systems such as *aspects of system architecture* (dynamics, availability, performance), *platform- and*

*device-heterogeneity* and *eventing and synchronization* (event-ordering, replication of state, concurrency management, etc.), specific issues arise such as *security and privacy* (e.g. protection of private data, shared data of a work group, as well as public data), *interaction with shared devices* and *aspects of user experience* (e.g. distribution of user interfaces, user awareness).

Depending on the type of collaborative system and the devices involved (influenced by the available infrastructure and predefined hardware restrictions) these different issues vary in importance and their implications on the development of such applications.

### 2.1.2 Types of collaborative systems

Over many years of research, different systems of categorization have been developed for the different approaches to how collaborative work can be supported by the use of computers and other electronic devices. Common ways of categorizing computer supported collaborative systems are by the functionalities of the system [11, p. 119ff] or by its focus on the dimensions communication, coordination and cooperation in the 3C model ([11, p. 125], [43]). These approaches to categorization are very useful for grouping and comparing concrete solutions for collaborative work. Alternatively, the *CSCW Matrix*, also known as the *Time/Space Groupware Matrix* [6, p. 742] categorizes collaborative systems<sup>1</sup> by the dimensions *time* and *space* (Figure 2.1). Possible characteristics of these dimensions are “at the

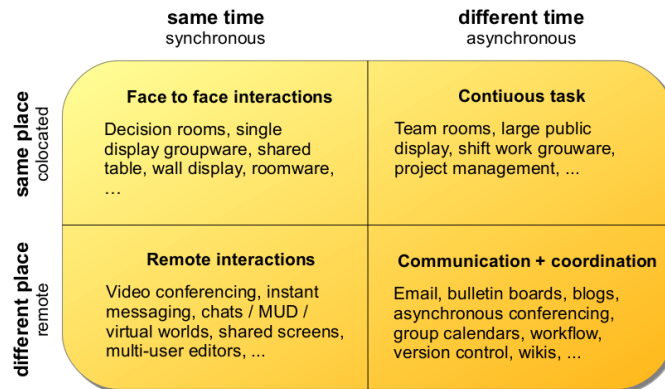


Figure 2.1.: The CSCW matrix – based on [96]

same time” (*synchronous*) or “separated by time” (*asynchronous*) and “at the same place” (*colocated*) or “at different places” (*remote*) respectively. These categories include several

<sup>1</sup>Here, the term *groupware* is used which is more restrictive than *CSCW*, since the former refers to a concrete software solution while the latter usually includes all aspects of the field of research. In this context, the two terms shall be understood as synonyms since the matrix is not restricted to software-technical aspects only. More about the terminological discussion in [11, p.92].

important differences between properties of collaborative systems which can seriously affect the technical realization of the corresponding applications and which should definitely be taken into account when building a holistic development toolkit for all types of collaborative systems.

### **Remote vs. colocated collaboration**

Spatial separation is about the location of persons within the collaborative space. If they are physically separated (in another room or on a different continent) this is called *remote* collaboration. *Colocated* collaboration on the other hand takes place if the involved persons are located in the same place and would be able to communicate without the aid of technology. Hybrid systems, where work groups (colocated) interact with distant collaborators (remote), are possible as well and are often called *Mixed Presence Groupware (MPG)*[78].

The spatial distance between the two parties in a remote collaboration is mostly overcome by public networks (e.g. the internet) and an external connection is therefore required. In contrast, in colocated collaboration, a rudimentary infrastructure can be assumed. Although the technologies applied to enable communication between devices (usually standardized network protocols) are often the same for colocated as well as for remote scenarios, the reduced available bandwidth due to the bridging of long distances through public networks often leads to increased latency for the communication between multiple remote collaboration sites. Therefore, flexible collaborative applications have to make sure that they adapt the amount of their communication to the given settings (e.g. by reducing their update intervals).

In terms of usability, requirements for collaborative systems differ depending on whether the collaboration is colocated or remote. Although the time between an executed action and its visual representation influences the usability of a system substantially in general (cp. [9], [17], [29]), this effect is even more important in colocated scenarios since the awareness of the delay between an action and the system's reaction can be increased by the user through observation of the physical space. Additionally, users have different expectations for connections which are over long distances and those which are very short. They tend to be much more tolerant of latency and network failures when bridging long physical distances than they are if two devices communicate with each other in the same room. For collaborative systems, this means that the latency of system updates can be relaxed much more for remote systems than for their colocated counterparts.

### Synchronous vs. asynchronous collaboration

Collaborative systems can – aside from their spatial dimension – be distinguished by their temporal characteristics. While synchronous collaboration happens at a specific time and multiple users interact with the system simultaneously and react to the actions of other users, asynchronous collaboration takes place in temporally separated working sessions. One example of synchronous collaboration is a brain-storming session where users let the inputs of other participants influence their own ideas. An example of asynchronous collaboration is the sequential, collaborative elaboration of a document which is edited by one user at a time. More examples of such asynchronous collaboration systems can be found in [89].

Similarly to the spatial distinction, the temporal characteristics also have substantial effects on the design of a collaborative system. In particular, the conflict management differs depending on when the users interact with the system. While conflicts in an asynchronous collaborative system are rather rare and can be prevented by simple locking mechanisms (cp. [11, p. 191ff]), the extensive use of such strategies would contradict the idea of synchronous collaboration since in practice only one user would be able to interact at a time. Synchronous collaboration therefore requires more complex conflict management strategies (cp. 2.4) which are dependent on the specific collaborative task (e.g. moving elements, editing text, pointing and highlighting).

Since this conflict handling has to happen in real time and multiple devices are interacting with the overall system – and are therefore confronted with a distributed system (cp. [45, p. 1ff]) – the performance is very important. A slow or very conservative conflict management can strongly affect the responsiveness of the system in a negative sense. That is why the optimal choice of the conflict management strategy plays an important role (cp. [27, p. 207]).

Asynchronous collaboration works only if the state of a system can be stored and is ready for editing in the future (cp. [14, p. 107]). In contrast, synchronous collaboration basically exists only at the moment of the interaction. Therefore, results and states of such systems do not necessarily have to be persisted in the long term (although it could make sense to store results of synchronous collaborative work). This means, that synchronous systems can (theoretically) exist without a long term persistence layer, which reduces the complexity particularly in the context of distributed systems with many devices and missing central instances (cp. 2.2).

In addition to the mandatory persistence layer, asynchronous collaboration requires a minimal infrastructure since the medium for the persisted data has to be accessible by the different users. Usually, this is achieved using a database on a central server, but it would be possible to use mobile storage medias (e.g. flash memory sticks) which can be exchanged between the different users as well.



### 2.1.3 Ad-hoc vs. infrastructure-based collaboration

Besides collaboration in a setting supported by infrastructure (e.g. an existing WiFi, an internet connection, a shared screen or other preinstalled devices) – as is possible in both colocated and remote scenarios – a colocated collaboration can (but does not have to) arise spontaneously (or “ad-hoc”) when people meet: “*When two individuals come in close physical proximity or meet face-to-face their respective spheres overlap enabling their personal mobile devices to interact. At that point, the devices can exchange information and access each other’s services*” [44, p. 77]. In contrast, remote collaboration is usually less spontaneous (cp. Kraut et al. in [33, p.137ff]). Although also possible for asynchronous scenarios, ad-hoc collaboration usually arises in combination with synchronous collaboration. Since no infrastructure can be expected in such situations, a system supporting the spontaneous appearance of collaboration has to be able to run on coincidentally available devices – usually brought by the users – without predefined network connections and system structures.

### 2.1.4 Devices involved

There are multiple factors that can influence the involved devices in a computer supported collaborative system. In addition to the spontaneous appearance of a collaborative environment and the possible lack of any infrastructural support (cp. 2.1.3), the definition of required devices for the execution of a specific application, as well as the restriction of devices able to interact with the system, play an important role in the technical realization – especially in terms of maintaining heterogeneity – of a collaborative environment.

Since the representation of information (especially in synchronous, colocated collaboration) is a challenge, a solution has to be found which can provide shared information to all users at the same time. Most systems define a screen which is visible to all participants and therefore becomes a *shared screen* (e.g. [83], [88], [26]). Depending on whether the shared screen is the only display or if there are others (e.g. private devices with visual output capabilities) integrated into the overall system, such systems are called *Single Display Groupware (SDG)* or *Multi Display Groupware (MDG)* (cp. [84]) respectively. Since the information that is represented is usually quite numerous and should be visible from different distances, this shared screen usually has a specific minimal size. Although the existence of such a big screen is given in many collaborative situations (in particular in meeting rooms), a generic collaborative system should be able to support such a setting but not take such an infrastructure for granted since colocated collaboration can occur spontaneously at non-predefined locations (cp. *colocated collaboration* in 2.1.2). Alternatives to a big shared screen are for example screens cloned on different devices so that every user can see a copy of the shared area (“*What You See Is What I See*” – *WYSIWIS* – cp. 2.8.4), the dynamic definition of shared resources

(e.g. a notebook screen is defined to be a shared screen for small work groups) or the combination of multiple small screens (e.g. multiple tablets) into a bigger shared screen (cp. [52]).

Additionally, existing collaborative systems usually rely on predefined, non-personal input devices (e.g. keyboards, touch-screens, pointing devices), which are taken for granted as well [88]. Although this allows for configuration- and installation-free “walk-up and use” of the system, it restricts spontaneous collaboration and - without the possibility of integration of additional devices - reduces the number of potential users. Additionally, predefined devices might be less suited for fulfilling a specific task than available devices (e.g. because they have been brought into the environment by the users), which would result in sub-optimal use of available resources and reduce the efficiency of collaboration support.

To reduce the infrastructural requirements for a collaborative setting, personal devices of users can be integrated into the system. Today, it is common for users to carry smart phones, tablets, e-readers, notebooks and other electronic devices with them and for those devices to be brought along to a collaborative session: “*Mobile devices like cell phones, PDAs and wearable computers have become our constant companions that are available wherever we go.*” [44, p. 76]. If personal devices can be integrated into a collaborative systems, then theoretically this not only allows as many users to participate as there are devices available, but also implies that the users do not need to get used to an unfamiliar device. They can profit from personal settings (e.g. keyboard layout, personal touch calibration on smart phones and tablets, personalized dictionaries for text entry, etc.) as well as from the familiar interface and interaction modalities of their device(s). The user takes advantage of so called habituation: “*When one uses an interface repeatedly, some frequent physical actions become reflexive [...] The user no longer needs to think consciously about these actions. They’ve become habitual.*” [79, p. 15].

Additionally, the ownership between a user and a device can be seen as a way to implicitly distinguish between private and public spaces. Since many private devices have their own displays, private information can be presented on the users own device, while public information can be shown on non-personal devices – such as those which are part of the infrastructure (if available) or which are explicitly defined for public access (cp. [20]). Other advantages can be found in reduced cost of infrastructure (none or only a few devices have to be provided and maintained) as well as in protection against vandalism since infrastructural resources (if there are any) can be protected by structural barriers (e.g. positioning of the shared screen behind glass or mounted on a wall at a non-reachable height).

Since neither the different types of available devices nor the software installed on them is under control on personal devices, a “walk up and use” functionality that allows the integration of such devices is quite difficult. Therefore, existing systems which support the use of

personal devices often require the installation of specific software in advance (e.g. [83]) and consequently complicate the occurrence of spontaneous collaborative situations. This also reduces the set of supported devices because the specialized software is usually not available for many different platforms and is therefore not well-suited for a heterogeneous environment. Additionally, users might not be willing to accept such an installation on their own devices, since they might not be convinced about the trustworthiness of the software and therefore decide not to participate in the collaboration: “[...] careful users concerned about their privacy have to make a tradeoff between the functionality offered by an app and its potential for compromising their privacy.” [15, p. 315]. To allow the “walk up and use” functionality of a system including non-controlled private devices a technology has to be chosen that neither requires installations nor configurations and which runs on as many platforms as possible.

### 2.1.5 Problem space

Although there are many general problems that have to be solved by any type of collaborative system, the importance, as well as the technical impact, of the different issues varies depending on the characteristics of the system. We have shown that not only the categories of time and space influence the needs of such an environment but also the fact that the devices involved may appear spontaneously. The main issues that computer supported collaborative systems face, and their impact depending on the characteristics of a system are summarized in Figure 2.2. The dimensions along which the issues are evaluated – in addition to the standard dimensions of space and time – are the distinction between situations where the devices involved have been predefined and those where a dynamic set of devices (usually private devices of users) are supported, as well as if the collaborative situation arises spontaneously (“ad-hoc”) in a non-defined environment or if it takes place in a predefined location where infrastructural support is guaranteed.

The **system architecture** defines the ability of the overall collaborative system to address changes in the topology of appearing and disappearing devices as well as the influence of such mobility on the availability of the system and its scaling capabilities depending on the amount of devices involved. While systems with infrastructure support are less affected, colocated ad-hoc situations in particular can suffer from these issues.

Platform- and device-**heterogeneity** are hard to achieve if the system needs to support a dynamic set of devices and if it is not possible to predefine the different device-types involved.

**Eventing and synchronization** issues arise when working in synchronous environments. This is especially true for ad-hoc situations since no central controlling instances (or similar mechanisms) can be used and therefore more complex synchronization strategies have to be applied.

	<b>Coloc. &amp; synch.</b>				<b>Coloc. &amp; asynch.</b>				<b>Remote &amp; synch.</b>				<b>Remote &amp; asynch.</b>			
	<i>Predef. dev.</i>		<i>Dyn. dev.</i>		<i>Predef. dev.</i>		<i>Dyn. dev.</i>		<i>Predef. dev.</i>		<i>Dyn. dev.</i>		<i>Predef. dev.</i>		<i>Dyn. dev.</i>	
	<i>Ad-hoc</i>	<i>Infr.</i>	<i>Ad-hoc</i>	<i>Infr.</i>	<i>Ad-hoc</i>	<i>Infr.</i>	<i>Ad-hoc</i>	<i>Infr.</i>	<i>Ad-hoc</i>	<i>Infr.</i>	<i>Ad-hoc</i>	<i>Infr.</i>	<i>Ad-hoc</i>	<i>Infr.</i>	<i>Ad-hoc</i>	<i>Infr.</i>
<b>System architecture</b>																
System dynamics																
System availability																
System performance																
<b>Heterogeneity</b>																
<b>Eventing and Synchronization</b>																
<b>Security and privacy</b>																
<b>Interaction with shared devices</b>																
<b>Aspects of user experience</b>																
Group awareness of users																
Distribution of user interfaces																

	No special issues
	Attention, there are some traps
	Be careful! This involves big complexity

Figure 2.2.: The main issues of computer supported collaborative systems and their impact depending on the characteristics of a system

Although important in any condition, **security and privacy** is even more important for environments with dynamic sets of devices, since private devices can contain sensitive data that should not be published or accessed during collaborative work. Therefore, users will be more careful about what they do on private devices than on other devices and have an increased need for security and privacy.

**Interaction with shared devices** is mainly an issue of synchronous collaboration (although it might appear in asynchronous situations as well). While the interaction can be coordinated quite well, if devices are predefined (e.g. remote controllers for a shared screen) the realization can be simplified, whereas the support of dynamic devices is more complicated since these devices are not prepared to integrate smoothly.

For aspects of user interfaces, **group awareness of users** implies issues mainly for synchronous situations since it is much more difficult to increase awareness of parallel executed actions than it is for sequential ones. Another difference exists between the environments in terms of location since actions from remote collaborators might have to be extended with additional information to compensate for a lack of the types of information that users can get in a colocated context (e.g. by asking other users or observing them to match the actions in the system with the user who performed them).

To support the presentation of different information on different devices as well as to improve control over the system for users, **distributed user interfaces** would be necessary. The distribution is more complicated for a set of dynamic devices since it is unknown how many devices are involved and no initial distribution strategy can be defined.

It is worth noting that in particular dynamic device support in combination with spontaneous, synchronous collaboration (independent from its spatial characteristics) implies the biggest challenges and the most complex development issues.

## 2.2 System architecture of distributed systems

*“A distributed system is the one that prevents you from working because of the failure of a machine that you had never heard of.” (L. Lamport – cp. [81, p. 4])*

System architectures for the integration of a dynamic number of devices – as is the case for most computer supported collaborative systems – can be separated by the classical architecture categories *client-server* and *peer2peer* (cp. [75, p. 36ff]). Intermediate forms of architectures also exist (e.g. hybrid peer2peer) and the different grades of structuring of these systems have effects on *system dynamics*, *system availability* as well as on *system scalability*.

This section points out the advantages and disadvantages of the categories of architecture and presents their suitability for collaborative systems.

### 2.2.1 System dynamics

*“Interactive workspaces will be dynamic. On short time scales, individual devices may be turned off, wireless devices will enter and exit the room, and pieces of equipment may break down for periods of minutes, hours or days. On longer time scales workspaces will incrementally evolve rather than being coherently designed and instantiated once and for all. [...] It is not realistic to expect a full-time system administrator to keep a workspace running, and at the same time users must be allowed to integrate even failure prone devices.” [39]*

The dynamics of a system is particularly important if the constellation of devices within a collaborative session changes frequently. Such changes can, among other reasons, be caused by the arrival or the departure of a user, by device properties such as the sleep mode to save energy or by technical restrictions such as network problems.

#### Client-Server

Classical client-server architectures (cp. [77, p. 36ff]) are dependent on the availability of a server, which results in the need for continuity of the environment or at least of this single device instance. If a client device leaves the setting, this does not have any (general) effect on the functionality of the overall system since such a device does not have an essential function in the program flow. But, if a server device becomes unavailable, the overall system will fail completely and the server therefore becomes the “single point of failure”.

Within such static architectures, the number of clients is limited since the resources of the server are restricted and cannot be easily extended. Although concepts exist to reduce this disadvantage like load balancing mechanisms (cp. [12, p.13ff], [45, p.29ff]), they are linked with additional infrastructural efforts.

Since client-server architectures are rather easy to build and maintain and the overhead of the device management is very low, client-server structures are well-suited for small work groups. They work best if they are applied to scenarios with infrastructural support (e.g. with a dedicated server machine in a meeting room or somewhere in a stable network).

### Peer2Peer

Peer2peer solutions do not involve central servers (except “centralized”, or “mediated” peer2peer networks with look-up servers [75, p. 37ff], [5]) but let the devices interact directly with each other instead. An always available device with server functionality is thus not required, since all the devices provide this functionality together. This makes peer2peer well suited for environments which are very dynamic and which contain no or only a very restricted stable infrastructure: *“Since communication end-points can move frequently and independently of one another, mobile peer-to-peer systems are highly dynamic.”* [44, p. 81] The failing of one or multiple devices therefore does not affect the functionality of the overall system, since their job is taken over redundantly by the remaining devices. Since in non-centralized systems – such as the peer2peer architecture – no single instance keeps and decides on the current program state, but this is rather a common functionality of multiple devices, replication of the application state as well as constant synchronization is needed. This is especially essential for collaborative applications, since such systems very often contain shared objects or data structures and accordingly, the devices have to notify each other about updates regularly or even in real-time: *“[...] copies of a shared object can be updated independently and thus might become inconsistent over time. A synchronization mechanism must be employed that either prevents or reconciles inconsistencies.”* [44, p. 83] Consequently, the redundancy and increased dynamics of such redundant systems (usually) comes with the cost of higher network load due to the extended communication need.

### Hybrid Peer2Peer

Some intermediate architectures exist, such as *hybrid peer2peer*, which are defined as second generation peer2peer networks by Eberspächer and Schollmeier in [75, p. 51]. The hybrid approach integrates the additional concept of a hierarchy of the devices involved and separates them into *ultrapeers*<sup>2</sup> and *leafnodes* (cp. [5], [75]). Here, the devices which are defined to

<sup>2</sup>Here, the terminology is not always clear – often this type of device is also called a “superpeer”

be ultrapeers are used as servers for a specific group of leafnodes but the ultrapeers still communicate between each other in a pure peer2peer manner. If an ultrapeer fails, the leafnodes it serves will be taken over by another available ultrapeer (cp. [75, p. 49]).

The advantages are that the necessary communication for the synchronization can be reduced and that – in contrast to the classical client-server structure – a considerable improvement of dynamics can be achieved since the system can react more flexible to the arrival or departure of devices because the system is able to run as long as at least one ultrapeer is available. In a hybrid peer2peer architecture the level of dynamics can be controlled by the number of available ultrapeers. If there is only one ultrapeer, this corresponds to a client-server structure, if all of the available devices are ultrapeers, it can be seen as a pure peer2peer architecture.

### Cloud computing

An additional possibility to integrate resources and therefore to increase the dynamics of static client-server structures is cloud computing. Here, static structures can be made more dynamic by the addition of external resources on demand, depending on the requirements (e.g. when additional capacities are needed as more clients arrive): *“One of the key features of cloud computing is that computing resources can be obtained and released on the fly. Compared to the traditional model that provisions resources according to peak demand, dynamic resource provisioning allows service providers to acquire resources based on the current demand”* [92]. One disadvantage lies in the fact that the “cloud” is usually located outside of the local networks and therefore can only be accessed through public networks (like the internet). Therefore it might be possible that no performance gain is achieved even though additional resources were made available since the external network connection represents a new bottleneck.

### Architecture of collaborative systems

Depending on the application, collaborative systems are focused on different system architectures. Asynchronous collaboration systems usually rely on client-server architectures, since they have to ensure long durations of availability of the shared information and therefore need constant reachability as well as a unique, always available entry point (e.g. [7])

Synchronous collaboration on the other hand uses different approaches which depend on the concrete environment in which the collaborative system is integrated. If the environment is well defined and controllable (e.g. teleconferencing or meeting rooms), static structures are the usual choice. If the collaboration takes place in non-defined environments (e.g. in the

context of disaster management) more dynamic architectures are chosen (cp. [49]).

Only a few existing collaborative systems adapt to their environment and provide functionalities which allow ad-hoc use with dynamic structures and profit from infrastructural settings as soon as they are available. One reason for this might be that most collaborative systems have been developed for a specific task or setting with a predefined environment and therefore the architecture of the system is given from the beginning (cp. 2.9).

### 2.2.2 System availability

The system's availability depends directly on the chosen system architecture. In a rather static environment such as a client-server architecture (but as well as a hybrid peer2peer architecture with only a few ultrapeers), the availability can be kept high if it is ensured that the device that provides the server functionality runs with a low failure rate and has a stable network connection – otherwise it would represent a *single point of failure* (cp. [50, p. 14ff]). With this architecture, if the environment is controllable, its availability and load capacities can more easily be predicted. “*Since star architectures<sup>3</sup> have only one hop between a network node and the central hub, they tend to be more predictable and reliable;*” [85, p. 338]. Thus, predictions about performance as well as stability can be made and – if necessary – actions can be taken to improve the situation by adding more static resources combined with load balancing and/or redundancy (cp. [12]). If the architecture is situated in a non-controllable environment – e.g. if a randomly chosen device within an ad-hoc session is defined to be the server instance – the failure rate will be coupled with this server device and cannot easily be predicted. Static systems are therefore well suited for controlled environments which have infrastructures in which server components can be integrated and managed.

With redundant architectures – such as the peer2peer approach (cp. *Peer2peer* in 2.2.1) – availability is independent of infrastructural actions. Since the overall system is functional as long as devices are available, very good stability can be achieved. Nevertheless, costs related to communication between devices can lead to reductions of performance, since the network can become a bottleneck. The dynamics of a peer2peer architecture is therefore best suited for ad-hoc scenarios if there is no or only very little infrastructure. Because of the increased communication needed to maintain consistency between devices this architecture is often restricted by the network in collaborative environments.

Hybrid systems are well suited if some devices have a constant availability and therefore can be defined to be an ultrapeer. They can reduce the cost of communication while still keeping the environment independent of any infrastructure (cp. *Hybrid Peer2Peer* in 2.2.1).

---

<sup>3</sup>A standard server-client architecture is a typical star architecture



The dynamic integration of server instances through the cloud can have positive effects on the stability of the overall system as well since they can be used to provide redundancy (cp. *Cloud computing* in 2.2.1). But usually, an infrastructure has to exist to create a connection to the cloud and this connection can itself become a bottleneck which cancels out the advantages of stability due to the cost of the performance drop.

### 2.2.3 System scalability

The scalability of a system defines how many devices can be added without noticeably affecting its stability and performance. The extent of system performance is more or less predefined for static systems since a restricted resource can be distributed based on the number of participants. In addition to the performance of the server(s) itself, network capacity plays an important role – in particular because all information comes together at a single point and consequently the data throughput of a server differs essentially from that of a client (*asymmetric functionality*, cp. [75, p. 11]). Scalability of the overall system is therefore dependent on the performance of the server, the number of devices involved as well as of the available network capacities (cp. [50, p. 55]). Again, the use of external resources on demand (cp. *Cloud computing* in 2.2.1) can affect the scaling behavior of a system in a positive way, since the cloud is able to react in a dynamic way to spontaneously appearing load peaks.

A more redundant architecture allows an improved load balance since all devices contain the same information. A central instance, which collects all information and therefore handles the main part of the communication load, does not exist. Therefore, the limits of more redundant systems in terms of scalability is not clearly defined. Although the main bottleneck of static systems is eliminated, other rare resources (e.g. network bandwidth) can fall into place and the extra effort of communication for synchronizing the devices can cancel out the advantage of the improved load balancing (cp. *Peer2Peer* in 2.2.1).

## 2.3 Platform- and device-heterogeneity

Heterogeneity is an issue for collaborative systems which want to relax the restriction of supported devices to a maximum. This not only addresses heterogeneity in the sense of different supported platforms but also in the sense of many different types of devices with their own input modalities and special functionalities.

### Model Driven Architecture

One approach to managing heterogeneity is to define functionalities as well as user interfaces using a *Model Driven Architecture (MDA)* at an abstract level (e.g. [59]) and to generate the concrete platform- and device-specific code based on that model. Problems appear when device specificities need to be considered and the functionality should be customized for specific devices. Those special cases either have to be handled as part of the model itself (e.g. [66, p.110ff], [21]) or by manipulation of the generated code in a further development step.

Model Driven Architecture is therefore well-suited for very standardized functionalities which will be represented in an identical or at least very similar way on all devices. In contrast, it is usually less suited for customizable and device-specific implementations.

### Platform-independent languages

Platform independent languages, with their best-known representative Java [106], promise the execution of the code written once on many different platforms without adaptation. This is achieved most of the time by compilation into a platform-independent byte code which is then processed by a device specific interpreter which translates the byte code into actual machine code (cp. [4]). The often cited performance loss which results from interpretation at run time is (at least for Java) more or less under control these days (cp. [108]) – nevertheless, for the execution of the code, a specific interpreter is still necessary and unfortunately this interpreter does not (yet) exist for all modern platforms and in particular for low-performance devices (e.g. phones, e-readers, etc.) or sometimes only provides sub sets of functionalities (e.g. [107]). Since the goal of such platform independent languages is to execute exactly the same code on every target device, it can be difficult to customize an implementation based on the type of platform capabilities of a specific device.

This type of technology is well-suited for cases when similar appearance of functionalities on many different devices is desired and less suited for device specific implementations.

### Cross-Compiler / Source to source translators

Cross-Compilers allow – similarly to “platform-independent languages” – writing code in one single programming language. Instead of the translation of the code in a platform-independent format, here machine code is generated in direct for multiple, different specific platforms (e.g. [98]). Instead of generating binary code, source to source translators translate source code from one programming language to another. With some cross compilers as well as several source to source translators, it is possible to configure alternative implementations

for some software components which can be replaced when customizing a functionality on different devices (e.g. [102]).

The option to define alternative implementations for different types of devices makes these types of technologies especially interesting for adaptation to device specificities. On the other hand, it needs a lot of preprocessing and is therefore often hard to manage and to test.

### Dynamic languages

Dynamic languages are (usually) interpreted script languages. Examples are Perl [114], Python [116], Ruby [118], Tcl/Tk [122] as well as ECMAScript [97] – better known by its most famous dialect JavaScript. Whether the script language is supported by a specific platform depends on the availability of the specific interpreter. In that context, JavaScript plays a special role, since its interpreter is broadly available in the form of JavaScript enabled web browsers: *“Web client programs (browsers) are available for all popular computing platforms and operating systems, providing access to information in a platform independent manner”* [8].

Every device that allows access to the internet and provides a (rather) modern internet browser is capable of executing JavaScript. Although the functionalities of the language are usually restricted by a so-called “sandbox” [25, p. 266ff] in which the code lives (in standard web browsers it is for example not possible to directly access the file system or other resources of the device). JavaScript is – because of its wide-spread support – a very important language. Also, because of its broad platform support and its functionalities in the areas of communication and widgets, Tcl/Tk is used rather often for the implementation of collaborative systems (e.g. [65], [48], [28], [54]).

Dynamic languages are capable of reacting very well to their specific environment (e.g. the executing device) at runtime. Because code can be added dynamically, the actual execution and combination of program logic does not have to be fixed at compile-time but can rather be decided on (and even controlled) at the execution phase. Although usually less performant and sometimes even restricted in functionality, dynamic languages are very well suited for the customization of logic for a specific execution scenario.

### Overcoming the heterogeneity with web technologies

As mentioned above, one of the most widely distributed technologies that exists is JavaScript because its interpreter is built into almost any current web browser. Combined with HTML and CSS, which are used for visualization, it makes up a technology set which we call “web technologies”. To address device- and platform heterogeneity in collaborative applications,

several toolkits have focused on implementations based on web technologies, particularly because these technologies seem to have been developed from the beginning under the aspect of collaboration: “*The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems*” [104]. While many projects like the *Basic Support for Cooperative Work (BSCW)* [7] are restricted to asynchronous collaboration by the provision of a shared workspace for the exchange of data and files only, some newer projects such as *Powermeeting* [86] try to make use of the newest features of modern web technology for synchronous systems. The extensions which have been included into the web technology standards (HTML5, CSS3, more performant JavaScript engines) are especially promising for their use in real-time systems and therefore for collaborative applications as well: “*This study has shown that with the advancement of Web 2.0 technology, browser-based real-time groupware can now offer a level of functionality, interactivity, and graphical user interface more akin to their traditional, desktop-based counterparts. Users seem to feel quite comfortable with their use. Such browser-based groupware may have provided an approach to address the adoption hurdle facing groupware for a long time.*” [58]

### Comparative discussion

We have seen different approaches to overcome the issues of heterogeneity when trying to integrate a big set of different devices. Some technologies (Model Driven Architecture and Platform-independent languages) are well suited for very standardized applications but less so for the customization of functionalities to device specificities. Others offer alternatives for different devices which are defined either at compile time (Cross-Compiler / Source to Source translators) or at runtime (Dynamic languages). Depending on the application to realize and the types of devices which will be supported, some technologies are more suitable than others – in general the choice of technology is a tradeoff between customizability and generalizability.

## 2.4 Eventing and synchronization

A well-known problem coming from the domain of distributed systems is the eventing and the synchronization of the application state over multiple devices. Since in a collaborative environment multiple devices interact with the same resources, Sun et al. developed a data consistency model for cooperative editing systems [76]. They describe a collaborative system to be consistent if the following requirements are fulfilled:

- **Convergence:** *when the same set of operations have been executed at all sites, all copies of the shared document are identical.*

- **Causality-preservation:** for any pair of operations  $O_a$  and  $O_b$ , if  $O_a \rightarrow O_b$ , then  $O_a$  is executed before  $O_b$  at all sites.
- **Intention-preservation:** for any operation  $O$ , the effects of executing  $O$  at all sites are the same as the intention of  $O$ , and the effect of executing  $O$  does not change the effects of independent operations.

A consistent system can therefore be achieved if dependent operations are executed in the same order on all devices involved.

This coordinated execution order can be achieved in different ways. Possible solutions can be found under the term *Concurrency Control* and are grouped into *optimistic* and *pessimistic* approaches (cp. [11, p. 187ff]).

### Pessimistic concurrency control

Pessimistic concurrency control systems can be further divided into *centralized* and *decentralized* solutions. A centralized control either contains one single instance that decides on the program flow and coordinates the other involved entities (*control unit*) [76], or a *token-passing* approach is established where a *token* is passed from device to device and only the device currently holding the token is allowed to contribute to the system (e.g. execute operations, manipulate shared data) [11, p. 190f].

Decentralized pessimistic control mechanisms offer different possibilities for establishing synchrony and therefore consistency between devices. In simple *locking* [11, p. 191ff], the access to a specific part of a system is locked until the initiator frees it or another condition is fulfilled (often a timeout). When using *floor-passing* [11, p. 194ff], the different parts of the applications are assigned to a specific device. Only the device holding “ownership” of a partial functionality is allowed to interact with it – if another device needs to access such an area, it can request the current owner for transfer of “ownership”. Transactions [11, p. 197ff] allow to define specific execution flows including involved resources. If a transaction is executed, the resources are locked and are released automatically as soon as the execution of the transaction has finished. One of the most often used pessimistic control mechanism is *Operational Transformation* (cp. [71], [70]). Here, conflicting operations are manipulated so that their effects result in a consistent state even if they are not executed in the same order. A typical application for that control mechanism is a collaborative text editor (cp. [76]).

Most types of pessimistic concurrency control strategies restrict interaction with a shared object to a single device and therefore serialize the appearance of events. Although this way of handling issues of conflicting events is straight-forward and has some very useful applications, pessimistic concurrency control often implies higher latency of a specific action

(because the device has to wait for the permission to participate) compared to optimistic concurrency control. One of the main advantages of pessimistic concurrency control is that the consistency can be ensured without the need of rollbacks or similar.

### Optimistic concurrency control

Optimistic concurrency control allows the simultaneous interaction of all devices with the overall system. Therefore, no structural order of the different events can be established. For the implicit decision on the common valid order of events (and therefore execution of functions), approaches have been developed as part of the event ordering in distributed systems.

The ordering of events in distributed systems has been addressed intensively for many years. For example Lamport published fundamental work for this area already in 1978 by introducing logical clocks [47]. Many follow-ups such as vector clocks [24] or interval tree clocks [1] were based mostly on the work of so called Lamport-Clocks and have extended the relatively lightweight concept for the purpose of optimized ordering mechanisms.

Other approaches also exist like the Network Time Protocol (NTP) [57] which synchronizes system time through multiple time servers and often achieves a precise enough clock synchronization between the devices.

Since the correct sorting of events by their order of appearance can only be applied if the received events are complete and no delayed events (e.g. by slow communication channels) can arrive at a later point in time, the responsiveness of a system which tries to execute these events and their underlying functionality conflict-free is rather poor and therefore has a very bad influence on usability (cp. [27, p. 211ff]). To reduce the effect of this poor responsiveness, different approaches exist which allow to execute the program flow even before sure that no conflicting (e.g. delayed) events arrive and to solve conflicts by the application of well-suited strategies.

These strategies mainly focus on how disordered operations can be rolled back and how the global order can be re-established (cp. [90, 82]). Other interesting strategies (like dead-reckoning) originate from the research field of group- and online-gaming (e.g. [41], [16], also cp. [18, p.266f])

Greenberg points out that *“The choice of a concurrency control method can be difficult. A wrong choice can lead to an unusable system. Selecting an overly powerful approach could be overkill for the application, and much development time could be expended for schemes that are unnecessary or used only rarely.”* [27].

Complexity is actually very high and to make things worse, not all strategies can applied to all types of events. There are events which become dispensable as soon as a newer event

containing the same type of information has arrived (e.g. position of a mouse pointer). A delayed event of that type can simply be ignored and an expensive conflict handling mechanism is not needed [62, p. 160]. But, there are irreversible events as well. This is the case if the execution of an event influences a non-controlled system (e.g. a credit card transaction has been executed on a third party system). Since those events cannot be undone, it has to be ensured before execution that no conflicts can arise, which usually implies the application of a pessimistic approach. Consequently, it is not sufficient to choose one single strategy for a specific collaborative system, but multiple strategies have to be chosen to address the different needs of applications: *"we believe that concurrency control needs are highly application dependent and that no one mechanism would suffice."*[65]

Optimistic concurrency control usually has a better responsiveness than pessimistic strategies since events can be handled immediately and are only rolled back in the case of conflicts and if required. But, if conflicts arise too often, the gain of responsiveness is reduced by the rollback mechanism which can have a negative influence on the user experience. The effectiveness of optimistic concurrency control systems depends therefore on the number of conflicts that actually appear[65].

### Concurrency control for collaborative applications

Collaborative applications are typically executed as distributed systems and therefore are affected by the issues mentioned above. Especially because of the need for fast responsiveness of such systems, concurrency control has to be chosen very carefully and ways have to be found to treat the different types of events. Although the decision of which concurrency control strategy is the most suitable has to be made based on the actual application, the manifold types of the different approaches should be encapsulated and simplified by a holistic toolkit for collaborative applications.

## 2.5 Security and privacy

Users of collaborative systems have different security and privacy needs. When Alice is working with her private device (cp. 2.1.4), she needs to be sure that a collaborative application only accesses data which will be used for the collaborative task but not her private resources (e.g. private pictures). So, she either has to trust the collaborative application or has to restrict the capabilities of the application by, for example, not giving the application access to the file system. Another possibility is to run the application in a *sandbox* (cp. *Dynamic languages* in 2.3) that restricts the functionality of the program and might ask for explicit permission to execute actions which go beyond standard functionalities. But, privacy also

means that Alice is capable of protecting her data from being seen by Bob (e.g. by looking at her device in a colocated collaborative setting).

In addition to issues of privacy, collaborative systems have to ensure technical security as well: private data, protected data (e.g. data which should only be seen by a specific work group) as well as public data have to be protected. Access to the collaborative setting has to be restricted and it has to be possible to investigate who was executing which actions if some irregularities appear.

Schäfer divides the topic into subtopics based on the usual *confidentiality*, *integrity* and *availability* (CIA) terminology, but further separates security into *confidentiality*, *data integrity*, *accountability*, *availability* and *controlled access* [69, p. 7f].

## Confidentiality

*“Transmitted or stored data should only be disclosed to authorised entities”* [69, p. 7]

Confidentiality means protecting data from unauthorized disclosure. This topic is strongly linked to the area of cryptography, which itself can be categorized into symmetric and asymmetric approaches (cp. [42, p. 47ff and p. 50ff]).

Symmetric cryptography ([69, p.31ff]) bases itself on only one single key, which is used for both decryption and encryption. Although this approach performs quite well, it implies the issue of transporting the key through non-protected networks (cp. [69, p. 111ff]). Asymmetric cryptography on the other hand uses two keys – so called *private-public key pairs*. The idea is to have one secret key for every user (private key) as well as one public key which has a mathematical dependency on that private key, and which is publicly available. Although theoretically feasible, it is practically impossible to recalculate the private key based only on knowledge of the public key. [69, p. 53ff]

If somebody wants a specific user to be able to read a message, the message can be encrypted with the public key of the recipient and can only be decrypted by application of the corresponding private key. On the other hand, a message encrypted by a private key can only be decrypted by the public key, which allows the creation of digital signatures. Anybody can check the authenticity of a sender through successful decryption of a signature with the corresponding public key and therefore is able to ensure that the message originated from this specific user (cp. [93, p. 10]).

Although asymmetric cryptography – compared to symmetric – results in increased calculation efforts, it has the advantage that no secret key has to be exchanged between users. This is why it is common that symmetric keys are protected by asymmetric encryption for their transmission through non-secure networks. This ensures that no non-authorized party



can access the shared key, while still profiting from the better performance of symmetric encryption mechanisms (cp. *Diffie-Hellman key exchange* [73, p. 190]).

This is particularly important in the context of collaborative systems because usually a lot of data is exchanged between devices and users. A shared key could be made available to all members of a work group, ensuring that nobody else would have access to the corresponding information while achieving better performance than with asymmetric encryption.

### Data integrity

*“It should be possible to detect unintentional or deliberate changes to data. This requires that the identification of the originator of the data is unique and cannot be manipulated.”*

[69, p. 7]

To make sure that messages have not been manipulated on their way from a sender to a recipient the concept of check-values exists (cp. [69, p. 83ff]). To ensure the integrity of data, Alice (the sender) creates a check-value by applying a hash function, encrypts the result using her private key and adds it to her message for Bob. Bob (the recipient) decrypts the check-value using Alice’s public key, calculates the check-value by applying the same hash function to the received message and checks if both check-values are equal [93, p. 11]. If the message had been manipulated between sending and reception (e.g. by manipulation, network failures, or similar), the two values would not match and Bob would be informed that there are issues with that message and that it should not be trusted. Depending on the application, the system could then either display an error message or simply ignore the message.

By applying data integrity, collaborative systems can therefore ensure that the actions and contributions are not manipulated on their way through the network and therefore create the foundations for accountability.

### Accountability

*“It must be possible to identify the entity responsible for a particular event (e.g. use of a service)”* [69, p. 7]

While data integrity makes sure that data is not manipulated on its way through the network, accountability ensures the traceability of messages and their actions. This can be achieved by consistent validation of digital signatures (cp. *Data integrity*) and the logging of sender names. The information can be visualized (e.g. by annotating executed actions or manipulated application states with user assigned colors) which enables other users to keep track of who has been causing which action and thus gives them a mean of control (e.g. they could

intervene directly by talking to the user or by correcting malicious actions). Accountability therefore is not only a technical requirement, but also is part of the establishment of the awareness of what other users are doing, and being able to trace the actions taken which might lead to a smoother collaboration.

### Availability

*“The services implemented in a system should be available and function properly”* [69, p. 8]

The availability of a system is part of security as well. A system should be reachable and work failure-free. While reachability can be influenced by the system architecture (cp. 2.2), failure-free execution mainly depends on the quality of the implementation of the system as well as on the tolerance of the system in regard to non-predicted states (cp. [36]). The quality of the implementation could be encouraged using tools which guide a developer, ensuring that critical parts of the code are covered by the implementation, and which might even provide default implementations and fallback solutions to make sure that the application and its functionalities are available and working.

### Controlled access

*“Only authorised entities should be able to access certain services and data”* [69, p. 8]

One important part of security is access control. Usually, some areas/functionalities of software will be restricted to a few authorized users. To authorize users, different possibilities exist. In static architectures, it is common to have one authentication instance (e.g. Kerberos [61]), that checks if and which access rights exist for a specific user. But, especially in dynamic systems, such a centralized check of access permissions is not possible: *“Mobile entities will often become disconnected from their home networks and must be able to make fully autonomous security decisions; they can’t rely on specific security infrastructures such as certificate authorities and authorization servers.”* [13]

For such dynamic systems (e.g. *Peer2Peer* – cp. 2.2.1), different approaches exist for authenticating users. Usually, so called *reputation systems* are applied which are based on trust between the users involved: *“Reputation systems have been often used in P2P networks to involve trust such as [87] [40], among others”* [2].

Although open collaborative systems without access control are imaginable, the control of access is a need in many collaborative scenarios. Because of its dependency on the available resources and system architecture, different solutions should be applied depending on the type of collaborative system.

Since collaboration can appear ad-hoc, and therefore a non-defined user group might appear, centralized user management with predefined user accounts is not suitable. Additionally, collaborators very likely have social relations of trust between them (they probably know each other at least indirectly through a third person when collaborating). Reputation based systems, therefore, seem to be a natural fit for authentication, especially in ad-hoc scenarios, and have to be considered seriously for control access in collaborative systems.

## 2.6 Interaction with shared devices

*"Unfortunately, modern window systems are tied to the notion of a single cursor, and application developers must go to great lengths (and suffer performance penalties) to implement multiple cursors" [65]*

If multiple users want to access a public device such as a shared screen (cp. 2.1.4), this not only implies issues of system support or conflict management logic (cp. 2.4), but also of visualization. For example, the simultaneous editing of a text box by multiple users or the parallel editing of multiple text fields is not easily feasible with standard components of today's libraries for graphical user interfaces, since they are only prepared for a single focused element at a time and expect only one input reference (text cursor) in this focused widget.

Several technical solutions to support multiple input devices exist – e.g. the *Multi Pointer X-Server (MPX)* of Peter Hutterer [37] is an extension of the *X-Window-System* [67], which has been integrated into the *X.Org*-system [125] and which enables the use of multiple mouse pointers controlled by different input devices on Linux systems. Although the basic support of this extension has been adopted by the widely distributed graphical software library *GTK+* [100], this support is restricted to the extension of the programming interfaces and does not include the application of these new possibilities to the graphical components and multi-focus functionalities: *"Missing stuff [...] – Complex GtkWidgets need multipointer awareness, this can be done one by one [...] – per-pointer tooltips and multiple keyboard foci? do we want to get that far?"* [101]

If such functionalities are to be supported, the use of specialized graphical frameworks (e.g. the *Windows MultiPoint Mouse SDK* [111]) or the self-made extensions of existing frameworks are the only options for a developer. Therefore, the support of such extended functions is usually combined with an essential restriction of applicable technologies and reduces, in almost any case, the number of supported platforms since the portability of such specialized libraries is rather poor. Therefore, multi-user support represents a serious restriction on supported devices within a collaborative environment and ways have to be found to overcome these limitations.

## 2.7 Aspects of user experience

In addition to the rather technical topics of computer supported collaborative work presented above, topics which affect the actual use and therefore the user experience of the different collaborators have to be taken into account as well. Two main topics in this context are user awareness and the distribution of user interfaces.

### 2.7.1 User awareness

One of the key elements of a collaborative system is the establishment of awareness about the activities of other users. The way in which actions of other users are represented (e.g. by displaying their mouse pointers on a shared resource) have to be differentiated between those which originated in a remote location and those that originated in a co-located context. Hill claims that “... *we need to distribute the synchronization events, but have visual representations of those events that are appropriate for either the local or remote user.*” [32, p. 545f]. And Tang et al. find that “*Presence disparity unbalances a collaborator’s experience of the group: maintaining awareness, sensing engagement and involvement and communicating is much easier with collocated collaborators compared to remote collaborators.*” [78]

### 2.7.2 Distribution of user interfaces

*“Developing user interfaces for a heterogeneous environment is a difficult challenge. Partial distribution of the user interface is an even harder one.”* [3]

The distribution of user interfaces – e.g. in multi-display groupware (MDG) scenarios – can be established with many different approaches. The idea to distribute components or parts of user interfaces to different devices and therefore to address the issue of restricted amount of space for visualization or to present private information on personal devices is obvious but not trivial.

Although some work that addresses the distribution of non-visual parts of graphical user interfaces (e.g. sound) exists (“*For example, a wireless PDA that lacks audio output is enabled [...] to exploit a stereo speaker in the vicinity for playback of the audio component [...]*” [31, p. 222]), most of the contributed work focuses on the distribution of graphical user interfaces (GUI).

Besides the variant to distribute the user interface equally by the concept of *What You See Is What I See (WYSIWIS)* [74] and therefore to duplicate and transfer complete views to the different devices (cp. [23]), solutions exist that allow to separate specific elements of the interfaces and therefore distribute only partial views.

Usually, such distributed user interfaces (DUIs) are combined by multiple components which communicate with and influence each other but are still independent and can be seen as a single functional entity. One example for such separable components are the different tools of an image editing software (e.g. GIMP [99], cp. Figure 2.3).

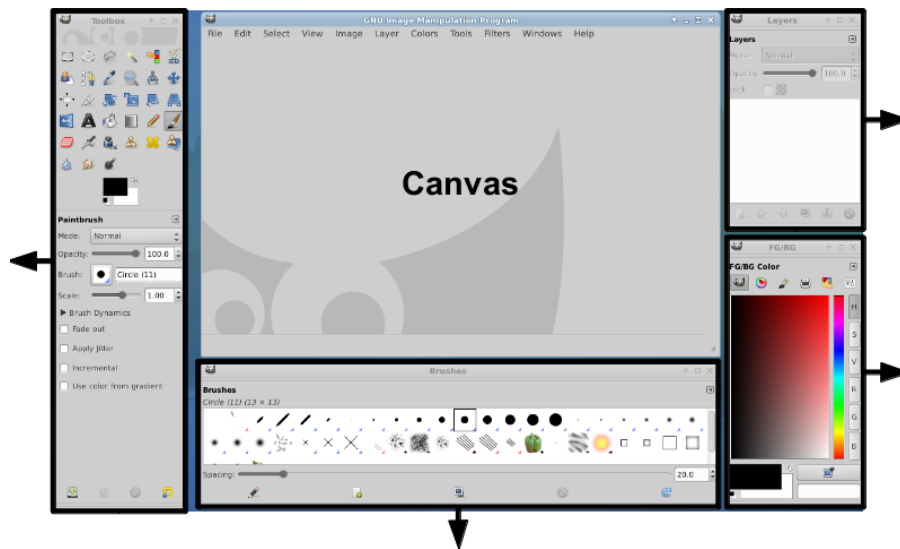


Figure 2.3.: Separatable GUI components in the example of the image editing software GIMP [99]

In the context of web technologies, the concept of applications combined by multiple different parts are known as *Mashups* (cp. [91]), where the different (visual) components define the granularity of the distributed interface.

The definition of such parts of a user interface can either be done explicitly by the declaration of meta data within the program code (cp. [31]), as part of the models in model driven architectures (MDA), or implicitly by the implementation of independent graphical components (e.g. dialogs, tabs, popups, etc.).

More complex variants of distributed user interfaces additionally react on the different specificities of the executing devices, and adapt their functionalities, for example to the provided interaction modalities: *"DUIs allow for the UI to be spread out over a set of displays/devices/platforms taking advantage of their unique properties instead of residing on a single display/device/platform with the interaction capabilities that are constrained on this display/device/platform"* [55].

Problems arise if the user interface needs to be distributed dynamically in a non-predefined environment: *"[...] there is almost no genuine DUI since UI elements have been developed in such a way that they simply remain in their initial context, while communicating with each*

other, but without any possibility to be rearranged.” [55, p.13]. Vandervelpen et al. differentiate between different distribution strategies depending on who decides on the distinction: *user-driven distribution* (the user decides explicitly, which components are distributed to which device), *system-driven distribution* (the system detects the available devices and defines an initial distribution) and *continuous distribution* (the system reacts to changes at runtime and re-distributes components if necessary).

For the system supported distribution logics, Luyten et al. [53] distinguish between *task-oriented*, *device-oriented* and *location-oriented* distributions, which – depending on their category – influence the distribution logic. In task-oriented distribution the task to execute decides which components are to be displayed on a specific device (e.g. based on the specific role of the user within the collaboration). In device-oriented distribution the available devices are analyzed for their functionalities and the user interface is distributed based on the different capabilities of the devices. Location-oriented distribution makes its distribution decisions based on the spatial distribution of the devices within the collaborative environment.

The distribution is also affected by the type of component: some components can be distributed multiple times (e.g. toolboxes), whereas some should be instantiated only once (e.g. canvas).

Most of the time a device-spanning management logic (such as the *Interface Distribution Daemon* in [53]) is applied which has – in the context of multi-user applications – to take device groupings (e.g. if a user owns multiple devices) into account when distributing the visual components.

An advanced distribution mechanism for user interfaces therefore should support the rearrangement of user interface components (either replicated or complementary), distributed by exchangeable decision logics.

## 2.8 Toolkits and solutions for collaborative applications

Thus far, numerous toolkits and frameworks for collaborative systems have been created. Most of them focus on single, fundamental problems of collaboration, but seldom on the full complexity of the field. Already in 1996 Roseman and Greenberg find that “*Virtually all toolkits [...] are just prototypes used to explore different ideas, abstractions, and architectures.*” [65] and Hill und Gutwin judge the different solutions to be focused differently: “*Different toolkits focus on different aspects of the groupware development problem: some focus on what can be done through different programming approaches [...], some on distribution architectures [...], and others on simplicity [...]*” [32, p. 544]

In this section, an overview of the different work in this field is given. Examples of solutions addressing partial characteristics of collaborative systems as well as approaches for more complete tools for the development are presented. Since there are many toolkits available, only a selection of them can be discussed here, which are representative of other, similar, but not listed solutions.

### 2.8.1 Multi-user / multi-device support

There are different solutions, which address the fundamental (technical) support of multiple users by software. We would like to distinguish between approaches for the separation of multiple input devices, toolkits for the design of graphical multi-user interfaces as well as solutions that adapt existing (legacy) applications for collaborative execution.

#### Technical solutions for multiple input devices

The separation of different input devices (usually pointing devices such as mices) usually requires an interface on the system layer to access the attached input devices one by one and to react to possible events (e.g. *RAW-Input* [117] and [80, p. 102]). This information is then provided by a specialized programming interface for software development. The range of such solutions varies substantially: While the *MID* project ([34]) – similar to the graphical software library *GTK+* in combination with *MPX* [101] – only integrates the additional device information into the eventing mechanism of the underlying technology and leaves the handling of that information to the developer, solutions like the *Windows MultiPoint Mouse SDK* (cp. [111]) or the *SDGToolkit* (cp. [80]) provide additional graphical elements, which offer fundamental support for the corresponding extra information (e.g. “multipoint button” in *Multipoint Mouse SDK*, or text boxes with multi-focus functionality in the *SDGToolkit*).

The dependency on system specific interfaces of such toolkits and libraries not only negatively affects their portability between different platforms but also leads to a varying range of functionality. While a solution based on *MPX* works with all pointing devices supported by the operating system using higher level of abstraction, others – usually based on low-level interfaces (e.g. *RAW-Input*) – only support some types of devices: “*MultiPoint Mouse SDK supports USB, PS/2, Bluetooth, trackpad, and wireless mouse devices. [...] Other HID devices (such as joysticks and game controllers) are not supported.*” [111].

#### Widget toolkits

In addition to pure multi-user functionality, different projects address the issues of the design and creation process for graphical interfaces supporting multiple users. Besides the *Microsoft*

*Mouse SDK*, which has been presented in the previous section, the *MAUI Toolkit* [32] in particular is dedicated to user awareness in collaborative environments. Thus, the established library for graphical user interfaces in Java, *Swing* [51] has been extended for multi-user support. Here, additional widgets have been developed that for example allow to show which user interacts with which part of the graphical user interface.

### 2.8.2 Extension of legacy apps

Some solutions have tried to transform existing applications into collaborative systems without adaptation of code. One of them is the project *mighty mouse* [10], which is based on *Virtual Network Computing (VNC)* and extends the pure transmission of the screen with a rudimentary floor control. Although this allows a shared view of a specific application, there is only one device at a time that is allowed to interact with the application. Because of the good support of *VNC* on many different platforms, this tool is easily executable on different operating systems. While this is true for standard computers, it is not necessarily the case for other devices such as smart phones or similar. The project *CollabWiseTk* promises “rendering any stand-alone client collaborative, without a code re-write” [48]. Here, the communication structure in the background of a Tcl/Tk application is extended to manage the synchronization between remotely located widgets.

Although very comfortable since applications can be used in a collaborative context without having to be adapted and rebuilt, solutions which extend legacy applications (usually) only clone the application and redistribute it without considering device specificities or different ways of multi-user management and are therefore rather restricted in terms of customizability and extensibility.

### 2.8.3 Communication

The area of communication contains different concepts. In the context of collaborative applications, the connectivity (how devices are interconnected) as well as the design and abstraction of the distribution of information (messaging) for the replication and notification, as well as the load balancing of the execution of program logic on the involved distributed devices are essential.

One of the most common functionalities of collaborative toolkits is the provision of shared data structures as well as concepts for the optimization of network communication between the devices. For example, the *GT/SD*[18] toolkit abstracts the connections between the different devices and provides the opportunity to access shared data with its “shared directory”



The *Kevlar* [35] project focuses on efficient data exchange as well by reducing the bottleneck of common web service communication paths through adaptation of the concept to peer2peer structures – particularly in the context of wide area networks (WAN). The *SyD* [63] middleware unifies the communication management with a device management functionality and provides programming interfaces for the look-up of available services in a collaborative environment.

#### 2.8.4 Distribution of user interfaces

Support for the distribution of (mainly graphical) user interfaces is also the focus of some toolkits. For example, *GrafiXML* [56] allows to define such interfaces by the use of a UI-description language named *UsiXML* which can be translated into different technologies (e.g. HTML, XUL, OSF Motif, etc.).

The *Toolkit for Peer-To-Peer Distributed User Interfaces* by Melchior et al. [56] allows to exchange Tcl/Tk widgets between different devices and the project *WebSplitter* [31] distributes parts of web pages onto different target devices.

Instead of the distribution of traditional graphical user interfaces, the project *ZOIL* [38] focuses on the design and distribution of post-WIMP (“window, icon, menu, pointing device”) user interfaces where the underlying objects (instead of the graphical and functional components) are synchronized between the different devices involved as is the case in most other approaches.

#### 2.8.5 General toolkits

Additionally, there have been some approaches to develop more general toolkits, which try to achieve as complete an abstraction of the complexity of collaborative systems as possible. One of the best-known and most cited toolkits of that kind is the *GroupKIT* [65] project. It addresses – among others – the topic of collaborative session management, programming interfaces for simplified communication between devices (including conflict management strategies), tele-pointers, and means for the visual distinction between remote and local mouse pointers.

The “meta operating system” *GaiaOS* [64] tries to abstract a collaborative environment (“Active Space”) extensively and to allow the developer to interact with a single, abstract instance instead of many different devices and services. Aside the management of the interconnected devices, the system provides a distributed, contextualized file system and manages the component-based applications developed with the provided application development framework.

## 2.9 Discussion

Although numerous solutions and proposals exist for the main issues facing collaborative systems, complexity of the concrete implementation of such applications is still an important issue.

A developer of a collaborative application is not only confronted with these numerous different challenges, but also with the issue of the integration of these different partial solutions. This integration is even more complex because the different partial solutions usually use different technologies which might be incompatible or involve other issues when they are combined.

Additionally, the platform- and/or device-type dependency of most of the existing toolkits implies a restricted extensibility of the chosen technology and, consequently, the solutions cannot, or only with a significant effort, be adapted to the fast ongoing development in the area of hardware and to the appearance of new types of devices: *“Although they (collaborative environments) share a good amount of common functionality, most of them are built from scratch, or are tailored to a specific device platform using proprietary libraries. An open and customizable environment for mobile collaborative applications is still missing.”* [72, p. 118]

Based on the knowledge gained from the related work as well as the strong belief that the effectiveness of existing (and upcoming) solutions for the different issues facing collaborative applications varies depending on the needs of a specific application and therefore no single optimal solution for a single problem can ever be provided, we are convinced that an open, extensible toolkit for the efficient development of computer supported collaborative systems is needed. Consequently, a technology has to be found which is future safe and extensible, which is supported by a maximum number of common device types, which allows colocated, remote as well as mixed presence collaboration and which adapts itself to the properties of the environment (e.g. different representations for remote and colocated users). Synchronous as well as asynchronous collaboration should be supported, as well as the possibility to execute an application in both “ad-hoc” scenarios and in infrastructurally supported environments. Therefore, the toolkit should not be restricted by expecting given, predefined devices and should adapt itself in an optimal way to the current environmental settings. Devices (e.g. private devices of the users) shall be integrated dynamically into the system while obeying the “walk up and use” paradigm to reduce the barriers for ad-hoc use and to let new users easily connect to the collaborative system. To achieve this, a broad heterogeneity of devices should be supported without the requirement of installation or configuration. The system should – depending on its situation – run in peer2peer environments as well as in client-server structures and allow the dynamic integration of resources (cp. 2.2.1). Eventing and synchronization strategies should be predefined and provided as default implementations, which can be overridden and/or replaced by specialized implementations if needed. The

affected user interfaces as well as their components should support multi-user use while still keeping their initial capabilities for single-user use. Structures for a simplified distribution of user interfaces as well as mechanisms to secure communication and user authentication should also be provided.

To increase acceptance of such a toolkit by potential developers, main stream technologies should be used to increase the probability of longevity.

In our research into existing solutions, we have found toolkits which offer development aids in specific environments, for specific purposes or under specific preconditions but no toolkit which is as dynamic and complete as described above.

In 1996, Roseman and Greenberg found that *“Groupware toolkits still have a long way to go to catch up to their single-user counterparts. We look forward to the day when all toolkits, perhaps influenced by GroupKit and others in its genre, will incorporate multiuser features. When that day comes, the artificial distinction between constructing single and collaborative systems will disappear.”* [65] Although significant efforts have been made since then, this finding is now – 16 years later – still valid. The development of groupware (or collaborative applications) is still not as well supported as single-user application development. One reason might be the huge complexity of the general problem of this topic. Another is that technical evolution has during the years created an even more heterogeneous landscape of devices and technologies rather than defining unified and generally accepted standards. Although all the new technologies have extended the possibilities and opportunities, they have increased complexity even more. That is why today the need for efficient toolkits is bigger than ever.

# 3

## Specification

---

<b>3.1. Context of the work</b> . . . . .	<b>39</b>
<b>3.2. Requirements</b> . . . . .	<b>39</b>
3.2.1. Technology . . . . .	39
3.2.2. Software architecture . . . . .	42
3.2.3. Basic functionalities . . . . .	43
<b>3.3. Choice of technology</b> . . . . .	<b>45</b>
3.3.1. Priorization of requirements . . . . .	46
3.3.2. Technology candidates . . . . .	47
3.3.3. Technology decision . . . . .	58
3.3.4. Implications and challenges . . . . .	62
<b>3.4. System architecture</b> . . . . .	<b>65</b>
3.4.1. Overview . . . . .	65
3.4.2. Work load distribution . . . . .	67
3.4.3. Server side functionalities . . . . .	67
3.4.4. Client side functionalities . . . . .	69
3.4.5. Communication . . . . .	70
3.4.6. Set-up . . . . .	71
<b>3.5. Discussion</b> . . . . .	<b>72</b>

---

The related work has shown that there is currently no flexible toolkit available which exhaustively supports the development of any type of computer supported collaborative application.

In this chapter, we will explain where the need for such a flexible software infrastructure came from and in which context we have applied it. We present the facts that have influenced our technology choice and the consequences of this choice for the software engineering process,

as well as the system architecture structure, by listing the additional challenges and benefits that our choice has implied. We then describe the chosen system architecture as well as how the overall architecture applies to the different environment we would like to support.

## 3.1 Context of the work

As part of the project “Interactive Collaborative Environments” (ICE), we are focusing on the optimization of collaborative work through the use of electronic devices. Therefore, our group has set up an interdisciplinary collaboration of computer scientists, HCI and UX experts as well as psychologists. The goal of this collaboration is to explore interaction between users and devices from a variety of perspectives and in different contexts.

To be able to support the varying needs and since researchers without special technical knowledge will be running the experiments, there was a need for a solution which runs on as many different devices and device types as possible, is easy to use and future safe, since the project is intended for the long-term.

Although the main focus of our research is currently set on synchronous co-located collaboration, asynchronous as well as telepresent scenarios might be the focus of future research and therefore the solution has to support all potential types of collaborative systems.

We therefore decided to create a generic toolkit for the development of collaborative applications which suits the needs of the manifold group of stakeholders within this project.

## 3.2 Requirements

To find an appropriate solution for the development of the software for the experiments in our project, different types of requirements have to be defined: Requirements for the *technology* will lead to a decision about which technology should be chosen for the implementation (cp. 3.3.3). Aspects of the *software architecture* define fundamental concepts which should be considered for the design and realization of the concrete modules, and *basic functionalities* are required to provide the necessary support for the establishment of a basic structure for a collaborative system.

### 3.2.1 Technology

Finding an appropriate technology is very important since it has a significant impact on the outcome of our toolkit. The wrong choice of technology could either restrict the functionalities or the supported devices of the resulting solution, or could lead to a lack of acceptance in

the developer community and therefore end up unused. Therefore, based on the needs of our project, we have defined the main requirements as *future-safety and acceptance, dynamics, availability, scalability and heterogeneity, easy set-up* as well as *privacy awareness*.

### **Future-safety and acceptance**

One of the most important requirements that a toolkit which will be used in different scenarios and accepted by other developers should ensure, is that it is based on **well established** – and therefore **well-known** – as well as **future-safe technologies**. This means that it should have broad acceptance and support and neither rely on proprietary technologies (since they could be deprecated by the company that owns them), nor on those which are currently trendy but have not been applied to a broad range of applications yet. A trend can pass, and if there are not enough applications of the technology, support and ongoing development can stop rather quickly. Another important aspect is that developers usually have their preferences of technologies, which depend mostly on their personal experience. Although most of them are willing to learn new technologies and approaches, they usually like to be able to reuse well-known concepts and languages. Therefore, to address as many potential users of the toolkit as possible and reduce their efforts for learning new technologies, the toolkit should be based on widely accepted programming languages and concepts.

The future-safety of a technology is always hard to predict since evolution in this area occurs very quickly and can hardly be foreseen. Indicators for longevity of a technology can be found in the degree of standardization (the more standardized the more potential implementations exist and therefore the lower the probability of deprecation of the technology), the technology's current spread (if a technology is widespread, the availability of legacy-support and backwards-compatibility are more probable than for technologies which are in little use) as well as the adaptability of the technology itself to new concepts and improvements.

### **Dynamics, heterogeneity, availability and scalability**

Given our context, we mainly have collaborative systems in mind which integrate multiple (private as well as shared) devices to solve collaborative tasks, and are therefore building a distributed system. This is why the chosen technology should be applicable in very dynamic situations, i.e. where very heterogeneous devices appear and disappear unexpectedly. This dynamics of device appearance and disappearance is not only caused by technical issues (loss of connectivity, low battery of devices, etc.) but also by the (potentially) ad-hoc nature of collaborative settings where users can join and leave at any time.

To support heterogeneous devices, the technology should not only support many different platforms and device types but also take into account that the capabilities of those devices differ – e.g. in terms of performance, screen size, network interfaces, etc. – and therefore should distinguish between the components which can be executed on the different devices (e.g. a smart phone cannot display big images but can be used as a text entry device). If a device does not fulfill the requirements of some components of the system (because of a lack of capabilities due to restrictions defined by the user), the device should still be integrated into the overall system, even though the set of supported components is restricted. Therefore, the technology should not decide on device support using an “all or nothing” concept, but rather try to integrate as many devices as possible.

To ensure availability of the system, the toolkit should make sure that the environment for the application execution can be kept as stable as possible. Therefore, it should be flexible enough to be applied to different system architectures to either profit from a stable environment with infrastructural support or to achieve any needed redundancy by allocations of multiple server instances to ensure a stable environment with high availability while trying to keep the management overhead for the replication and synchronization as low as possible.

In addition to the availability, scalability should be considered as well. If for examples many additional devices arrive in a specific situation, the system should find ways to scale accordingly, either by dynamic allocation of additional resources or by optimizations of the communication structure between the devices (e.g. load-balancing).

### **Easy set-up**

A collaborative system which should be set up within a short time by persons with unknown technological knowledge (e.g. end-users in an ad-hoc situation) has to ensure that the set-up process is as easy as possible. One key factor to achieving such an easy set-up system is the realization of the application with a configuration and installation free technology, creating a “walk up and use” scenario. Only if the configuration and installation requirements can be reduced to a minimum, can it be ensured that a simple set-up is achievable.

### **Privacy awareness**

When users interact with potentially unknown collaborators and systems using their private devices – which often hold sensitive personal information – the need for ensured privacy arises. When users connect to a system the first time and they do not have any information about the trustworthiness of the application that will be executed on their device, they need to be made aware that the system is under their control using clear restrictions that they

themselves can adapt. Therefore, the chosen technology should provide means to ensure that the executed application is running with only restricted privileges and explicitly asks for permission for anything that is out of scope of the set of allowed functionalities. This way, users can be sure that no files or other device resources can be accessed either by the system itself or by other devices connected to the system unless explicit permission is given by the user.

### **3.2.2 Software architecture**

Besides the requirements of the technology itself, software architecture aspects influence the suitability and acceptability of the toolkit. These requirements therefore have to be considered when the toolkit is actually engineered and implemented.

#### **Simple APIs and reuse of concepts of the technology**

To make sure that developers feel comfortable with a toolkit, new concepts and specific interfaces should be included as rarely as possible. To minimize the overhead of becoming familiar with a new technology, it is important that concepts and conventions of the chosen technology are obeyed and kept constant as much as possible. Therefore, one goal of the design and definition of the toolkit APIs is that the extended functionality should be hard to distinguish from the standard-set of functionalities of the underlying technology and that it should reuse standard interfaces whenever possible. This not only helps the developer profit from their experience, common knowledge and a broader supportive community, but the toolkit can also profit from the ongoing development of the underlying technology provided by third parties.

#### **Modularity and extensibility**

The toolkit should be built in a modular way. Since its purpose is to provide functionality for varying scenarios, not all functionalities will be needed and therefore, unnecessary functionalities should be excludable from an application. Additionally, different implementations for a specific issue could be better suited depending on the type of application and therefore should be replaceable. Modularity helps third party developers add their own components and therefore extends the toolkit if they have a specific need that has not been addressed (or not in the needed way) yet.



### **Resource saving through lazy code loading**

Since there can be different modules of functionalities, not all of them will be executed on every device and at the same time. To save resources (e.g. memory, cpu power, etc.) it is important (especially because low-end devices with very restricted capabilities may be involved) to reduce the overhead of non-used elements and therefore to load and execute the different elements “lazily”, at the very moment that they are accessed. This way, the device only has to handle the code that is actually executed and can save both internal resources and network traffic.

### **Device-specific adaptation by design**

Different devices need different implementations of components. Examples of device-specific needs are different layouts for devices with differing screen sizes, the size of buttons, awareness of different input modalities (touch events / mouse events) and other interactive elements based on the type of device. Since the heterogeneity of devices our toolkit should support is wide, the need for customization based on different types or capabilities of devices should be regarded as essential. Therefore, our toolkit needs to provide ways to differ between the specificities of devices by design – meaning that it should contain prepared structures for replacing implementations based on what kind of devices the application runs on.

### **Communication: Responsiveness and optimized network traffic**

Communication is an essential part in any type of distributed systems. As seen in the related work, synchronous collaboration in particular – due to the need for good system responsiveness – has very high demands on low network latencies, which makes the organization and establishment of communication a huge challenge. The system therefore should not only provide comfortable ways (APIs) to let the different devices communicate with each other, but to also take the optimization of network-traffic into account.

### **3.2.3 Basic functionalities**

In addition to software architecture requirements, the toolkit has to provide solutions for concrete needs which do not affect the overall technology but which have to be handled in almost any (collaborative) application. Therefore, the following features define a set of fundamental functionalities which should be provided by the toolkit:

### **Easy access**

While the simplicity of the initial set-up of the system is supported by choosing an appropriate technology (cp. 3.2.1), potential users should also be able to connect their devices to the already running overall system in a simple and easy way, without the need for special technical knowledge. The creation of a network connection to the surrounding environment is part of the process of entering a collaborative system, as is the launch of the current client application which provides the needed functionality to interact with the other devices. The toolkit therefore should provide ways to simplify these tasks.

### **Messaging and eventing**

Since we are in the context of distributed systems, the relevant coordination and (conflict) handling of messages and events should be provided by the toolkit. Aware of the fact that the effectiveness of these mechanisms depends on the actual need of the application (cp. 2.4), the toolkit should not provide a single eventing mechanism that is assumed to suit any type of situation, but rather should provide different kinds of implementations allowing developers to choose the one that is the most suitable for the system they are developing. The developers have to be made aware of the advantages and disadvantages of the chosen strategy by the toolkit through hints as part of the API (e.g. by the requirement of an explicit selection of a specific strategy) which at the same time hides the complex realization of the implementation behind the scenes.

### **Application level security**

Security aspects come up when messages are exchanged between devices which should not be seen by other participants or devices of the collaborative setting. Besides system messages (events), this applies to user messages as well. Especially when working in ad-hoc mode without trusted instances (because devices belong to a potentially unknown user), the toolkit should provide means to ensure that information is kept secure independently of how it moves through the network. This can be achieved by encryption of the messages before their transmission over the network.

### **Multi-user support**

The support of multiple users often implies the introduction of shared resources which might need to be accessible synchronously. Although almost no current operating system or graphical user interface library supports the interaction of multiple input devices (e.g. multiple

mouse pointers, multiple keyboards) on the same application, the toolkit should provide a way to transform single-user scenarios into multi-user scenarios and provide means to handle conflicts. At the same time, the developer should be released from having to handle the effects which are caused by multi-user scenarios and should therefore be able to implement and test in well-known (single-user) scenarios.

### **Grouping of devices**

If Alice owns both a tablet and a smart phone she might want to simultaneously use one of the devices for controlling the mouse pointer on the shared screen (e.g. the smart phone) and the other device (the tablet) for text entry. To be able to do this, the two devices have to be grouped to allow the system to interpret the events triggered by any of Alice's devices as originating from a virtual device spanning all physical devices that Alice owns. The toolkit therefore has to ensure that her actions in the collaborative environment are annotated with her user account regardless of which of the two devices triggered them.

### **Distribution of user interfaces**

It might be interesting to distribute components of the user interface to different devices. One of the most common examples of distributed user interfaces is having a painting application where the canvas is presented on a big screen and the color palette is provided on a mobile device.

The distribution of user interfaces not only implies issues of state replication and synchronization between the different user interface components, which have to be solved to ensure that all user interfaces react to the influences of actions executed on other devices, but also raises the need for dynamic layouting mechanisms which allow the addition and the removal of components at runtime on the different devices (e.g. when redistributing a graphical component to another device). Besides the manual control of which element should be distributed to which device, the toolkit should provide means to automate this process by enabling clever algorithms to make the distribution of user interfaces more fluent, even in very mobile and dynamic environments.

## **3.3 Choice of technology**

The choice of technology can have a big impact on the viability of the toolkit. Therefore, the choice was made based on the knowledge gained from the related work and on the technological requirements mentioned above. After an overview of the different potential

candidates, we present our decision making process as well as the implications and challenges of our decision.

### 3.3.1 Priorization of requirements

Although all requirements defined in 3.2.1 should be considered in the technology decision, some of them are more important than others. Therefore, we first prioritize the requirements and categorize them into “must have”, “important” and “nice to have” requirements.

#### “Must have” requirements

“Must have” requirements are those which have to be fully fulfilled by the chosen technology. One of the most important goals is to establish a toolkit which can be used a long time in order to support experiments during the whole duration of the project ICE. **Future-safety** and **developers’ acceptance** are therefore two of these “must have” requirements since different scientists and engineers will have to work with the provided solution over time.

Given our context, we want to support as many different devices and device types as possible to be able to compare them and their influence on collaborative settings. Additionally, we do not want to restrict collaboration to predefined settings but also take its natural dynamics (in terms of users and devices) into account. The **support of heterogeneity of devices** as well as the **support of dynamics of devices** are therefore essential as well.

#### “Important” requirements

“Important” requirements are those which can have a big influence on the capabilities of our system but can – in case of multiple technology candidates – be relaxed if necessary. This means that the main functionality of these requirements has to be achievable but compromises can be made in the extent to which they are fulfilled.

The **easy set-up of the system** is an important aspect for the acceptance by end-users as well as for allowing ad-hoc scenarios. In the best case, a true “walk-up-and-use” scenario can be established, where only very little or almost no preparation work has to be done before being able to use the system. Other requirements such as **scalability and reliability** are important too because we do not want to restrict our collaborative settings by technological limits but rather depending on the actual usability restrictions for the end-users. Nevertheless, some natural limits are given (e.g. it is hard to imagine a synchronous colocated collaboration scenario with a hundred participants or more). Therefore, scalability is important to us only to a specific (realistic) limit.

### “Nice to have” requirements

“Nice to have” requirements are those which help to establish more comfortable and more powerful collaborative solutions. Although it would restrict the capabilities of our solution, the non-fulfillment of these requirements would not necessarily exclude the technology from being considered in our decision making process.

Since in our scenarios personal devices can play an important role, the question of **privacy** should be taken into account to make sure that no bad feelings arise due to privacy issues when end-users are interacting with the system using their personal devices. The **availability of toolkits and frameworks** for the development of applications with a specific technology helps to support the developers and therefore simplifies the development in important ways. Nevertheless, most issues are already overcome with the requirement of the *developers’ acceptance* and therefore this requirement can be seen as a nice to have supplement. Although very interesting for integrating new functionalities and making use of the very special features of a specific device, **access to low-level functionalities** is only nice to have as long as standard device functionalities are supported by the technology.

### 3.3.2 Technology candidates

The following categories contain different candidates of technologies which could be used to realize a toolkit that fulfills our defined requirements. We talk about the advantages and disadvantages of the different possibilities and compare them, to be able to make a founded and valid decision. We distinguish between *native programming languages*, which typically target a specific type of platform or device type, *model driven architectures* and *cross-compilers* and *source to source translators* which can make native technologies more portable and *platform independent languages* which run on many different platforms by abstracting a specific set of functionalities. Additionally, we present *dynamic languages* in general and *web based technologies* (with and without plugins) and their possibilities for extension.

#### Native programming languages

Native programming languages target a specific platform and allow to make extensive use of the resources of a device. Typically, they provide programming interfaces which offer low-level access of device functionalities, making them quite powerful in terms of possibilities, but also in the harm they could cause since they usually have very few restrictions on how they can affect the main system (they usually have the same permissions as the system user has).

### ***Advantages***

Native programming languages let the developer address very device specific capabilities. Developer access to (low-level) functionalities is only slightly or not at all restricted, which helps customize and optimize programs for a specific device. The performance of native applications is usually rather good and their look and feel is in most cases consistent with the operating system as well as with other applications which usually improves the user experience.

### ***Disadvantages***

Native programming languages on their own support only a restricted set of devices and/or platforms. If a program needs to run on a heterogeneous set of devices, re-programming with other native technologies is needed in most cases and is sometimes rather hard to achieve due to the necessary adaption of low-level functionalities. Due to the manifold implementations needed in a heterogeneous environment, maintenance efforts can become an issue since changes or bug fixes have to be applied to all versions of the program.

Native programming languages have difficulty handling unknown devices since they need preprocessing (compilation) for the new platform they should support. Support for a new type of device is therefore difficult to establish.

“Walk-up and use” is restricted when using native programming languages since the application has to be installed on the device before it can be used. Additionally, since permissions of a native application usually correspond to the system permissions of the executing user, it is not easily possible to restrict the capabilities of that application at runtime. Therefore, the users have to trust that the application does not do any harm to their private data. Although mobile platforms in particular have several coarse-grained permission requests which have to be accepted before installation (e.g. through the AppStore of Apple or Google Play), users only have the choice to “take it or leave it” meaning that they cannot decide on specific permissions but must either install the software and accept all permissions or decide to not to install the software and therefore not to be able to use the functionality.

**Examples**

C, C++, C#, Dalvik (Android) and Objective C (iOS)

**Fulfillment of requirements**

Requirement	Category	Description
Future safety	Must have	<b>Good</b> – Some of the native programming languages are widespread and constantly adapted to new developments
Developers' acceptance (well established & well-known)	Must have	<b>Good</b> – Some of the native programming languages are very well accepted by huge communities of developers
Support of heterogeneous devices	Must have	<b>Bad</b> – With native languages, the software has to be re-programmed or at least recompiled for new platforms
Support of dynamics of devices	Must have	<b>Medium</b> – The reaction to the dynamics of devices depends on the actual implementation.
Easy set-up of the system (“walk-up-and-use”)	Important	<b>Bad</b> – Software developed with native languages usually has to be installed on the device before it is ready for execution.
Scalability and reliability	Important	<b>Medium</b> – Scalability and reliability relies on the actual implementation of the software
Privacy awareness	Nice to have	<b>Bad</b> – Native applications are hard or even impossible for the user to restrict at runtime. Native applications therefore have the potential power to misuse private data and resources when installed on a personal device.
Accessible low-level functionalities	Nice to have	<b>Good</b> – Native applications usually do not have any restrictions on access to low-level functionalities and are therefore able to provide them to the application.
Availability of toolkits and frameworks	Nice to have	<b>Good</b> – For most native programming languages, various development tools as well as third party libraries exist.

**Model Driven Architectures, cross-compilers and source to source translators**

Due to the disadvantages of native programming languages, different concepts exist for how (especially) native applications can be made more portable between devices and platforms.

One of them is the definition of the program logic in a more abstract way through the use of models. These models then generate actual program code in different shapes – e.g. in different programming languages. Cross-compilers are able to compile a native code to different versions of machine code (e.g. for different platforms) while source to source translators translate (as their name says) the source code from one programming language to another one.

**Advantages**

All of these “abstractions” help to reduce initial development and maintenance efforts since changes have to be made only in one place. They allow to profit from a broader portability of the application between platforms and therefore improve the support for heterogeneity.

### *Disadvantages*

The abstraction comes at the cost of less control over device specific implementations unless they are defined explicitly (e.g. by configuration of an alternative implementation for a specific platform). Although these constructs help to increase portability, the number of supported platforms in practice is usually still rather low and therefore, the issues of native programming languages are solved only partially.

### *Fulfillment of requirements*

Requirement	Category	Description
Future safety	Must have	<b>Good</b> – Since with these technologies, implementations in other technologies are generated, they are adaptive to the evolution of technologies and could replace a technology if it is deprecated.
Developers' acceptance (well established & well-known)	Must have	<b>Medium</b> – Model driven architectures are not very widespread and therefore not very well known to developers.
Support of heterogeneous devices	Must have	<b>Medium</b> – Although potentially applicable to many different platforms, practice shows that the amount of different platforms supported by model driven approaches is often limited
Support of dynamics of devices	Must have	<b>Medium</b> – The reaction on the dynamics of devices is depending on the actual implementation.
Easy set-up of the system (“walk-up-and-use”)	Important	<b>Bad</b> – Resulting code is usually based on native programming languages and therefore usually has to be installed on the device before it is ready for execution.
Scalability and reliability	Important	<b>Medium</b> – Scalability and reliability relies on the actual implementation of the software
Privacy awareness	Nice to have	<b>Bad</b> – Resulting code is usually based on native programming languages and therefore the same restrictions for privacy ensurance apply.
Accessible low-level functionalities	Nice to have	<b>Medium</b> – Although potentially accessible, low-level functionalities are hard to use since model driven approaches define logic on a more abstract level and device specific implementations therefore have to be made explicit by re-definition.
Availability of toolkits and frameworks	Nice to have	<b>Medium</b> – Depends on the actually chosen technology

### **Platform independent languages**

Platform independent languages such as Java run on all machines which have a runtime of the specific language installed since the code is actually interpreted by an intermediate layer – also known as a “virtual machine” (VM). The coverage of functionalities of that virtual machine can differ depending on the implementation. For example the Java language differentiates between the Java Micro Edition (JavaME) with fewer functionalities, the Java Standard Edition (JavaSE) with standard functionalities and the Java Enterprise Edition (JavaEE) with extended functionalities.



***Advantages***

A platform independent language allows to write an application once and execute it on any machine – independent of its platform or device type – as long as the machine has a corresponding virtual machine installed. It is therefore not necessary to change and/or re-compile the code for a new type of device that needs to be supported.

***Disadvantages***

To run a platform independent language a virtual machine must be available for the specific device. For some devices no such virtual machine exists and for others only a reduced version of that VM (e.g. JavaME) is available which might have restricted capabilities and thus not be able to execute the application. This restricts the heterogeneity of supported devices and could therefore prevent some users from participating in a collaboration. Additionally, since the execution of such an application requires the pre-installation of the virtual machine, this technology restricts the “walk-up and use” capability of the application as well.

Although a platform independent language runs in a sandbox (the VM) and therefore more fine grained permission control would be possible, the manipulation of these values in practice is not easily achievable (especially for users with limited technical knowledge) and many of the possible configuration options are not at all editable.

***Examples***

Java, Groovy, Scala, JavaFX, Clojure, etc.

*Fulfillment of requirements*

Requirement	Category	Description
Future safety	Must have	<b>Good</b> – Java and its virtual machine in particular are widespread and adapt to new concepts regularly.
Developers' acceptance (well established & well-known)	Must have	<b>Good</b> – A huge Java developer community exists and the language is very well-known in academics as well.
Support of heterogeneous devices	Must have	<b>Medium</b> – Although Java code itself is platform independent, the need for an interpreter which is not available for every platform reduces the number of target devices.
Support of dynamics of devices	Must have	<b>Medium</b> – The reaction to the dynamics of devices depends on the actual implementation.
Easy set-up of the system (“walk-up-and-use”)	Important	<b>Medium</b> – Although the installation of a virtual machine is required before code can be executed, this is a one-time installation which can then be reused for other applications.
Scalability and reliability	Important	<b>Medium</b> – Scalability and reliability relies on the actual implementation of the software
Privacy awareness	Nice to have	<b>Bad</b> – Although running in a virtual machine and therefore theoretically configurable in terms of allowed functionalities, practice shows that interfaces and tools are seldomly provided to the end-user for such configurations at runtime.
Accessible low-level functionalities	Nice to have	<b>Medium</b> – Low-level functionalities can be accessed if the virtual machine provides direct access or through the use e.g. of interfaces to native programming language (e.g. JNI).
Availability of toolkits and frameworks	Nice to have	<b>Good</b> – several mature development environments exist for such languages

**Web browser plugin technologies**

A special case of platform independent languages are plugin technologies of web browsers. Most of these languages allow their execution in both environments – within web browsers and standalone applications. Although the virtual machine is usually the same for both modes, the permissions of the executed application differ.

*Advantages*

One of the main advantages is the possibility to execute the application in different modes and therefore other permission settings. An application can – in case of doubts about its trustworthiness – be executed either in the browser with a restricted set of functionalities (and therefore a restricted risk of harm) or as a standalone application with a lot of capabilities and therefore extended functionalities. Although such technologies typically require the creation of two separate versions for the different modes, they can be based on the same code for common functionalities which reduces efforts for maintenance.

*Disadvantages*

Again, a pre-installation (of the plugin or the virtual machine) is required and therefore “walk-up and use” is restricted. Additionally, plugins do not exist for all platforms and

browsers and there exists a chance that they will lose importance in the near future due to the rise of more standardized technologies (HTML5, CSS3 & WebGL) for their main use cases (typically animations and complex graphics).

### *Examples*

Adobe Flash, Microsoft Silverlight

### *Fulfillment of requirements*

Requirement	Category	Description
Future safety	Must have	<b>Bad</b> – Web browser plugins are usually proprietary solutions and can be deprecated at any time. Additionally, HTML5 in combination with CSS3 and WebGL might compete with these technologies and have several advantages due to their degree of standardization.
Developers' acceptance (well established & well-known)	Must have	<b>Medium</b> – Although developer communities exist for these technologies, only a few applications have been created with these technologies, which have been built for a long lifetime. Therefore, the establishment of these technologies is not given in a long-term perspective.
Support of heterogeneous devices	Must have	<b>Medium</b> – Several plugin technologies do not support all main platforms and therefore do not fully support a very heterogeneous set of devices.
Support of dynamics of devices	Must have	<b>Good</b> – Although the system architecture can still be varied, a technology with client-side application execution allows to join and leave rather easily because its existence does not affect the execution of the overall system as long as they do not provide server functionalities.
Easy set-up of the system (“walk-up-and-use”)	Important	<b>Medium</b> – A one-time installation of the browser plugin is required. Once installed, the application can be started by simple access of the URL within a web browser.
Scalability and reliability	Important	<b>Good</b> – Although scalability and reliability relies on the actual implementation of the software, the reduction of importance of client devices (because the disappearance of a client device does not affect the functionality of the overall system) helps to improve reliability.
Privacy awareness	Nice to have	<b>Good</b> – Web browser plugins run in a sandbox and are therefore restricted by the plugin capabilities.
Accessible low-level functionalities	Nice to have	<b>Medium</b> – An application written with this type of technology only can access the low-level functionalities which are provided by the plugin technology. The plugin technology itself has full access to low-level functionalities since it is written as a native application.
Availability of toolkits and frameworks	Nice to have	<b>Medium</b> – Although all web browser plugins provide development toolkits, these are mostly provided only by the company that owns them and the number of these toolkits is therefore rather restricted.

### **Dynamic languages**

Because of their nature, which allows the execution of dynamically added code at runtime, dynamic languages are mostly interpreted script languages. For the interpretation, such languages need the availability of an existing interpreter for the platform they are currently running on.

### *Advantages*

The dynamics of those languages lets them execute code at runtime, which allows to “inject” code. This is an essential pre-requisite, especially for lazy code loading and execution. Additionally, they do not suffer as much from platform-dependency as long as they do not access device specific libraries.

### *Disadvantages*

These languages require – just as the platform independent languages – the availability of an interpreter on the devices and platforms they support. The support of heterogeneity therefore directly depends on the availability of the interpreter.

### *Examples*

JavaScript, Perl, Ruby, PHP, Objective-C, etc.

### *Fulfillment of requirements*

Requirement	Category	Description
Future safety	Must have	<b>Good</b> – Dynamic languages usually have a very good adaptiveness to new trends and are very flexible for extensions.
Developers’ acceptance (well established & well-known)	Must have	<b>Good</b> – Many of the most common dynamic languages of technology have big developer communities and are applied in many different software projects.
Support of heterogeneous devices	Must have	<b>Medium</b> – The support of heterogeneous devices varies between the different languages of this category. Some of them have a very broad support because of their wide spread interpreter while others focus only on some specific platforms and device-types.
Support of dynamics of devices	Must have	<b>Medium</b> – The reaction to the dynamics of devices depends on the actual implementation.
Easy set-up of the system (“walk-up-and-use”)	Important	<b>Medium</b> – A one-time installation of the interpreter is required.
Scalability and reliability	Important	<b>Medium</b> – Scalability and reliability relies on the actual implementation of the software.
Privacy awareness	Nice to have	<b>Medium</b> – Whether the application can be restricted in terms of functionality to ensure privacy during runtime depends on the actual dynamic language.
Accessible low-level functionalities	Nice to have	<b>Medium</b> – Depending on the execution context, some of these languages allow direct access to low-level functionalities while others restrict them.
Availability of toolkits and frameworks	Nice to have	<b>Medium</b> – The availability of toolkits and frameworks depends on the actual language of this category.

### **Standard web technologies**

Web technologies are based on the HTML markup language which can be styled by Cascading Style Sheets (CSS). Modern web applications usually also contain ECMAScript (also known as JavaScript) to add functionalities to the web page which are executed within the

browser. Web applications which make use of JavaScript as well as asynchronous resource loading are called AJAX (Asynchronous JavaScript And XML) applications. When we are talking about standard web technologies, we are mainly talking about the full standardized technology stack (HTML, CSS and JavaScript) while HTML and CSS are only used as the representational layer (and therefore for visualization) and can be created and combined dynamically using JavaScript<sup>1</sup>.

JavaScript is a dynamic language and has a special position because of its widely distributed interpreter (included in almost every web browser) and because of its distributed nature. Additionally, it has interesting aspects in terms of “walk-up and use”.

### ***Advantages***

Standard web technologies run without installation or configuration and without re-compilation of the application on any device running a (modern) web browser, which is pre-installed by default on many devices.

Due to its wide distribution (the whole internet is based on these technologies), the probability of longevity is rather high. Additionally, evolution of the technologies (especially HTML5, CSS3, WebGL and web sockets) is providing new features to create rich user interfaces which were not possible before.

Additionally, web technologies have a huge developer community which is supported by well-established and feature-rich development kits (e.g. the Google Web Toolkit, jQuery, etc.). Privacy can be ensured since web applications run in a very restricted sandbox (the web browser). Nevertheless, several interfaces to device-specific resources exist (e.g. geo-location) but the application has to ask for permission first and the users can decide if they want to let the application access this additional element. The same is true for private data, since local files on the device can only be accessed with explicit permissions from the user (e.g. by the selection of a file in an upload dialog).

### ***Disadvantages***

Although the number of interfaces for extended functionalities (e.g. geo-location) is increasing regularly, web applications still have a rather restricted set of functionalities. Additionally, different web browsers interpret some definitions (in JavaScript, CSS and HTML) differently. Although these issues are partially handled by frameworks and toolkits, this makes development less comfortable. Since web applications cannot define their own communication layer but have to make use of the ones provided by the web browser, they are rather restricted in how they can communicate. The standard protocol HTTP has issues with increased over-

---

<sup>1</sup>Although JavaScript is a dialect of the ECMAScript standard and therefore contains additional functionality, we are using the terms synonymously in this thesis

head for several applications and the uni-directionality of such communication channels (the communication can only be initiated by the web browser but not by the web server). This implies several issues, especially for real-time applications.

Although the communication issues will be solved (at least partially) by the introduction of web sockets, and work-arounds for bi-directional communication for compatibility with legacy browsers exist, network performance will probably never be as optimal as with applications developed with other technologies.

### *Examples*

HTML, CSS, JavaScript

### *Fulfillment of requirements*

Requirement	Category	Description
Future safety	Must have	<b>Good</b> – Because of the masses of already existing content and applications, backwards-compatibility is implicitly ensured. Additionally, the technology adapts rather fast to new concepts and ideas.
Developers' acceptance (well established & well-known)	Must have	<b>Good</b> – A huge community of web application developers exist and web applications are used as front-ends for very important and long-term applications in practice.
Support of heterogeneous devices	Must have	<b>Good</b> – Every device that runs a web browser with JavaScript is supported.
Support of dynamics of devices	Must have	<b>Good</b> – Although the system architecture can still be varied, a technology with client-side application execution allows to join and leave rather easily because its existence does not affect the execution of the overall system as long as it does not provide server functionalities.
Easy set-up of the system (“walk-up-and-use”)	Important	<b>Good</b> – Because web browsers are often pre-installed on many devices, no pre-installation is required, and the application can be immediately accessed by a single URL request.
Scalability and reliability	Important	<b>Good</b> – Web technologies have been developed from the beginning to support multiple clients at the same time. By appropriate distribution of the load between the server and the client, further improvements can be achieved in terms of scalability and reliability.
Privacy awareness	Nice to have	<b>Good</b> – Web applications run within the browser sandbox which is rather restrictive and therefore does not allow a web application to access private data unless it is explicitly allowed by the user.
Accessible low-level functionalities	Nice to have	<b>Bad</b> – Access of low-level functionalities which are not provided by the web browser are not accessible directly.
Availability of toolkits and frameworks	Nice to have	<b>Good</b> – Several tools and frameworks as well as third party libraries exist for the development of web applications.

### **Extended web-technologies**

Projects exist (e.g. [115]) which wrap web applications with a standalone native client that provides JavaScript interfaces for native resources. That way, the restriction of the sandbox can be overcome if a user wants to make use of extended functionality by installing a “native”

application.

### *Advantages*

Web applications can – similarly to plugin based web applications – be run in two different modes: either with a restricted set of functionality but without privacy issues and installation needs, or with extended functionalities (including potential dangers).

### *Disadvantages*

Besides the need to install software, one of the main disadvantages is the use of non-standardized APIs which have to be adapted if the application has to switch to another framework, or if a similar but not identical interface is standardized and included within the HTML/JavaScript specification.

### *Examples*

Phonegap[115], Appcelerator Titanium[94]

### *Fulfillment of requirements*

Requirement	Category	Description
Future safety	Must have	<b>Good</b> – The technology is based on web technologies and therefore has the same future-safety.
Developers' acceptance (well established & well-known)	Must have	<b>Good</b> – Development is made with web technologies which are accepted by developers.
Support of heterogeneous devices	Must have	<b>Good</b> – Devices which do not use the extended version can still execute the standard web application.
Support of dynamics of devices	Must have	<b>Good</b> – Although the system architecture can still be varied, the technology with client-side application execution allows to join and leave rather easily because its existence does not affect the execution of the overall system as long as it does not provide server functionalities.
Easy set-up of the system (“walk-up-and-use”)	Important	<b>Good</b> – It is up to the users to decide if they want to install an extra application or if they want to interact with the web application without the need for installation.
Scalability and reliability	Important	<b>Good</b> – The arguments for scalability and reliability of standard web technologies apply to extended web technologies as well.
Privacy awareness	Nice to have	<b>Good</b> – It is up to the users to decide if they want to execute the native application (which provides more functionalities but could also access private data) or to use the restrictive web application.
Accessible low-level functionalities	Nice to have	<b>Medium</b> – The extended web technologies allow to access more low-level functionalities (the ones which are provided by the actual extension) than would be possible with standard web technologies executed in a web browser.
Availability of toolkits and frameworks	Nice to have	<b>Good</b> – Toolkits, frameworks and third party libraries of standard web technologies can be reused.

### 3.3.3 Technology decision

Based on the different advantages and disadvantages, the technology has to be chosen very carefully. The comparison matrix in Figure 3.1 shows that several technologies are better suited for collaborative applications than others. While almost all of the different categories have representants which are well-known and well-established, future-safety – although hard to predict – seems to be very unclear for specific web browser plugin technologies (due to the competition with new evolutions of the standard web technologies).

The support of a heterogeneous set of devices is one of the biggest advantages of web technologies while it is very hard to achieve with native programming languages since code has to be manipulated and/or recompiled to support multiple platforms.

Although possible with all types of technology, the support of the dynamics of devices sometimes has to be handled in more complex ways and is not directly supported by the technology itself and might need to be achieved using third party libraries.

The set-up of the system is mainly a problem when using native programming languages since the user has to pre-install the software to be able to interact with the system. Platform independent languages, web browser plugins as well as dynamic languages require pre-installations as well, but since they only need the one-time installation of a common interpreter, the complexity of installation is less of an issue. Web technologies do not need an installation at all, letting the user interact with the system by simply accessing a specific address through a web browser (which is installed in most systems by default).

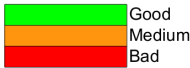
Scalability and reliability depends on the actual realization of the software. Nevertheless, some technologies offer more advanced support for scalable and reliable infrastructures due to their origins (e.g. because web-technologies were built to support the interaction of large amounts of users).

Privacy is not easily controllable in non-web-based applications since they have the potential power to misuse their extended functionality. On the other hand, this is the main disadvantage of web technologies: low-level interfaces, as well as sensors, are not directly accessible by them and therefore possibilities are restricted to a subset of functionalities.

The availability of toolkits and frameworks is given in most cases. Nevertheless, especially for less known or proprietary technologies (such as most web browser plugin technologies), only a few tools exist and which are not easily extensible for the support of new functionalities.



	Native languages	MDA, cross-compile	Platform independent	Web browser plugin	Dynamic languages	Standard web tech.	Extended web tech.
<b>Must have</b>							
Future safety	Good	Good	Good	Bad	Good	Good	Good
Developers' acceptance	Good	Medium	Medium	Medium	Medium	Good	Good
Support of heterogeneous devices	Bad	Medium	Medium	Medium	Medium	Good	Good
Support of dynamics of devices	Medium	Medium	Medium	Good	Medium	Good	Good
<b>Important</b>							
Easy set-up of the system	Bad	Bad	Medium	Medium	Medium	Good	Good
Scalability & reliability	Medium	Medium	Medium	Good	Medium	Good	Good
<b>Nice to have</b>							
Privacy awareness	Bad	Bad	Bad	Good	Medium	Good	Good
Accessible low-level functionalities	Good	Medium	Medium	Medium	Medium	Bad	Medium
Availability of toolkits and frameworks	Good	Medium	Good	Medium	Medium	Good	Good



Good  
 Medium  
 Bad

Figure 3.1.: Comparison matrix for technology candidates

## Decision

Despite some of the restricted possibilities, we decided to use web technologies to develop our toolkit. The main reasons for the choice were the incomparable support for a very heterogeneous set of devices and for a true “walk-up and use” scenario, and the extensibility and future-safety of the technology. Privacy awareness, support and broad acceptance in the developer community, as well as the possibility of lazy code loading also enforced the decision for this technology.

The opinion that web technologies are well suited for collaborative application and could become important for them has come up more often recently (cp. [58], [30]) – especially because of the current evolution of new standards which are supported by most new generation browsers, including those for mobile platforms: *"There is a strong need for better tools in this area – e.g., groupware toolkits that use Web technologies, and development environments for Web applications."* [30]

Since the functionalities as well as the possibilities of optimization are very restricted by the browser sandbox with this technology, ways have to be found and integrated into the toolkit, for nevertheless realizing functionalities which are necessary for collaborative applications. Additionally, although several instances of fundamental support for the mentioned requirements (e.g. code and layout distinction between different devices) exist, clear concepts and structures for how the requirements can be fulfilled are still missing and have to be defined.

Several toolkits exist to support the development of web applications, and most of them have concepts and functionalities necessary for overcoming the differences between the different browser implementations. Others go beyond these simple functionalities and provide even

more complex structures. Most of these toolkits are purely JavaScript based and provide their functionalities as JavaScript libraries (e.g. jQuery). The Google Web Toolkit is a special type of toolkit for web applications – the program code is written in Java and then compiled into optimized JavaScript.

### The Google Web Toolkit and JavaScript

*Due to its non-modular approach, writing, testing and debugging Web applications in Javascript can be a difficult process laden with errors [...]. GWT makes this process less painful by allowing programmers to write applications in traditional Java code, which can then be translated by the GWT compiler into Javascript applications ready for deployment on the Web [...].[58]*

Everything which can be done by the Google Web Toolkit would be possible with handwritten JavaScript as well since it does not add any functionality to the standard web technologies but rather adds structures and supports an improved development process.

Although the main part of JavaScript is standardized as ECMAScript, the interpretation of the code is still different between browsers. Toolkits like GWT help to overcome these issues by compiling Java code and creating different files of JavaScript which are customized for interpretation by the different browsers. Since the differences are handled at compile-time, browsers do not have to download code which is tailored for a different browser, as is often the case in cross-browser libraries. Additionally, bandwidth can be saved and the program code can be optimized.

Since differences between the browser implementations are usually rather small, the Google Web Toolkit contains a mechanism that allows to redefine partial detail implementations which are then recombined in browser specific combinations: If browser A interprets a function differently from browser B, an alternative implementation for that single function can be defined and results in two different JavaScript code files. Browser A will then download only that code file which contains the appropriate code. This way, all functionality which is common to both browsers can be re-used and only special cases have to be handled.

This functionality called “deferred binding” in GWT terminology is not only very well-suited for alternative implementations of standard functions of the JavaScript language but can also be used for the definition of alternative implementations of other application components. Other than the “user.agent” property (and therefore the distinction of different browser types) any other information accessible through JavaScript can be used to separate implementations based on device specific properties (e.g. screen size, touch capabilities, etc.)

Another concept introduced by the Google Web Toolkit is “code splitting” where split points within the standard code flow can be defined. The compiler then splits the code into multiple

JavaScript source files and loads them at the time when the code execution reaches the split point. This functionality not only helps to reduce the overhead of downloading the whole application at start time, but also lets the developer define pieces of code which may not need to be downloaded at all (because the user does not make use of a specific component) and therefore to save resources on the network as well as on the device (because the code does not need to be stored in the cache and/or analyzed by optimizers of the JavaScript engine).

Since code for the Google Web Toolkit is written in Java before being translated to JavaScript by the toolkit itself, the toolkit improves the integration, especially with Java based server backends. Additionally, it allows experienced Java developers to become familiar with the toolkit in short time and therefore increases the number of potential users of the toolkit. Compatibility with standard native JavaScript is achieved by the so called JavaScript Native Interface (JSNI). Therefore, developers can choose if they want to write their code in Java or JavaScript and can profit from the integration of other JavaScript libraries. The well-structured and type safe language Java allows to define a clean interface with good guidance and development rules and is supported by many powerful development environments and tools. Optimizations can be achieved by JavaScript code obfuscation (which usually results in reduced file sizes compared to hand-written code, and therefore optimizes network load) as well as code optimization algorithms, bundling of resources (e.g. composition of images into collections to reduce server round-trips), obfuscation and bundling of CSS files and more.

Since one of the requirements for the toolkit was to not be dependant on a proprietary technology that could be deprecated anytime, it is important to mention, that if GWT disappears the written code could easily be exported in human-readable JavaScript and therefore is not fully dependent on the continuation of the Google Web Toolkit.

The advantages mentioned, and especially the “deferred binding” provided by GWT, which is very well suited for our requirement of device specific adaptations, were the factors that led to our choosing GWT as the base for our extensions for collaborative work. Although it is hard to predict, we feel that there is a good probability that GWT will have longevity since Google itself has based several bigger projects on the technology of GWT and its developer community is constantly increasing.

### **Backend technologies**

For the server backend, web technologies give the freedom of many different technologies which provide the functionalities needed for a web application: PHP, .NET as well as Java to name only a few among a very heterogeneous set. Due to the extensive use of client-side functionalities, modern AJAX applications (e.g. applications written in GWT) relax the functional requirements for the server side. Nevertheless, several communication interfaces

have to be established and especially more complex functionalities that require special resources or more computational power than a browser could provide have to be realized on the server side. Although, potentially, all possible backend technologies could be supported, we have chosen Java for the current implementation. The simplified integration with the GWT code (since both the client and the server side can be written in the same programming language and even share some common code) as well as the platform independence, and therefore the better portability between potential server devices (Java web server containers even exist for smart phones: cp. with iJetty [105]), give an increased flexibility for the deployment. Additionally, external services (such as dynamic servers in the cloud) exist for the Java technology, which supports the choice of a technology that can be applied in many different situations.

### 3.3.4 Implications and challenges

The choice of using web technologies has, as already mentioned, several implications on the capabilities of the resulting toolkit. The cost for platform-heterogeneity and privacy awareness are the restricted functionalities brought on by the sandbox (the web browser) environment that a web application is running in. Although concepts for increasing capabilities using special extensions (e.g. PhoneGap) exist, the main issues – of which some are the main advantages at the same time – are impossible or not easy to overcome.

#### Bi-directional communication

One of the main issues with web technologies is the set of communication standards which is available. Since a web application does not have access to network resources by itself, it is only allowed to use the interfaces which are provided by the browser, and these are very restricted. Before the introduction of web sockets (which is at the moment of writing still a draft specification and not implemented in all major browsers), the only communication protocol accessible by web application were HTTP requests.

Due to the nature of the HTTP protocol, the connection always has to be initiated by the client, meaning that the server has no directly supported way of sending information to the web browser explicitly (“server push”) but instead has to wait until the client has sent the next request to provide the new data. Although several work-arounds arose (under the name “comet” – cp. [30, p. 170]) which vary depending on the capabilities of the web server as well as of the web browser (e.g. long polling, iframe streaming, etc.), some of these connections suffer from rather poor responsiveness and sometimes very inefficient network use. Additionally, the HTTP protocol has (compared to other network protocols) a relatively big overhead because of its header data. Due to these disadvantages, as well as to reduce server

round-trips, best practices appeared (like bundling multiple resources to a single result element, as well as compression of transferred data). Nevertheless, performance is still an issue when trying to achieve bi-directional communication between server and client by means of HTTP.

The technological evolution called “web socket” is an attempt to define a standardized bi-directional communication protocol that reduces the above disadvantages. But, since the specification of this new technology is still in progress and, especially on the back-end side, many possible implementations exist, more abstract toolkits are needed to let the developer ignore all of the existing differences. Luckily, work has been done in this direction and libraries such as the “atmosphere framework” (cp. [95]) which provide an abstract programming interface that negotiates the bi-directional communication (based on the capabilities of the browser and the server), supports web sockets and falls back to the most appropriate comet implementation if needed, have just been released.

Although the evolution of web technologies is moving in the right direction and offering many new opportunities, it is clear that web technologies will probably never allow the fine grained control of network communications that would be possible with native applications (e.g. lacking support of light-weight protocols like UDP).

Although not optimal, Gutwin et al. [30] show that performance of bi-directional communication with web-technologies can be sufficient even for real-time applications if the number of exchanged messages can be kept considerably low (cp. 5.1.1).

### **Browser incompatibilities**

As mentioned before, there exist browser incompatibilities in interpreting HTML, CSS and Javascript. Additionally, many proprietary extensions of these technologies that have appeared are not standardized, and therefore not supported by all browsers. Nevertheless, developers have gotten used to these – often very useful – functionality extensions and applied them as soon as they were integrated into the most common web browsers. Something which can be seen very often for additional functionalities is that, although they provide the same effects, the syntax for the specific element differs (e.g. by introducing browser-specific prefixes for CSS-properties like “-moz-...”) for different browsers.

The developer – again facing the trade-off between extended functionality and standard compatibility (and therefore support of most web browsers) – usually has to be aware of all the different ways of implementation as well as possible fall-back solutions for browsers not supporting the extended functionality, which is quite difficult to achieve. That is why several frameworks and toolkits have been built which try to abstract these incompatibilities for common functionalities and which usually allow developers to react to such different imple-

mentations and manage those which were not captured by them yet.

The Google Web Toolkit provides – in addition to the already presented “deferred binding” concept to react to different JavaScript structures – the functionality of conditional CSS, where different styling configurations can be defined for different web browsers. Since the HTML structure of a GWT application is usually built entirely in JavaScript, HTML incompatibilities between browsers can be handled by the deferred binding mechanism as well.

### **Performance**

Compared to other (compiled) languages, JavaScript interpretation is less performant due to its interpretational part as well as due to its single-threaded nature (although concepts such as “web workers” are coming up but are not yet mature enough to be applied). Nevertheless, a lot of effort has been put into the improvements of the Javascript interpretation engines of all major web browsers to prepare them for new types of web content – so called Rich Internet Applications (RIA) which try to achieve a user experience almost as rich as native applications. Additionally, optimizations can be achieved by writing more efficient code. Code optimizers and code obfuscation help to reduce the load of the application and adaptive implementations (e.g. dynamic manipulations of message sending and processing interval depending on the available resources) help to reduce the amount of code that has to be executed at once.

### **Scalability**

Web applications were originally built to support a very large number of clients. Although extensions such as bi-directional communication reduce the amount of that scalability a little, the technologies themselves still profit from the many established tools to react to a higher demand for resources. Scalability of web applications therefore can be controlled in a fine-grained fashion based on available resources and demand. It either can be run on a single, central web server instance (e.g. on a smart phone or a computer), best suited for private or semi-private scenarios with a restricted number of clients, on a professional hosting solution with managed server management (either a single server or a clustered environment) usually accessible through the web (or in a more complex local area network), or it can even be published in the cloud with dynamic resource aggregation at high demand peaks. Peer2peer situations can be built by having web servers installed on every device involved, while hybrid peer2peer solutions involve only some of the devices which are running a web server and which are communicating directly with each other. The shift of the main part of the functionality to the client side (to the web browser) can improve scalability even further since the potential bottlenecks (the servers) act mainly as code providers and communication gateways, and

therefore do not have to execute complex calculations or other heavy load functionalities.

Web technologies are therefore very adaptive to many different scenarios and can be applied in many different situations without the need for code adaptations.

### **Restricted access to resources**

Web applications cannot directly access hardware (e.g. sensors) as well as the filesystem of a device because of the sandbox they are running in and communication restrictions (e.g. the “same-origin policy” which only allows to communicate with the same host that the current web page originates from) make access to special resources very difficult. Although new, standardized interfaces for hardware access are built as part of HTML5 (e.g. geo-location, video-camera, etc.) so the amount of control will be restricted compared to the native access of such components. Alternatives to provide access to those functionalities are tools like Phonegap – where the user has to install the web application as a native app and is then able to access the components by non-standardized APIs – or the definition of other accessible interfaces (e.g. RESTful) by applying a local “server” on the device which redirects the commands to the component. Independently of how resources are extended, installations of native code – and therefore the lack of true “walk up and use” functionality – is inevitable and new security and privacy issues can arise. Additionally, portability of these functionalities is restricted which is why such work-arounds should be used very cautiously.

## **3.4 System architecture**

The choice of technology as well as its implications and challenges affect the system architecture. Therefore, we will present the types of structures that are possible given our technology choice, how the functionalities are distributed between the client and the server side, and how the communication paths are built between devices as well as software components.

### **3.4.1 Overview**

Since we want to make sure that our applications can run in different environments, we split the functionalities into server and client side components and define which functionalities shall be provided by which part. Depending on the actual system architecture that a specific collaborative setting has, the devices involved have to either be a client, a server or both. In Figure 3.2, the three main types of system architectures are shown, where the different devices involved have different roles. In client-server architectures (A), we have pure clients which are either accessing a local server (which can be a client as well) or they are accessing

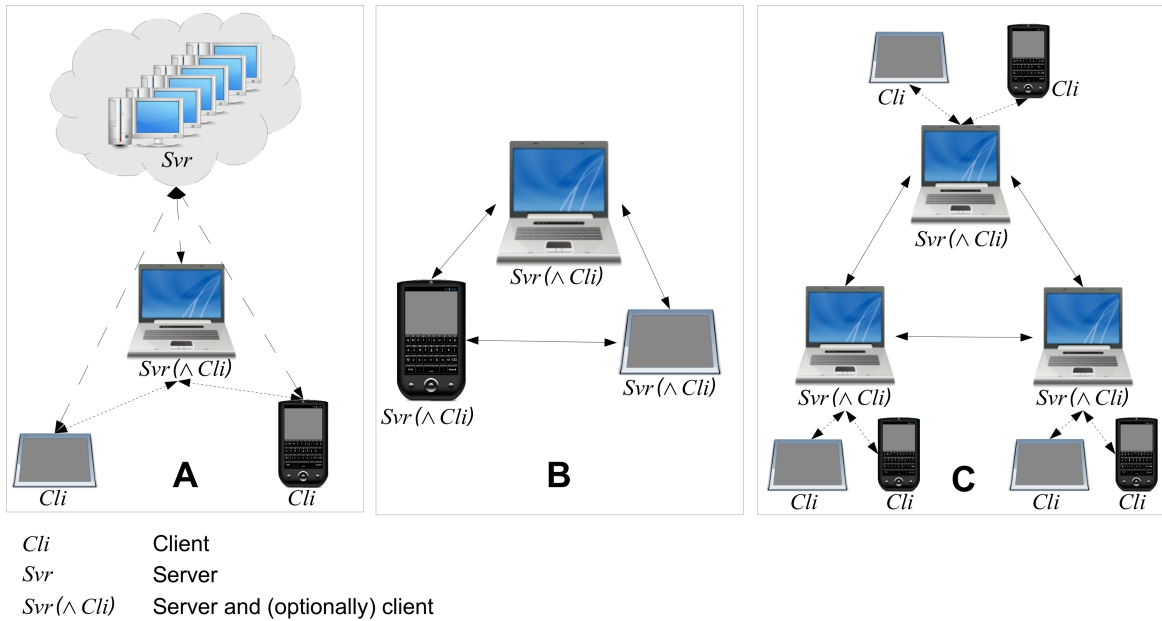


Figure 3.2.: System architectures: Client-server with optional cloud resources (A), peer2peer (B) and hybrid peer2peer (C)

dedicated servers in the cloud.

In pure peer2peer situations (B) all of the devices involved are servers and (usually) clients at the same time, replicating the application state between themselves.

In hybrid peer2peer architectures (C) some dedicated devices are servers (and potentially clients as well) while other devices are clients only, accessing one of the “super-peers”.

While the pure clients do not need to have any installation, and only need to connect to a server through their web browser, the server components have to be installed on a device first. Therefore, architecture B (pure peer2peer) is less suited for a “walk-up and use” scenario than architectures A or C. Additionally, architecture B might not be realizable due to the lack of availability of the back-end technology (in our case Java) for some of the device platforms involved. That is why the probability of architectures A and C is much higher.

Since architecture C (and B) can be created by extending architecture A when integrating additional server instances and establishing synchronization between them by standard means (clustering, distributed databases, etc.), we focus on architecture A in the following sections. Nevertheless, the solutions that we show are equally valid for architectures B and C as well.



### 3.4.2 Work load distribution

Depending on the amount of available server instances, one device might need to handle many clients at the same time. Therefore, the distribution of the work load is essential for preventing breakdown of devices with server functionalities due to a significant load of functionalities they have to fulfill.

The distribution of the functionalities that should be provided to either the client or the server instances always implies several tradeoffs. A server oriented approach, where most or even all of the functionality is executed on the server side, relaxes the constraints on the clients. On the other hand, it increases the requirements for the server instance since that device has to run the application for all of its clients. By putting more functionality onto the client devices, a better distribution of the work load is achieved. Unfortunately, this involves more complex coordination and messaging, and the application performance actually depends on the capabilities of the client.

Especially in ad-hoc collaboration, the server might be a standard device that does not have the same high-performance capabilities as traditional web servers. However, it still has enough power to handle basic server tasks, although some functionalities should then be off-loaded to client devices to lighten the load. Therefore, and since even small and mobile devices provide good enough hardware capabilities to execute more complex applications on their own today, we have decided to move most functionalities to the client side and therefore to relax the requirements for the server side.

Additionally, we want to make sure that the server is very loosely coupled to the clients and therefore should provide its functionalities “stateless” whenever possible, meaning that the functionalities should not need to store client specific information to handle its request. This helps to minimize the overhead and makes the system more flexible in handling appearing and disappearing devices.

### 3.4.3 Server side functionalities

Since the functionalities executed on the server shall be minimized, we have decided for the following general functionalities which must be executed on the server: Application code and resource provider, gateway, external network connection, synchronization, management functionalities and extended functionalities.

### Application code and resource provider

The application code has to be located somewhere, ready to be accessed by the client instances. The provision of the actual JavaScript files and other resources (e.g. images, stylesheets, etc.) is obviously one of the main functionalities of the server side.

Besides the static application code, the server may be used to share dynamic content as well: If a user wants to share a picture, the picture can be uploaded to the server, stored and made accessible for other users.

### Gateway

The server mainly acts as a gateway for the clients. Since web clients cannot (yet – cp. [124]) communicate directly (cp. Figure 3.3), they can use the server functionality to reroute their messages. In addition to simple message passing, this can involve some control mechanism as well (e.g. message-ordering, message-validation, etc.).

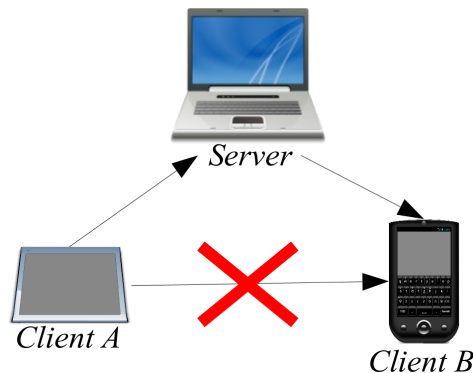


Figure 3.3.: Direct communication between clients is not (yet) possible with web technologies and therefore has to be rerouted by the server

### External network connection

The server might act as a proxy for the access of resources outside of the collaborative setting (e.g. the internet). By acting as a proxy, a server can add functionality to the accessed content (cp. 4.3.5).

### Synchronization

If multiple server instances participate in the same working session, their application state has to be replicated and kept in sync during execution. Additionally, they have to make sure

that the message-passing of the gateway functionality is transferred to the other servers as well. Elaborated mechanisms (such as messaging systems, distributed databases, etc.) which solve the problems mentioned before, as well as clustering functionalities, are available for most of today's web server implementations and therefore can be realized with the support of well established technologies.

### **Management functionalities**

Some information is more easily accessible on the central instance of a server – such as information about devices involved, logged in users, etc. Therefore, some management functionalities have to be provided by the server side such as device grouping, decision about and control of the distribution of user interfaces as well as functionalities for auto-calibration (e.g. the initiation of a architecture change by aggregating or releasing other servers).

### **Extended functionalities**

Some functionalities just cannot be achieved by client side JavaScript and therefore have to be executed on the server. Dynamic image generation and processing, complex calculations and control of external devices are just a few examples.

#### **3.4.4 Client side functionalities**

As mentioned, most of the functionalities should be executed on the client side to achieve a better load balance between the devices. In addition to the actual program logic, the following – more general – functionalities should be under the control of the client devices: Security, layouting, management functionalities (manual configuration) and the execution of the application modules.

### **Security**

Although usually security is a topic for the server side, in ad-hoc scenarios the server might be owned by somebody unknown and therefore not trustworthy. Therefore, sensitive data has to be protected from disclosure to the server side as long as the server itself is not one of the recipients. To ensure these requirements, messages have to be encrypted before they are transmitted and credentials have to be exchanged very carefully.

## Layouting

The distribution of user interfaces requires a very flexible layout on the different devices which is capable of handling appearing as well as disappearing components at runtime. Since this layout depends on the specificities of a device, the concrete layouting has to be processed on the client side.

## Management functionalities (manual configuration)

Other than the overall management functionalities processed on the server side, the user should be able to overrule some decisions like the distribution of user interface components. The client side should therefore provide a generic way of customizing the settings of the available parameters of both the overall application as well as the single components.

## Execution of the application modules

The different application modules have to be instantiated and their life cycle has to be controlled by the main application. Therefore, the client side has to make sure that the modules are loaded, executed and terminated properly.

### 3.4.5 Communication

The communication structure is an important part of the overall system architecture and differs depending on the chosen strategy. While the way of communications between the client and server is rather restricted (because of the limitations of web technologies), communication between servers is less restricted since they have different protocols and mechanisms available. The actual realization of our toolkit therefore focuses mostly on the optimization of the bi-directional communication between clients and servers because server to server communication and the state replication between them is a more common use case and is addressed by already existing solutions (e.g. replicated databases, messaging systems, etc.).

To be able to establish the necessary state replication of real-time applications running on different clients, the establishment of a bi-directional communication between the server and its clients is a strict requirement. Since the type of bi-directional connection (web socket, comet) which can be established depends on the server as well as on the browser, the communication structure has to be negotiated for every client and therefore suffers from varying network latency values.

### **Messaging organization**

Due to the rather restricted data throughput achievable in most bi-directional communication solutions for web technologies, the way in which the messaging is organized has to be considered very carefully.

Since the message channels between the client and server are very restricted, the way in which messages are sent has to be set appropriately. Therefore, the client has to optimize the messages which should be sent out to the server (e.g. by collecting messages and sending them bundled) and organize the handling and interpretation of received messages. Since this functionality is a very essential part of collaborative applications, it should be built into well established and easy to understand components of the framework (cp. 4.2.1).

### **Secure messaging**

To ensure the non-disclosure of messages to non-authorized devices when they are routed through servers which may not be authorized to read them, the messages have to be encrypted on the client side. Therefore, every client has to generate a private-public key pair which allows to either encrypt the message in a device specific way, or to exchange symmetric encryption keys for communication between a group of devices.

### **3.4.6 Set-up**

The set-up of the system depends on the system architecture that will be used. The most simple set-up is the use of a pre-setup system with one or multiple servers either in the local area network (LAN) or in a wide area network (WAN) (e.g. in the cloud).

That way, no action has to be taken by the user on the client and all collaborators can work together simply by connecting to the same network and accessing the same web-page.

For systems without pre-setup servers, at least one of the clients has to install a runtime (in our case any Java Servlet container) as well as the collaborative application. This installation is a standard installation process and should not need special technical knowledge of the user. Additionally, a common network has to be established – either an already existing network can be used (e.g. a publicly available WLAN), a wireless LAN can be set-up by the use of a standard WLAN router or an “ad-hoc”-WiFi network can be set up if supported by the involved devices.

## 3.5 Discussion

Based on the context of our project as well as on the requirements, we want to develop a toolkit which is future safe and which allows to develop applications which are platform independent, scalable, dynamic, easy to set up and controllable by the end-user in terms of access permissions of the application to potentially sensitive data. We have found that all of these specificities can be supported by web technologies and have therefore decided to create our toolkit with this technology, even though it involves several restrictions that have to be overcome.

The toolkit shall – as Roseman and Greenberg have stated – help to let “[...] *the artificial distinction between constructing single and collaborative systems [...] disappear*”[65] and therefore be based on APIs and concepts which are already well established for standard (single-user) application development and – in the best case – the developers will be unaware that they are actually writing a multi-user application unless they have very special needs and want to achieve non-default functionalities.

The toolkit should be extensible to support the manifold needs which will arise in the future, as well as for the easy replacement of single components of the system (e.g. to adapt to different capabilities of devices). Nevertheless, standard functionalities which are used in most collaborative scenarios should be provided by implementations ready to use by end-user applications but also prepared to be extended and optimized by other developers.

Last but not least, the toolkit should ensure that the set up and the use of the system is as simple as possible. Aware that a toolkit for collaborative applications can only be successful if it is not only accepted by the developers but that the developed applications are launchable and accessible in a simple and fast way by end-users (the actual collaborators), tools have to be provided to reduce to a minimum the effort and technical knowledge for accessing a computer supported collaborative session.

# 4

## Toolkit

---

<b>4.1. General concepts</b> . . . . .	<b>74</b>
4.1.1. Modules . . . . .	74
4.1.2. Coding conventions and concepts . . . . .	78
<b>4.2. Basic functionalities</b> . . . . .	<b>81</b>
4.2.1. Distributed eventing mechanism . . . . .	82
4.2.2. Security . . . . .	85
4.2.3. Device grouping . . . . .	87
4.2.4. Layouting . . . . .	87
4.2.5. Multi-user support . . . . .	90
4.2.6. Easy access . . . . .	93
<b>4.3. Software modules</b> . . . . .	<b>94</b>
4.3.1. Drag and drop . . . . .	94
4.3.2. Remote mouse controller . . . . .	96
4.3.3. Remote keyboard . . . . .	98
4.3.4. Extended widgets for multi-user and multi-device contexts . . . . .	99
4.3.5. Collaborative web browsing . . . . .	102
<b>4.4. Use of the toolkit in practice</b> . . . . .	<b>103</b>
4.4.1. Module development . . . . .	104
<b>4.5. Comparison with standard GWT</b> . . . . .	<b>105</b>
<b>4.6. Discussion</b> . . . . .	<b>106</b>

---

After having elaborated the requirements which have to be fulfilled to fill the need for a generic toolkit for collaborative applications, in this chapter we present a concrete realization of such a toolkit. We will show which concepts and structures we have integrated and which issues

(and therefore which basic functionalities) we addressed. We will also show how we have realized different software modules which can be seen as proof of concept implementations and which can act as examples, of how modules for collaborative applications can be extended or created.

## 4.1 General concepts

As mentioned in the specification section, we did not want to build a new toolkit from scratch but rather profit from mature concepts and implementational parts of already existing and well established toolkits, so we chose to base our extensions on GWT. To make the integration of our extensions as smooth as possible, we needed to follow the concepts of GWT itself as well as more general structures and code conventions of Java (since GWT builds on Java). But, we also had to introduce new concepts to cover issues of the multi-user and multi-device scenarios we want to address, and which we tried to keep as close to the existing ones as possible to allow developers to find our extensions easy to learn and to apply.

### 4.1.1 Modules

The requirement for building a toolkit which suits the very different needs that could arise in collaborative applications necessitates the concept of modularity. Functionalities have to be split up into smaller components which are replaceable and dynamically recombable. Therefore, we present how these different modules can be integrated and combined into an actual application, how they can be configured at runtime and how they can be made self-adaptive to the situation at runtime through automatic adaptation of their properties.

#### General module concept

Although the Google Web Toolkit already has a concept of modules, the understanding of these modules in GWT terminology is the simple separation of parts of code into different components which can be combined to reuse the module's functionalities. Since we want to go one step further and let a module not only contain a specific part of code but we also want to have its life cycle under control, and to be able to distribute, start, stop and configure it programmatically the module concept of our toolkit is closer to the life cycle concept of OSGi [113] than of GWT – although its realisation is based on the fundamental structures of GWT modules.

The definition of a module is very lightweight in our case. All that has to be done to create such a component is to implement the `interface TWICEModule<M extends Widget>` where M is the



class of the main widget (which can be any standard GWT widget) which will be attached to the application. The interface defines (besides methods to control the appearance in the final application) two main methods: `void start(M instance)`; which defines what should be executed when the module is started and `void stop(M instance)`; which defines what should be executed when the module is stopped (e.g. freeing memory, stopping regular notifications, etc.).

Additionally, a module can define fields in its implementation which should be configurable by simply adding the annotation `@Configurable` to the field to configure. Thanks to the “deferred binding” mechanism, our toolkit generates the necessary code to provide automatized as well as manual configuration methods (through the layouting system – cp. 4.2.4).

Additionally the toolkit functionality handles the lazy instantiation process of the module (cp. *Resource savings by lazy code loading* in 3.2.2): The implementation of the `TWICEModule<M extends Widget>` is instantiated as a lightweight placeholder filled with the configuration settings. Therefore, it is strongly recommended not to let the widget (M) implement the interface directly but rather to create a standalone class for the `TWICEModule`. The heavy-weight widget (M) with its full implementation is not instantiated (and therefore does not occupy resources) until it has been accessed for the first time. On first access, the lightweight placeholder is invoked, which causes the instantiation of the main widget, its configuration and addition to the layout and therefore its integration into the application. Finally, the start method of the module is invoked, which triggers the execution of the module.

### Device dependent modules through “deferred binding”

One of the main reasons for having a modular approach is the possibility to react to the different devices on which a collaborative application could be executed. Therefore, the different capabilities of the devices involved have to be analyzed and appropriate modules have to be delivered. A device might support multiple different implementations and therefore choose the most suitable one while still offering the alternatives to the user for overruling the system’s choice (e.g. because the alternatives match the needs of the task better, are preferred by the user for personal reasons, or the logic which has to judge the suitability makes a non-optimal decision).

The previously mentioned concept of “deferred binding” in GWT (cp. 3.3.3) was mainly developed to allow the separation of code for the different web browsers to handle the varying interpretation of JavaScript. The fundamental idea of deferred binding is to create multiple versions of the application at compile time while defining some decision logic which chooses which version of the JavaScript code matches with the current client and should be downloaded and executed (in its standard use, the “user.agent” information about the clients browser type and version is used to separate implementations e.g. between Webkit-based

```
1 <define-property name="deviceType" values="cursor,touch" />
2   <property-provider name="deviceType">
3     <![CDATA[
4       if (window.sessionStorage) {
5         var type = window.sessionStorage.getItem('ch.unifr.pai.mice.deviceType');
6         if (type!=null){
7           return type;
8         }
9       }
10      var args = location.search;
11      var start = args.indexOf("deviceType");
12      if (start >= 0) {
13        var value = args.substring(start);
14        var begin = value.indexOf("=") + 1;
15        var end = value.indexOf("&");
16        if (end == -1) {
17          end = value.length;
18        }
19        return value.substring(begin, end);
20      }
21      var ua = window.navigator.userAgent.toLowerCase();
22      if (ua.indexOf("android") != -1){
23        //Firefox Mobile 6 does not respond correctly to the "ontouchevent" therefore, this
24        //is a little hack since not all android devices have to be touch devices – but
25        //most of them are
26        return "touch";
27      }
28      if ("ontouchstart" in window.document.documentElement) {
29        return "touch";
30      }
31      else{
32        return "cursor";
33      }
34    ]>
35  </property-provider>
```

Listing 4.1: Definition of the property for distinction of implementations in deferred binding

browsers, Firefox, Internet Explorer, etc.).

This mechanism is accessible to us and has been reused for other device specific implementations (in addition to making the distinction of the web browser). Since any information which is accessible by JavaScript (e.g. screen size, touch capability, etc.) can be used for the selection of the implementation for a specific interface, this mechanism is perfectly suited for our needs. To give an impression of how such a device specific implementation selection works, we present a concrete (but reduced in complexity) example which distinguishes between the implementation of a mouse controller depending on the capabilities and specificities of a device. Listing 4.1 is an excerpt of a GWT module descriptor file. Here, line 1 declares the property (“deviceType”) that will be used to separate the devices depending on their touch capability. Possible values of this property are “cursor” and “touch”. Lines 2 to 33 declare the logic for how the value for this property can be examined by JavaScript. Lines 4 to 9 check if the value for the property is stored in the HTML5 session storage, if available. If not, lines 10

```
1 <replace-with
  class="ch.unifr.pai.mice.gwt.mouseControl.client.TouchPadCursorWidget">
3 <when-type-is class="ch.unifr.pai.mice.gwt.mouseControl.client.TouchPadWidget" />
</replace-with>
5
6 <replace-with
7 class="ch.unifr.pai.mice.gwt.mouseControl.client.TouchPadMobileWidget">
  <when-type-is class="ch.unifr.pai.mice.gwt.mouseControl.client.TouchPadWidget"
9 />
  <when-property-is name="deviceType" value="touch" />
11 </replace-with
```

Listing 4.2: Use of a property for the choice of a device specific implementation

to 20 check if the value can be found as a simple URL parameter of the web browser. That way, a user can override the detection mechanism by simply adding “deviceType=touch” or “deviceType=cursor” respectively to the requested URL. If the value is still not defined, lines 21 to 25 handle an issue for the mobile version of Firefox which does not respond correctly to the “ontouchevent” and therefore, a work-around has been defined. Lines 26 to 31 then check if the browser supports the “ontouchstart” event, which indicates whether the client is running in a browser with touch capabilities or not. Depending on the outcome of this check, the value “cursor” (for non-touch devices) or “touch” (for touch devices) respectively is returned. Listing 4.2 – again part of a standard GWT module descriptor file – defines in lines 1 to 4 that by default, the interface or class “TouchPadWidget” should be replaced by the “TouchPadCursorWidget” which is a mouse control implementation suited for cursor oriented devices (e.g. standard computers). Lines 6 to 10 on the other hand define that this class shall be replaced with the implementation “TouchPadMobileWidget” if the property “deviceType” (which we have defined above) equals the value “touch”.

The only requirement for having this replacement applied to the code and the implementation to be replaced based on the current mode is to instantiate the component with the factory method `TouchPadWidget widget = GWT.create(TouchPadWidget.class)` instead of the standard method `TouchPadWidget widget = new TouchPadWidget();`. Due to the instantiation of the `TouchPadWidget` by the factory method, the variable `widget` will either be an instance of “TouchPadCursorWidget” or “TouchPadMobileWidget” – depending on the value of the property “deviceType”.

In addition to using the “deferred binding” mechanism for separating implementations of whole modules, it is also possible to apply it for single widgets. Examples can be found in section 4.3.4.

## Module configuration

As mentioned when discussing the general module concept, our toolkit takes care of the technical requirements (the generation of code) to configure the different modules. Therefore, the programming interfaces exist and can be accessed either by a supervisor (an external module that collects information about the current situation and reconfigures the modules accordingly) or by other components – such as generic user interfaces (cp. 4.2.4) which offer configuration dialogs for every component or a general setting screen that allows the user to reset the properties. Interfaces that react to configuration orders from another device in the network would also be imaginable, which would allow to create external controlling instances.

## (Self-)Adaptation for performance optimization

In addition to the adapting to input from the outside (through the configuration interface), modules can adapt themselves as well. They for example can measure the time it takes until they get the response from the server when they have sent out messages and adapt their message update interval accordingly (e.g. if the latency is bigger than the message sending interval, the interval can be decreased and therefore the network load reduced).

### 4.1.2 Coding conventions and concepts

Several coding conventions and concepts already exist due to our choice of technology – others (such as the lifecycle management of the modules) still have to be established. This section presents what already exists and what we have elaborated to create a well defined structure for how applications should be built with our toolkit.

## Reuse of APIs

The reuse of APIs is an essential but not always easily achievable goal. Very often, interfaces exist for specific functionalities – such as drag-and-drop. When extending these functionalities – e.g. to enable multi-user drag-and-drop – the developer (Alice) has the choice to either define her own interfaces which have to be specially handled, or she can try to make her implementation compatible with the already existing APIs. One of the benefits of reusing the already existing APIs is that the developer of an end-user application (Bob) does not need to know about Alice's new drag-and-drop implementation for multi-cursors but only implements against the standard interface. The standard drag and drop implementation can then be replaced by Alice's new implementation for multi-cursor scenarios (e.g. by the “deferred binding” mechanism – cp. 4.1.1) and Bob's application will integrate this new functionality

without changing any piece of code. Although Alice's extension now runs well with the standard toolkit, the existing API might be too restricted (e.g. does not allow Bob to choose the conflict handling strategy if two users want to drag the same object). Therefore, Alice can choose a standard strategy to be applied when her implementation is accessed through the standard API and create another specific interface for her implementation which can be controlled in much more detail if necessary. That way, Bob has the choice to just use the standard functionality (through the standard API) or to make use of the special functionalities that Alice's implementation provides.

This concept of being compatible with already existing APIs helps developers get in touch with the toolkit very quickly and – in the best case – they are completely unaware that their application will be executed in a multi-user context at some point in the future.

### ***Reuse of standard events***

The reuse of API's also means relying on standard events and extending them with the needed functionality. Alice's implementation could introduce her own new multi-pointer mouse events for the drag and drop functionality which would have to be handled specifically – a better way is to reuse the standard mouse events and to simply extend them with the additional information needed which is easily achieved with JavaScript due to its weak typing nature.

In our example, Alice can extend a standard mouse event with information about the device from which it originated (e.g. by adding a device id) to be able to separate events based on the input device that triggers them. A mouse event triggered by the native (directly attached) mouse will not contain the additional information in the event (it will return a "NULL" device id). Due to the special value of the device identifier, it is up to the developer to decide if the event should be treated the same way as a remote pointer or handled separately by explicitly treating it in the event handling logic.

### ***Reduction of differences between single- and multi-user software development***

As a result, the reuse (and extension) of standard events ensures backwards compatibility to the standard (single-pointer) scenario. Because the native mouse behaves exactly the same way as any remote pointer would, it is possible to use the development tools of the existing IDEs to develop the application (in the case of GWT e.g. the "development mode" which allows to execute and debug Java code in the browser without the need of recompilation). The development process therefore stays the same as it would be for a traditional single-user application.

Additionally, chances are good that legacy applications (developed for single-user scenarios) can be migrated (and therefore extended) with minimal or no effort to other scenarios (such

as multi-user support). Since Alice's multi-pointer implementation triggers standard mouse events, legacy applications which have been created for single-pointer scenarios have registered handlers for the same type of events and therefore are executed properly independent of the actual pointer which triggers the event. The non-consideration of the additional device information in the event handling of legacy applications is usually only an issue for complex actions (actions such as drag and drop or gesture recognition may not function correctly) but not for standard behaviors (e.g. mouse down, mouse up, click, etc.).

As shown, the reuse of APIs and concepts of the underlying technology is an essential part of achieving the reduction of complexity for the development of the multi-user scenario, as well as an important step towards a development environment which lets the difference between the development of single- and multi-user scenarios disappear, as has been promoted by Roseman and Greenberg [65].

### **(Automatic) distinction between implementations**

Now that Alice has implemented a new library for multi-cursor drag and drop which is compatible with the standard interfaces, it is still up to Bob to decide if his application should run in multi-pointer or single-pointer mode and therefore which drag and drop implementation to choose. Since Bob is used to implementing web applications for single-user scenarios, he decides to go ahead and implement the single-cursor functionality first. How can Bob's single-user code now easily be transferred to the multi-cursor scenario? The answer lies in the functionality of "deferred binding" – the possibility to replace implementations automatically based on program logic. Bob therefore registers standard handlers for the drag and drop functionality as he is used to doing when developing standard single-user applications and does not care about the actual implementation that realizes the drag and drop functionality. Since Alice has defined a property that tells the application if it is running in multi-pointer or single-pointer mode, and which is used to replace the implementation of the drag and drop functionality automatically, the switch between single- and multi-user mode can be achieved at runtime by setting this specific property (e.g. by definition of a URL parameter, a button inside of the application, etc.).

### **Hidden complexity**

Extended functionalities and complex structures should be hidden in the toolkit as much as possible. In addition to the use of simple interfaces, Java has the concept of annotations which can help indicate special behaviors or specificities of classes, methods and fields. With this functionality, information about how a specific class should be treated by the system can be defined in an easily readable way without cluttering the code with complex instructions or

```

1 <!-- HelloWorld.ui.xml -->
2 <ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'>
3   <div>
4     Hello , <span ui:field='nameSpan' />.
5   </div>
6 </ui:UiBinder>

```

Listing 4.3: Declarative widget description in GWT

```

1 public class HelloWorld {
2   interface MyUiBinder extends UiBinder<DivElement, HelloWorld> {}
3   private static MyUiBinder uiBinder = GWT.create(MyUiBinder.class);
4
5   @UiField SpanElement nameSpan;
6
7   public HelloWorld() {
8     setElement(uiBinder.createAndBindUi(this));
9   }
10
11   public void setName(String name) { nameSpan.setText(name); }
12
13   public void getElement() { return nameSpan; }
14 }

```

Listing 4.4: Binding of the declarative widget description by annotation in GWT

defining of too many interface implementations of a specific class. This concept is – originating from the Java language specification – again a reuse of the GWT standard functionality. The GWT uses annotation based concepts (e.g. in their UIBinding-mechanism) that allow to declare the structure of a graphical widget in a descriptive way (cp. Listing 4.3 from [103]) and to bind elements to Java fields through annotations (cp. Listing 4.4 – line 5, from [103]) in order to be able to access and manipulate them through code. The information defined in classes, methods or fields can be used to generate code based on reflection at compile time during the “deferred binding” phase of the GWT compiler – therefore, complex code mechanisms can be hidden and added at compile time. This helps to simplify and increase readability of the code. Potential uses might be the declaration of the distribution strategies of a specific module (e.g. if a module shall be present on specific device types only), the definition of fields which should be configurable by the user, etc.

## 4.2 Basic functionalities

In addition to these general concepts of how to structure the different components and how to let them react to differences between executing devices, several basic functionalities which are needed for all types of collaborative work have been elaborated.

### 4.2.1 Distributed eventing mechanism

Something that does not exist yet in GWT – since GWT focuses on standard web applications and not necessarily on real-time applications – is a distributed eventing mechanism. To establish such a distributed event mechanism, several issues have to be handled. In addition to establishing a bi-directional communication channel, clock synchronization and different strategies to handle ordering conflicts, how to integrate a distributed eventing mechanism into the toolkit in a simple and API compatible way will be discussed.

#### Bi-directional communication channels

As mentioned before, one of the main disadvantages of web technologies is the lack of proper bi-directional communication between server and client. But, when working with events which need to be exchanged between different devices with minimal latency, bi-directional communication is an absolute necessity.

Besides the recently appearing web sockets, fallback solutions for legacy browsers and web servers exist (“Comet”). Since the different ways of working around the unidirectional restriction of network communication are more or less powerful, a clear hierarchy of preferred solutions exists. Therefore, our toolkit needs to determine the most optimal mechanism for bi-directional communication supported by both the client and the server. Other projects have addressed exactly this issue. They let the client and the server negotiate about the concrete technology that should be used and provide the bi-directional functionality through more abstract interfaces valid for all possible implementations. Although multiple such third-party libraries exist, we have chosen the “atmosphere” [95] library due to its already available interfaces for GWT as well as its use in several larger projects. As with everything in our toolkit, the bi-directional communication mechanism is integrated as a single module and can easily be replaced if another (better) framework is used to establish the connection.

#### Clock synchronization

Clock synchronization can become a rather complex topic – since we are working with potentially very low performance devices, a very light weight although not perfectly precise concept was an appropriate solution in this case. To make sure that events are executed in order, our system lets the application on the client (running in the web browser) guess the current system time of its direct server. This is achieved by sending a request to a method of the server that returns its current system time. The network latency of the request is measured and the server time is adapted by adding half of the request duration. Despite up- and down-streams between client and servers are seldomly symmetrical (meaning that the



request might not be transferred at the same speed as the response) and simplification by ignoring the processing time of the request on the server, the concept of estimating half of the time is also known in other contexts (e.g. in protocols such as NTP) and gives a rather good estimation since variances are usually in regions of few milliseconds. With the information of the current system time of the server, the offset between the server clock and the one of the client can be calculated and when firing events, the timestamps of these events can be adapted to the estimated server system time. This “synchronization” of the system clocks has to be done regularly because the offset of the different system clocks might shift over time.

Other strategies (such as the use of vector clocks) would be possible as well and could replace our very light-weight approach. For our needs, the presented approach works sufficiently well and can therefore be seen as one way to achieve message ordering on the client side.

Since it is possible that some devices have better guesses of the servers system time than others, the events ordered by the timestamp are not guaranteed to represent the actual order of the events that happened in the real world. Nevertheless, our synchronization strategy mechanism allows all devices to interpret the same order and therefore lets them be consistent with one another even if they are not consistent with the real world. Because variations are mostly low values of milliseconds, users will not be able to identify the non-perfect clock-synchronization and the user experience will not be affected.

If multiple servers are involved, the servers have to ensure the appropriate adaptation of the time stamps when rerouting event messages, which can be achieved by standardized clock synchronization protocols.

## Functionality

The Google Web Toolkit contains a local eventing mechanism, the “event bus”, which is a singleton instance and acts, as its name says, as an event bus that lets the different components attach handlers for specific events and to fire events to notify other components about changes.

This well established concept of an event bus – which is used in many other non-GWT technologies as well – has turned out to be quite easy to extend and is therefore very well suited for our needs. We have extended the standard implementation of GWT (`SimpleEventBus`) with remote event capabilities. If a component fires an event that extends our provided `RemoteEvent` class, the event bus serializes the event, enriches it with information such as the originating device and user, the estimated timestamp of the server (cp. *clock synchronization*), the recipients, etc. and sends it to the server. On the other hand, if an event is pushed by the server to the event bus, the event is deserialized and fired in the local event bus controlled

by an exchangeable conflict management strategy. That way, the sending and the receiving of events through the network is fully transparent for the developer as long as such an event bus is used to communicate between the different components (which is good practice in GWT development anyways).

Additionally, to support the different strategies of conflict management, we have created three new subtypes of remote events: `BlockingRemoteEvent`, `UndoableRemoteEvent` and `DiscardingRemoteEvent`.

The `BlockingRemoteEvent` is a very strict event – it blocks the execution of an event until it is sure that no event will arrive and conflict with this event. The processing of this event might be rather slow since the server push has to receive a notification from all of the other devices first, so that they can confirm that they do not have any other potentially conflicting events in their queue. These events should therefore be used very cautiously and only if they are absolutely needed.

An `UndoableRemoteEvent` is an event which can be undone. It is executed immediately and therefore provides very fast responsiveness, but it might conflict with another event that arrives later. Its handler (the `UndoableRemoteEventHandler`) requires – besides the `onEvent` method which handles the arrival of an event – the methods `undo` as well as `saveState`. While `saveState` is executed before `onEvent` and therefore can be used as a hook which allows to save the current state (e.g. of an object), the method `undo` is called when a conflict has appeared and the operation which has been executed `onEvent` has to be undone. This way, the developers can be guided to provide the necessary functionality for undoing events.

A `DiscardingRemoteEvent` is the most responsive and lightweight event. It is executed immediately upon arrival and never rolled back. This is because of its nature, which implies that every newer event fully replaces the former events. One example would be an event that repositions an element. If a delayed event arrives, it does not have any effect on the application state because the element has already been moved to the most current position and therefore the delayed event can be ignored. Although the event history of the different devices might not be the same (because the delayed events have been skipped), they still hold the same current application state.

While every event could be sent as a `BlockingRemoteEvent`, performance would be very bad since the event could only be executed once all devices involved have agreed by sending a confirmation message. `UndoableRemoteEvents` are only possible if all functionalities executed during the event handling are under control and can be rolled back (e.g. those which do not affect third party systems such as credit card transactions). `DiscardingRemoteEvents` are only suited for very specific events, those which mainly provide constant updates of similar properties. It is not possible to ensure that a specific event is ever processed by the recipients since it could have been skipped because of the earlier arrival of a newer event. Additionally, the

`DiscardingRemoteEvents` could – although not affecting the application consistence – influence the user experience because, for example, somebody has pointed with a remote pointer to a specific location on the screen which will not be visible because the repositioning of the pointer has been skipped.

Other events with different handling strategies might easily be added (e.g. such as those that support “operational transformation”) and therefore the eventing mechanism might be customized for the specific needs of a module.

## Use

Remote events can be defined just as standard custom `GWTEvents` as Listing 4.5 shows. The declared variables (`x`, `y` and `blocked`) are serialized automatically by the toolkit. Just as the `SimpleEventBus` of the GWT, the `ServerPushEventBus` of our toolkit can be instantiated either by the standard constructor `new ServerPushEventBus()` or the deferred binding command `GWT.create(ServerPushEventBus.class)`. But, since the event bus should usually only be instantiated once per application (singleton pattern), a convenience interface has been built and the event bus can be accessed through the static method `CommunicationManager.getBidirectionalEventBus()`.

Its use in the actual components is exactly the same as for the standard event bus (cp. Listing 4.6).

## The different eventing strategies

As mentioned in the state of the art section (cp. 2.4), different strategies for handling conflicting events can be applied and vary in their suitability depending on the given application. Our toolkit therefore provides the mentioned differentiation of event handling when working with remote events and different strategies can now be injected into our distributed event bus implementation. More optimistic strategies might result in a more regular execution of the `undo` functionality of an event, while more pessimistic strategies delay the execution (the firing to the local event bus) of the events. This way, the event handling and the complexity of conflict handling is configurable by the developer and can therefore be adapted to the special needs of the current application. More complex implementations would even allow to differentiate the strategies to be applied depending on the actual application specific types of events, which allows a very fine-grained control.

### 4.2.2 Security

Besides the simple distribution of events and messages, the event bus also includes functionalities for securing the data transfer – every client owns a private and a public key. Additionally,

```
public abstract class SomeRemoteEvent extends
2 UndoableRemoteEvent<SomeRemoteEventHandler>{
4     public static final Type<SomeRemoteEventHandler> TYPE = new Type<SomeRemoteEventHandler
        >();
6     public static interface SomeRemoteEventHandler extends UndoableRemoteEventHandler<
        SomeRemoteEvent>{
8     }
10     public Integer x;
10     public Integer y;
12     public Boolean blocked;
12 }
```

Listing 4.5: A custom remote event

```
SomeRemoteEvent event = GWT.create(SomeRemoteEvent.class);
2 CommunicationManager.getBidirectionalEventBus().fireEvent(event);
4 CommunicationManager.getBidirectionalEventBus().addHandler(SomeRemoteEvent.TYPE,
    new SomeRemoteEvent.SomeRemoteEventHandler() {
6
8     @Override
8     public void onEvent(SomeRemoteEvent event) {
10         ...
10     }
12     @Override
12     public void undo(SomeRemoteEvent event) {
14         ...
14     }
16     @Override
16     public void saveState(SomeRemoteEvent event) {
18         ...
18     }
20     }
    });
```

Listing 4.6: Creation of a remote event and firing through the event bus

shared keys for group communication can exist. The server push implementation processes the message before sending it out and encrypts the message content (but not the header so that servers – even if they are not allowed to decrypt the message are able to reroute it to the right recipients). It is also possible to sign the message, to ensure that it is sent by the actual client. That way, encryption, decryption as well as signature checking can be handled within the event bus mechanism and are therefore transparent to the user.

### 4.2.3 Device grouping

If a user owns multiple devices and wants to interact with all of them in the same collaborative session, the system has to be able to merge these devices to a single “meta-device” so that the messages – independent of the device – all originate from the same virtual instance. To achieve such a device grouping, the private credentials have to be shared between the two devices. An initial device has to be selected (the one that appears first in the system) which then creates all its credentials as well as its UUID. As soon as the second device appears, it has to be paired with the first device. Therefore, the device sends a pairing request to the initial device. Some control mechanisms can be established (e.g. a password has to be entered – just as in the bluetooth pairing process) for ensuring that the device that wants to be paired is known by the user. The pairing request is then accepted on the initial device and the UUID as well as the private and public key are transferred to the second device.

The second device now uses the same credentials as the first one and is therefore indistinguishable from it for other devices in the same collaborative session. If the paired devices are unpaired, both of the devices have to create new credentials and therefore to reconnect as if they were new clients in the session.

### 4.2.4 Layouting

The layouting of an application is very important. Modules can be distributed and replicated dynamically on different devices, and they can have different dimensions and different ways of representing visual elements. Therefore, a way has to be found, to adapt the layouting engine which takes care of the representation of the different modules depending on the different components.

Since the way in which visual elements can be represented is very device specific, the layouting module is again device dependent and therefore provides different implementations for different devices. Thus far, we have separated mobile and standard devices so smart phones and tablets have different representations than notebooks or standard personal computers. Further distinctions are well imaginable and part of future work.

## Standard layout

The standard layout for (mainly) cursor oriented devices was inspired by the concept of the graphical user interface of the Eclipse integrated development environment (IDE). Eclipse is based on a plugin structure comparable to the module concept of our toolkit and therefore has to solve the same issues of how to arrange the graphical output of its different components. We have therefore built a layout structure that looks rather similar to the one of Eclipse, including its main functionalities (cp. Figure 4.1). Every instance of a component is

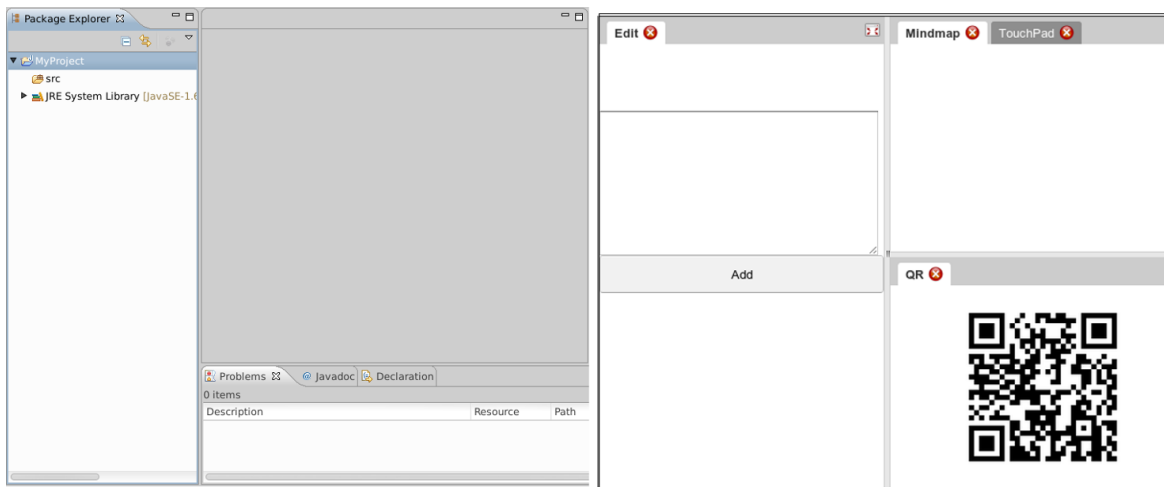


Figure 4.1.: The IDE Eclipse (left) as a template for the design of the dynamic layout for big screen and cursor oriented devices (right).

represented as a tab which can be rearranged by drag and drop. When an element is dragged to a border, a new visual area will be created. The dimensions of the different visible areas are manipulatable by drag and drop as well and therefore, the screen can freely be designed by grouping common elements and splitting the screen into useful sections.

## Mobile layout

A screen of a smart phone or a tablet is usually not big enough to display multiple components side-by-side and therefore the standard layout is usually not appropriate for mobile devices. Inspired by other mobile applications (such as Facebook, Google+, etc.) which have to integrate multiple components into the mobile versions of their applications as well, a design pattern can be extracted which seems to be the current state of the art of mobile user interface design (cp. Figure 4.2). The web application shows a menu button at the top left corner of the screen and presents a single active page in the main area of the screen. When pressing on the menu button, a list of available components appears and lets the user to switch between

the different elements. The idea behind this reuse of well established concepts of layouting is to provide a familiar interface to the user and therefore to establish consistency throughout applications.

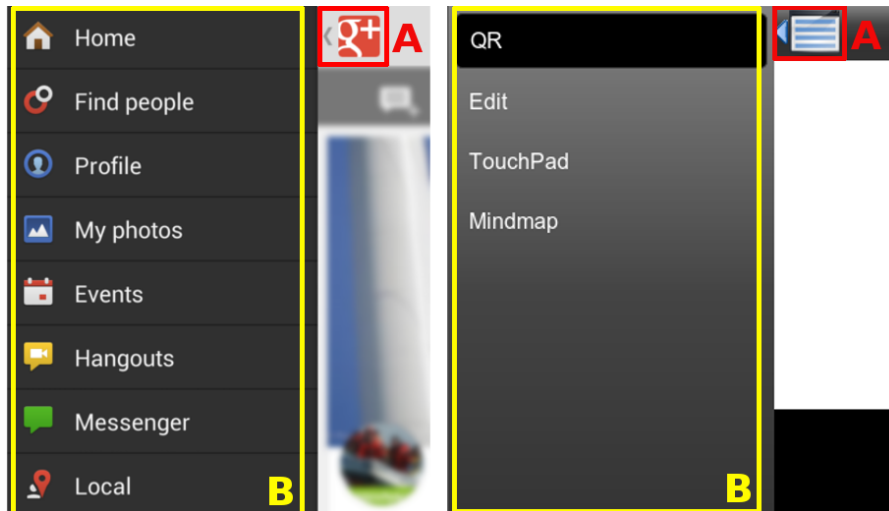


Figure 4.2.: The mobile layout of Google+ (left) and the TWICE toolkit's dynamic layout (right): The menu button (A) is positioned on the top left and when pressed, the menu bar (B) appears from the left.

### Other layouts

In addition to the separation between big screen cursor-oriented and mobile touch based devices, there are other things which need to be taken into account. It could, for example, make sense to allow restricted split screens for tablets (e.g. two components side-by-side) or to make use of the multiple screens of devices such as the Nintendo DS (and in that case maybe even go further by supporting the 3D functionality of the upper screen) – but the separation between the two main types of devices works for most devices and can be seen as a proof of concept of how different layouts can be established depending on different device specificities.

### Automatization of component distribution

Now that we have found ways to represent components appropriately depending on the executing device, the question of how the different components are actually distributed arises. For the moment, we focused on manual distribution (by letting the user select the component(s) to be executed) but all the necessary technology is available for automatic distribution

as well. We have the flexible layouting system and the bidirectional communication channels since there are no technical restrictions to prevent an automatic distribution of components.

### 4.2.5 Multi-user support

For synchronous collaboration, a common use case is the manipulation of elements on a shared screen. Unfortunately the standard graphical toolkits – and therefore web browsers – do not support multiple input devices (e.g. multiple cursors, multiple foci for text input) and since web technologies do not have access to native device information, it makes access to the extended information even harder. We therefore explored different ways of bringing this extra information to the web application. Although our focus was set on multiple mouse pointers as well as on text input, the presented approaches are suitable for other additional information as well.

Besides establishing technical means to overcome the restrictions of a lack of multi-input device support, we also thought about the scalability issues of such a solution (e.g. if too many users want to access the same shared screen) and present our ideas of how to overcome these issues.

#### Getting the extended cursor information

To be able to get extra information about the originating device of, for example, a pointer movement or click, we needed to somehow integrate this additional information into the event flow of the web browser and make it accessible to the web application for further actions. We tested different possibilities and will now show the results, the benefits and the disadvantages of the presented solutions.

##### *First solution: MPX based Webkit browser*

Our first solution was based on the functionality of MPX by Peter Hutterer [37] which was integrated into the X.Org [125] stack and was recently supported by the GTK+ graphical toolkit [100]. We chose to fork the standard webkit library, migrate it to the (at that time) beta version of GTK+-3.0 (including the integration of the basic MPX support) and to pass the additional information down to the actual mouse event which is sent to the web page. Thanks to the dynamicity of JavaScript the additional information of the originating device identifier was added as a non-standard field of the event itself. Thus, we were able to distinguish between the cursors on the JavaScript level and were able to write libraries which were taking this extra information into account (cp. 4.3.1). In addition to the extension of the web browser, we developed a little daemon that listened for the attachment and detachment of pointing devices and integrated them dynamically. This solution turned out to be rather



reliable and – despite working with beta versions – was quite stable. Additionally, all of the devices which are supported by the operating system are immediately supported with this system (e.g. wii-motes, joysticks, etc.). We then developed a RESTful interface which allowed to control mouse cursors through the network. Because of the platform dependency (it only works on Linux) of this solution and the need to install our customized web browser and therefore break the true “walk-up and use” scenario, we have tried to find other ways in which web browsers could be extended.

Additionally, the customized browser involved the issue of maintenance. Since today web browsers have to be updated very regularly (mainly due to security issues), the task of keeping our extended browser up to date would have required a significant effort.

### ***Second solution: Browser plugin***

Therefore, we decided to write a web browser plugin which triggered events by the use of the standard system pointer to the currently executed web page instead. Since the plugin was not directly accessing the native device control of the operating system and we wanted to eliminate the platform-dependency that we had introduced with our dependency on the MPX stack before, we focused on the network-controlled mouse cursors only. Our controller with a RESTful interface was therefore redirecting the commands to the plugin which was then visualizing the mouse pointer as well as firing the events in the web browser. Although this solution now worked in a standard web browser (in our case Firefox) and the task of maintaining the browser itself was released, the solution still was not exactly what we wanted it to be because of its requirement of a plugin-installation.

### ***Third solution: External information mapping***

Our next idea was to map the event with the extra information in an indirect way. Instead of letting the plugin fire the events, a controller with the network interface recorded the incoming events as well as the timestamp, and then repositioned the standard system pointer on the web browser. Since this movement registered by the JavaScript listener of the web application, the web application then asked the controller through another REST-interface for the meta information of the device based on the event properties (in the case of the mouse pointers, the absolute X- and Y-coordinates as well as the timestamp). As long as the controller was installed on the same machine as the web browser, the same system clock was used and since the request speed through the local network interface was fast enough, we achieved a rather good performance and accuracy of the event mappings. The “same origin policy” – which appears if the browser is not running on the same machine as the server of the actual web application – was overcome through the use of JSON-P (cp. [109]). We therefore had a system which only required the availability of the additional component of a

controller that was able to provide a REST-interface and to control the mouse cursor. This now allowed to use any web browser since no plugin was required with this solution anymore. Nevertheless, the installation of the controller on the multi-pointer device was still not as comfortable as it should have been (and did not fulfill the requirement of true “walk-up and use”) – so the final idea was to use web sockets to send the actual mouse events directly to the web browser without any intermediate layer.

#### ***Final solution: Web socket based event simulation***

All the previous solutions had made use of actual native cursor events triggered from a pointing device managed by the operating system. This newest solution on the other hand does not even know the operating system and is therefore completely independent from the platform. Thanks to the chosen technology, it is possible to simulate original events and trigger them through code. This means that mouse events can be sent (although several browser specific issues have to be overcome) all by code without the need of a native mouse event at all. Thanks to this, we were able to overcome the previously needed additional controller which means that we are able to simulate multiple pointers on any modern web browser with web socket support without the need for any installation on that device. In addition to the establishment of a true “walk up and use” environment and even one that includes multi-pointer functionality, we were able to establish a privacy aware mechanism as well. Since the remote mouse pointers do not exist for any other part or application of the device than the actual dedicated shared web application, pointers can never access any object outside of the current browser window. Users can therefore easily provide their devices as shared resources without having to worry about whether remote users will be able to access something other than the actual shared web application. A functionality like this would have been very hard to achieve with any of the other solutions mentioned before.

#### **Expiring pointers**

Now that remote pointers are available in the web browser and can be handled by JavaScript, the question arises of how to manage the usability aspects of multiple mouse pointers interacting with the same resource simultaneously. In a related experiment [46] and [68] executed with our system, it was shown that there is a maximum of 6 pointers which can be visible on a single screen before affecting the performance of the involved users for standard tasks. Since our system allows much more than 6 collaborators to interact at the same time, solutions had to be found that would take this result into account. We therefore decided for a strategy of expiring pointers. Although the number of simultaneous pointers is restricted to 6, these pointers can dynamically be assigned to different users. If Alice does not use her pointer for a specific amount of time, the assignment of the specific pointer to Alice is released and

Bob can request for it. That way, the number of simultaneous interacting users through remote pointers is restricted, but not the number of users in general who can interact with the system.

Although this default restriction is based on the results of the mentioned experiment, the number of pointers is parametrizable, meaning that it can be adapted even at runtime (cp. 5.4).

#### 4.2.6 Easy access

We have seen the main advantages of web technologies for collaborative work, but one thing which is still hard to achieve is to guide users to connect to the correct network and the correct URL to participate in a collaborative session. In the worst case, the users have to log into the defined network by hand and fill in the address bar of their web browsers by typing the sometimes very technical access URLs (especially in local area networks). Although still valid as a fallback solution if no simplification of access can be applied, the access to a collaborative session should be supported by additional technical means. Some of the possible technologies that exist and that have been taken into account for our toolkit will therefore be presented in the following subsections.

##### QR tags

QR tags are visual tags (cp. Figure 4.1 at the bottom right for an example) which contain information that can be interpreted by specialized software. Many users of smart phones or tablets have such an interpreter installed (e.g. Google Goggles) and therefore only have to take a picture of the specific QR tag and the software will take the corresponding actions. QR standards exist, for example, for accessing WLANs or letting the device be redirected to a specific web page. To support this way of directing the user to the correct page, the toolkit contains a standard component which is able to create QR codes dynamically and is therefore able to present them, for example on a shared screen, so that arriving users can connect their devices in a simple and comfortable way.

##### Bluetooth / E-Mail / Short text messages

Another possibility is to notify devices through bluetooth, email, SMS or similar. If a device in the setting is already connected to the session, that device could send the URL by any of the mentioned channels to a device that is not yet connected. That way, information about the WLAN and access URL can be spread without the need to type in these parameters by hand.

### **Landing page of WiFi hotspot**

Some WLAN routers allow to define a landing page to which a user is redirected when trying to access a web page through the wireless LAN the first time (usually for authentication purposes). We have successfully tested this way of easy access with a Linksys WRT54GL WLAN router running a DD-WRT firmware using the “NoCatSplash” functionality.

This configuration is very well suited if the server of a collaborative setting is well known and accessible by a static IP address. Such conditions can be found mostly for static installations (e.g. a meeting room) or if the WLAN router is part of a mobile set up belonging, for example, to a notebook which has the role of a dedicated server (e.g. if a teacher brings his notebook as well as the router to the classroom and lets students connect to the wireless network).

### **Public directories**

If a connection to a public network (e.g. the internet) exists, the information about the session can be published in an online directory. The user then has to access the well-known (and maybe even bookmarked) web page, can search for the collaborative session and then be informed about the connection parameters.

## **4.3 Software modules**

Besides the presented basic functionalities which are required for almost all collaborative sessions, additional software modules have been developed for special purposes – mainly influenced by the needs of our experiments. These functionalities are all available as modules as well and can freely be applied to and integrated with any type of collaborative application.

### **4.3.1 Drag and drop**

Drag and drop is a very well known paradigm to reposition, scale and associate visual objects on a screen. Although HTML5 has started to standardize the use of drag and drop, there are still issues – especially for less common interaction types like our multi-pointer implementation. We therefore have implemented our own drag and drop library supporting most types of interaction modalities – single-pointer, multi-pointer as well as touch.

```
public class DraggableLabel extends Label implements Draggable {  
2  @Override  
  public boolean isDraggable() {  
4    return true;  
  }  
6 }
```

Listing 4.7: An example of a draggable widget

## Functionality

While single-pointer drag and drop is well supported in most browsers and implementations, touch drag and drop has some issues with the distinction between a scrolling action (swiping over the screen) and the actual drag.

While we have solved the issue with touch using a simple distinction of the gestures if a drag gesture starts on a draggable object or not, the multi-pointer support needed more complex management.

As shown in the section about the multi-user support, we were able to add information about the originating device to a mouse event. This information had to be considered within our library because otherwise, events of other pointers (the ones which were not in dragging mode) would influence the drag of the object. For example, if pointer A starts a drag and pointer B causes a “mouse move” event, a standard drag and drop library would reposition the dragged element to the new position of pointer B. Our library on the other hand separates the events depending on their source and therefore only applies events of pointer A to the dragged element.

## Use

When the drag and drop library module is included, the functionality can be used in the following way: The widget to be dragged implements the interface `ch.unifr.pai.mice.gwt.dragNDrop.client.intf.Draggable` (cp. Listing 4.7). The method `isDraggable()` allows to dynamically define at runtime if the element is currently draggable or not (e.g. the value might become false if the widget is locked by another user). The widget then can be made draggable by calling the static method `DragNDrop.makeDraggable` taking the widget as an argument as well as optional configurations (`DragConfiguration`) to be able to define how the widget should react when the drag starts (`onStartDrag`), when it is dropped (`onDrop`) and after it is dropped (`onEndOfDrop`). If we want to define another widget to react to drops and hovering widgets, we can define a `DropTargetHandler` on any widget that implements the standard GWT `HasMouseOverHandlers` (which is almost any widget available) and define what logic should be executed when an element

is hovering (`onHover` and `onHoverEnd`) or dropped above this area (`onDrop`). Besides the dragged widget and the identifier of the executing pointer, the percentage of the intersecting area of the dragged widget with the drop target is given as a parameter. That way, the drop target can react differently if an element is hovering in the area only partially or if it is intersecting fully with the underlying area.

In addition to this basic functionality, many opportunities for detailed configuration exist (e.g. if the widget will be visually moved itself or if a semi-transparent copy will be dragged as a placeholder until the element is dropped, logic for rejecting the drag on a specific drop target, etc.) – for more details see the API documentation in [123].

### Future improvements

Since the standardization of the HTML5 drag and drop events was not very mature and not fully included in GWT at the time of designing the API, the drag and drop module still has its own interfaces which are not directly compatible with the standard APIs that are available today. Nevertheless, an initial proof of concept refactoring has shown that it is possible to transform the APIs of our implementation to support these newly defined standard interfaces. With the new APIs, for example, the widget to be dragged would not have to implement a specific interface (`Draggable`) anymore but could also just define its HTML attribute `draggable` and be draggable by default. Although this does not allow as fine grained control as with our specific interface, a default behaviour can be defined for such elements and therefore compatibility to the default APIs can be guaranteed just as was defined in the *Reuse of APIs* section in 4.1.2.

#### 4.3.2 Remote mouse controller

For controlling the pointers on the shared screen, web based mouse controller clients have had to be developed. Since the way in which the position as well as clicks, drags and drops can be handled differs depending on the type of the executing device as well as its input modalities, multiple implementations had to be realized to support as many devices as possible.

### Functionality

The mouse control is achieved by sending coordinates and/or events such as mouse down, mouse up and click by a simple HTTP request (cp. Figure 4.3). The event is then pushed to the shared device through web socket server push (for the moment we have restricted the functionality to web socket compatible web browsers for performance reasons). To achieve a constant message rate, the client sends out the position information (if changed) in a

configurable interval (80ms by default).

To make use of the different input modalities, we mainly distinguish between mouse and touch oriented devices. For mouse oriented devices, position values of a mouse pointer within a sensitive area are translated to the scale of the shared screen's resolution. If a pointer enters the sensitive area, the mouse events are captured and transmitted to the server. For touch devices, the sensitive area records the touch gestures and calculates the position of the mouse cursor based on the relative movement of the touch. This is then again translated into absolute positions and transmitted to the server for rerouting to the shared screen. We developed additional implementations based on scrolling (e.g. for electronic readers) and with more special strategies that support very special conditions such as the one of the Nintendo DS which has two screens but where only one of those screens is interactive and captures events.

To be able to calculate the absolute position of the pointer in any of the implementations, the resolution of the target screen has to be known. This is achieved by constant feedback about the current screen size transmitted in the response of the update request. In addition to the pixel size, the client device also gets information about the color that has been assigned to its pointer so that that color can be presented as the background of the screen to make users aware of which mouse pointer they are controlling.

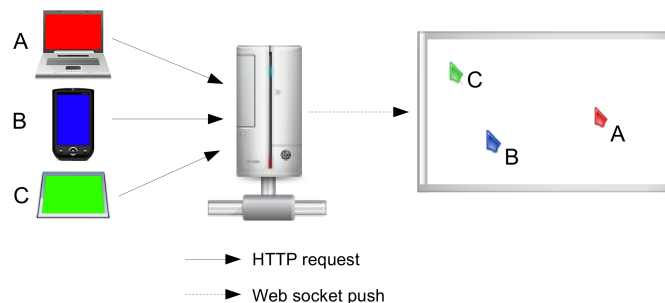


Figure 4.3.: Control of the mouse cursor: A, B and C send the mouse control information to a server which reroutes them to the shared device. Within the response, the clients get information about the pixel size of the target screen and about the color of the pointer which they are controlling

## Use

The mouse control component is a simple toolkit module which can be integrated dynamically into any application. To make a device multi-cursor enabled, the module has to be started in the code by creating a `MultiCursorController` by deferred binding (`GWT.create(MultiCursorController.class)`) and starting it by `controller.start()`. This component is already included in several other

modules (e.g. in the layouting) and therefore usually does not have to be integrated into the application explicitly.

### 4.3.3 Remote keyboard

The remote keyboard – similar to mouse control – allows users to input text on a remote device. Again, different implementations have to be realized – mainly depending on if the device contains a hardware keyboard or if the appearance of a keyboard can be triggered.

#### Functionality

For devices with a hardware keyboard, the realisation of the remote keyboard is rather simple: The component listens to key presses, captures them and sends them to the server where they are – again – pushed to the shared device.

For devices with software keyboards which are not on the screen by default, things are a little harder. Since we do not have direct access to the native functionalities of the device and we are therefore not able to request the software keyboard from the system, we have to work-around it with a little trick. Because all devices trigger the keyboard when a textbox is focused (since the user is expected to enter a text), we have styled a textbox (using CSS) to look like a standard button that has to be pressed when the user wants to show the keyboard. Once the text field is in focus, we can hide it by scaling it down and moving it outside of the visible area. This gives the impression that we have a button to trigger the software keyboard, even though this functionality is not in the scope of web applications. Since we now have focus on a text field, it is enough to listen to the keyboard events arriving in that text field and to send them – again as with the more simple implementation – to the server for further distribution.

#### Use

Again, the keyboard is created as a module and can be integrated whenever needed into a collaborative application. Additionally, the layout module for touch devices by default contains the button that triggers the software keyboard. Although it is possible to integrate the keyboard component as a standalone module, this special handling has been created since the software keyboard does not have a graphical representation (except of the button) and might be more useful in combination with other modules rather than as a standalone functionality.



#### 4.3.4 Extended widgets for multi-user and multi-device contexts

The extension of existing standard HTML widgets (e.g. text boxes, check boxes, buttons, etc.) for the multi-user and multi-device context is a challenge. Because all of these elements are usually provided by the graphical libraries that the web browser is implemented with and therefore do not contain functionalities for multi-user access, we have built work-arounds to establish such functionality which will be presented in the following subsections using the example of a multi-focus text box. Additionally, such widgets are only built to exist once (because of the single-device paradigm) and therefore are not prepared to be duplicated across distributed user interfaces. Therefore, we have developed a synchronization mechanism with which a widget uses the communication module of our toolkit to notify its duplicate about state changes.

##### **Focus widgets**

Focus widgets are widgets which gain the focus of the application when selected (e.g. by a mouse click) and that accept input from a device. Standard widget toolkits usually only allow a single widget to be focused at a time and only by a single input device. This helps to significantly reduce complexity of device management and is fair enough for single-user devices. But, since in our system multiple users can interact with the same device, we had to find ways around this restriction.

##### ***Functionality***

Since web browsers are standard applications, they use normal widget toolkits for their implementations. This implies that the components provided by the web browser for the representation of a web application (such as text boxes, etc.) are standard widgets as well and therefore are only capable of single-focus. To overcome this issue, alternative implementations have to be realized which do not directly use the widget's standard components but rather allow to handle the events (such as focus, blur, etc.) by themselves.

Although potentially possible with legacy HTML components (we have implemented a basic proof of concept text box with multi-focus capabilities), the new features of HTML5 simplify the development of such rather complex widgets a lot. We have therefore decided to focus our main development on the new technologies and to re-build a multi-focus textbox based on a canvas element, although this implies a slight restriction on the supported web browsers (only HTML5 web browsers will be able to execute the “multi-user” functionality for the moment) which are capable of executing these widgets.

In addition to the canvas element, a hidden standard textbox is included which is used as the “backend” of the events. Therefore, all the input information which is captured by our canvas element is rerouted to the standard textbox in order to be backwards compatible with the single-focus implementation and to make sure, that standard event handlers (for the reuse of APIs) can be attached.

In Figure 4.4 the activity diagram of an example use case of the multi-focus implementation is presented. Device A (the red cursor) clicks on the canvas. Based on the click position, the canvas calculates at which position index of the text field the click has arrived and draws a blinking text cursor at the specified position. The component registers the focus of device A for our widget and a focus event is fired on the hidden textbox to which handlers might be registered and which are informed about the focus of the specific element. If device A now sends a keypress event (in our example, the letter “o” has been pressed), the focus registry is aware of the current focused element of device A and redirects the event to the widget. The widget itself adapts its value, shifts the text cursor to the right and redraws itself. After that, it updates the value of the hidden text field to inform registered change-value handlers. The same registration process is applied when device B (the blue cursor) clicks on the text box. Since the text cursor position of device B is smaller than the cursor of device A, the interpretation of the keypress (“l”) of device B affects the position of both text cursors and therefore, both of them are shifted to the right by one position. Finally, device A causes a click outside of the textbox which triggers a blur event on the canvas. The canvas removes the text cursor from itself, unregisters the focus of device A and sends a blur event for the hidden text box.

### *Use*

The use of a multi-focus textbox is handled just like a normal GWT textbox – the only difference is that the textbox should not be instantiated by the default constructor but rather by the deferred binding command `TextBox textbox = GWT.create(TextBox.class)`. Depending on the mode of the device (multi-pointer mode or not), a standard textbox or the extended multi-focus version is instantiated.

The example of the multi-focus textbox can also be seen as a show case of how our toolkit tries to integrate extended functionalities into widgets in a very smooth and almost non-noticeable way. It takes the default widget as the base, including its interfaces, and then extends it with a default configuration for the additional functions. If a developer needs more fine-grained control, it is still possible to cast the widget to the actual class and manipulate it through extended APIs (e.g. `((MultiFocusTextBox)textbox).setMaxUsers(3);`).

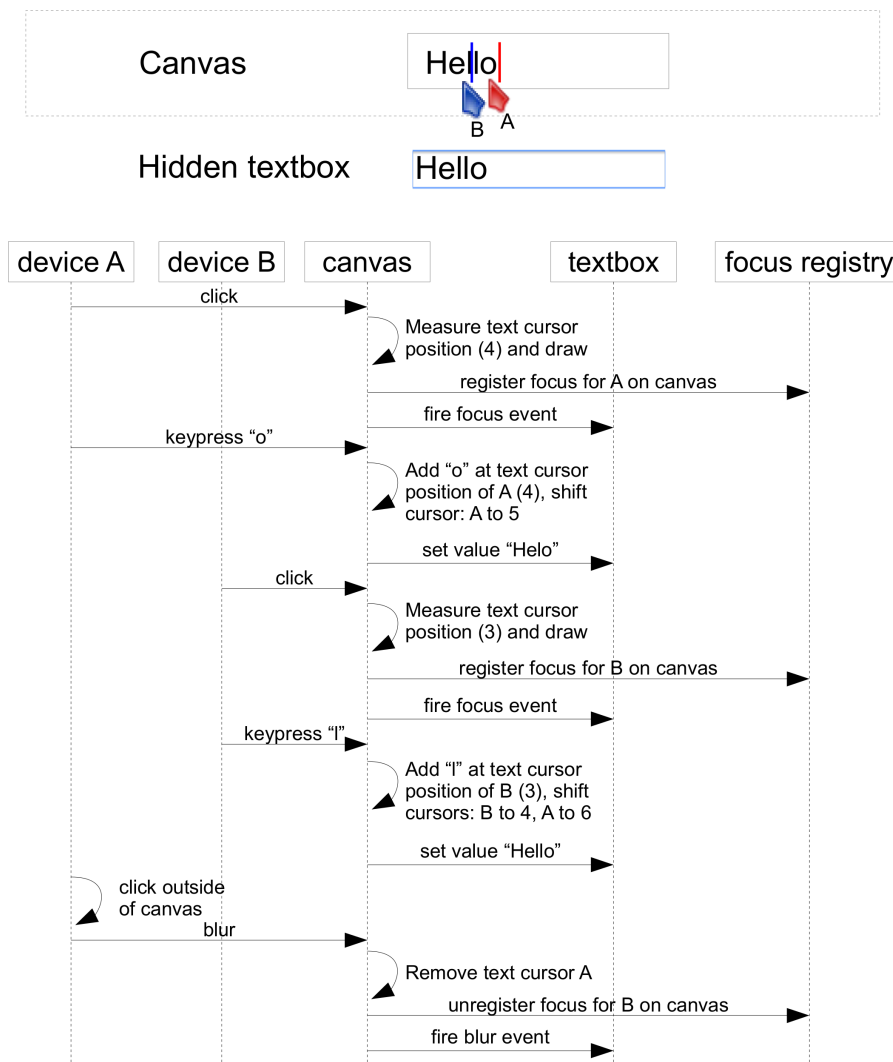


Figure 4.4.: Activity diagram of a multi-focus textbox with two devices (A=red and B=blue)

### Self synchronizing widgets

Self synchronizing – or “remote” – widgets are widgets which automatically synchronize their content within the same working session through the event bus mechanism. One example would be if two devices run the same application module which includes a self synchronizing text box, the value of the text box will immediately be updated on device B if it has been manipulated on device A.

### **Functionality**

As mentioned before, synchronized widgets rely on the general eventing mechanism provided by the toolkit. The widgets (e.g. a text box) fire change events to the event bus and register a handler for these types of events at the same time. After filtering out events that originated in itself, the widget updates its value according to the received messages from other devices.

### **Use**

Again, these kinds of widgets are special implementations of the standard widget and can either be accessed through the deferred binding mechanism (e.g. `GWT.create(TextBox.class)`) or explicitly by standard instantiation with the available constructor (e.g. `new RemoteTextBox()`).

## **4.3.5 Collaborative web browsing**

A more advanced widget that we have developed is a collaborative web browsing frame. This collaborative web browsing frame extends third party web pages (e.g. `www.google.ch`) with multi-user (or other types of) functionalities. A detailed description of this module can be found in [68]. Since these web pages are not under our control, we had to find ways to extend the original pages in a legal and generalizable way.

### **Architecture**

Our solution works with a forward proxy approach. The server contains a proxy servlet through which clients can access external web pages. The url, which is requested by the clients, is therefore adapted from e.g. `http://www.google.ch` to `http://proxyserver/http://www.google.ch`. The proxy server not only executes the requests on behalf of the client but can also adapt the response before returning it to the client. This possibility of manipulating the response allows us to inject JavaScript which extends the functionality of the original web page even though we do not have that page under control, and we did not have to find ways to take over control by other (mostly illegal and non-generalizable) means (e.g. cross-site scripting).

A forward proxy has several advantages compared to a reverse proxy<sup>1</sup> such as the independence from configuration and no requirements for predefined infrastructures. But, there are some disadvantages as well. Since web pages usually contain hyperlinks which point to some other addresses, the loss of control is a serious issue. If somebody clicks on a non-manipulated link, the client will be navigated to the original page without the prefix of the proxy server in the URL. As soon as a user navigates away from the proxy server, control of this page is lost

---

<sup>1</sup>A reverse proxy is transparent to the client: The client still accesses the original URL (e.g. `http://www.google.ch`) but, due to a configurational setting or by the design of the infrastructure, the request is passed through a proxy server as well.

and it cannot be regained automatically. Therefore, the proxy server has to make sure that all URLs presented on a web page are manipulated and replaced by adding the proxy-prefix to the target URL. But, since many web-pages these days use AJAX and links might be created dynamically using JavaScript, it is not enough just to manipulate these URLs on the server side while processing the response. We therefore have implemented a controlling JavaScript which is injected into the response for the client and which listens for any changes in the DOM structure. If a link is generated or manipulated by a third party JavaScript, our logic makes sure that the URL is rewritten immediately.

### Privacy and security

The forward proxy solution also implies some issues related to privacy and security. Since the proxy server has the possibility to transform the response of the original page, it would also be possible to misuse this kind of functionality. Therefore, users should be advised not to execute security and privacy sensitive actions through the proxy server. Since we do not expect privacy sensitive actions to be executed on a shared device anyway, and since our solution allows to make use of shared and private browsing sessions simultaneously side-by-side (e.g. one frame or tab of the browser accessing a web page through the proxy for the shared scenario and one accessing the web page with its original URL and therefore in a secure way for private use), we do not consider this restriction as a serious issue as long as the user is made aware of the potential risk.

### Use

The collaborative web browsing component is a standard module and can therefore be included as an element into a collaborative application. The implementation shows how server side resources can be used to extend information or external resources for collaborative use and therefore shall be seen as a more complex example of a customized functionality extension.

## 4.4 Use of the toolkit in practice

When using the toolkit, a developer can access the different projects, which are organized as Maven [110] packages. Depending on the developer's needs, not all modules have to be used and some can even be replaced. Nevertheless, there are several modules which form the foundation of a standard collaborative application developed with our toolkit and therefore, the standard way of developing a collaborative applications will be presented in this section.

```
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.4.0//EN" "http://google-  
web-toolkit.googlecode.com/svn/tags/2.4.0/distro-source/core/src/gwt-module.dtd">  
2 <module>  
  <inherits name="com.google.gwt.user.User" />  
4 <source path="client"/>  
</module>
```

Listing 4.8: A GWT module descriptor for libraries

### 4.4.1 Module development

For the development of collaborative applications with the toolkit, two different types of components exist: executable applications and libraries. The developer can freely choose where to define the modules (cp. 4.1.1) and could either implement them in one single but very large application project or split them up into multiple projects, providing the modules as libraries. For the re-use of modules, we highly recommend the idea of splitting modules up into separate projects to allow a modular integration of the different functionalities. Obviously, closely related modules might be defined within a single development project – but granularity should still be kept reasonably fine.

An application project differs from a library project mainly because the application actually contains a graphical interface while the library does not. The different projects, as well as their dependencies, are managed and organized by Maven. Although it is not absolutely necessary, users of the toolkit are advised to continue to use Maven, especially because it is a defacto standard of today's Java development.

#### Writing a library

For writing libraries, experience has shown that it is good practice to create two separate projects for the library. One GWT project without `EntryPoint` (cp. Listing 4.8), producing a JAR file including the source code of the classes which will be translated into JavaScript (cp. Listing B.1) and a “testing-application” with a dependency on the library and which implements test scenarios and/or show cases for the use of the library. This testing-application serves not only as an example for other developers to see the functionalities, purposes and uses of the library but also acts as a test bed for the actual development.

#### Writing an application

To write an actual application which can be deployed and executed is almost identical to writing a standard GWT application. The project to create is a standard GWT project (again using Maven for dependency resolution – cp. Listing B.2) which has dependencies

on the different modules which are required for the application. Not mandatory, but recommended, are the modules *DynamicLayout* (for the provision of the device specific layouts and lifecycle management of the modules), *ServerPush* (the communication structure including the distributed event bus and security mechanisms) and *DragNDrop* (the implementation of multi-pointer and touch drag and drop – this module is already part of *DynamicLayout*).

The dependencies not only need to be defined in the Maven-POM file but in the GWT.xml-descriptor file as well (as is usual in GWT).

When using the *DynamicLayout*, modules that will be shown as part of the application can be added by calling the method `public <W extends Widget> void addComponent(String name, Object component, AsyncCallback<W> callback)`. Here, the name of the component can be defined (the label that is used, for example, in the menu or as the header of the tab), the second argument is the instantiated module (by deferred binding: e.g. `GWT.create(TouchPadModule.class)`) and the third is a callback method which is invoked at the moment when the actual module is instantiated.

## 4.5 Comparison with standard GWT

In this section, we want to summarize the components that we added in our toolkit to standard GWT, and therefore give an overview of the extensions available to the developer.

### Extended concepts of GWT

Concept	Descripton
Modules	Extension of the standard GWT module concept of simple code separation to support life cycle management of the different components which can be combined into a full application (cp. 4.1.1).
Reuse of APIs	Enforcement of the good practice to reuse standard APIs as well as standard events while extending them to the additional needs (cp. 4.1.2).
Deferred binding	Extension of the standard use of deferred binding (the separation of different web browser implementations) by the introduction of separation logic based on input modalities and other device specificities (cp. 4.1.1).
Hiding complexity	Extension of the use of annotations to hide the complexity of configurable module parameters (cp. 4.1.2).

### Newly introduced components and concepts by TWICE

Component	Description
Device grouping	The technical possibility to group devices and let them appear as one device for multi-device per user use (cp. 4.2.3)
Layouting and component distribution	Different dynamic layout implementations depending on the screen size as well as the input modalities of a device (cp. 4.2.4)
Multi pointer support	A solution to integrate multiple pointers on a shared device based on pure web technologies triggering standard mouse events (cp. 4.2.5).
Easy access	Tools to simplify the actions for an end-user to connect to an already existing collaborative setting (cp. 4.2.6).
Remote controllers	Remote control widgets for mouse pointers as well as for the insertion of text. Different input modalities are handled through different specific implementations (e.g. “touchpad” for touch devices, pointer position translation for cursor-oriented devices) – cp. 4.3.2 and 4.3.3.
Collaborative web browsing	A component that allows – based on a forward proxy mechanism – to extend legacy and third-party web pages and web applications with multi-user functionalities (cp. 4.3.5).

### Extended components of GWT

GWT	TWICE	Description	GWT use	TWICE use
EventBus	ServerPushEventBus	Extension of the standard event bus for remote communication functionalities (incl. the introduction of remote events, security, clock synchronization and conflict management strategies) – cp. 4.2.1.	new SimpleEventBus()	new ServerPushEventBus() or GWT.create(EventBus.class)
HasDragEndHandlers HasDragEnterHandlers HasDragHandlers HasDragLeaveHandlers HasDragOverHandlers HasDragStartHandlers HasDropHandlers	DragNDrop DragNDropHandler DropTargetHandler	Multi-pointer aware drag and drop library (not yet standard API compatible – cp. 4.3.1).	addDropHandler setDraggable ...	DragNDrop. setDropHandler DragNDrop. makeDraggable ...
TextBox	MultiFocusTextBox	Textbox with multi-focus capabilities (based on the canvas element and therefore only works with HTML5 browsers – cp. 4.3.4).	new TextBox() or GWT.create(TextBox.class)	new MultiFocusTextBox() or GWT.create(TextBox.class)
TextBox	RemoteTextBox	Textbox with self-synchronizing capabilities – cp. 4.3.4).	new TextBox() or GWT.create(TextBox.class)	new RemoteTextBox() or GWT.create(TextBox.class)

## 4.6 Discussion

The main requirements defined in 3.2 have all been addressed with our toolkit – the provision of simple APIs and the reuse of concepts, the modularity as well as the extensibility of the toolkit components, lazy code loading to save resources, device-specific code adaptation



---

by design, the establishment of bi-directional communication channels including tools for network-traffic optimization that can be used for messaging and remote eventing mechanism, the establishment of application level security, components for multi-user support, the grouping of devices for handling multi-devices per user scenarios as well as the distribution of user interfaces to the different devices. Some of the realizations and implementations are more advanced (e.g. the dynamic layouting, the multi-user support) than others (e.g. messaging and eventing), mostly due to the manifold approaches of how a single issue of collaborative applications can be addressed.

Nevertheless, we were able to show how all of the mentioned requirements can be addressed and therefore how a solid and extensible toolkit (TWICE) can be built which is able to hide the complexity of distributed and multi-user oriented collaborative applications, especially by the reuse of programming interfaces and the functionality of dynamic replacement of partial implementations (modules). Even though our toolkit hides the complexity as much as possible and tries to help developers of single-user applications feel comfortable using it, the detailed control of special (e.g. device specific) functionalities is not hidden. Extended APIs for more detailed control are accessible to developers, allowing them to develop more complex applications when needed.

# 5

## Evaluation and real world use

---

<b>5.1. Technical evaluation</b>	<b>109</b>
5.1.1. Performance	109
5.1.2. Heterogeneity	114
5.1.3. Scalability	115
<b>5.2. In-use evaluation</b>	<b>116</b>
5.2.1. Initial experiments	116
5.2.2. A Fitt of distraction	118
5.2.3. Distributed user interface experiments	119
5.2.4. Computer supported brain storming	121
5.2.5. Usability experiments	122
5.2.6. Multi-Zoom	123
<b>5.3. A modular mindmap application</b>	<b>124</b>
5.3.1. The latest version of the mindmap application	125
5.3.2. Non-integrated components	127
5.3.3. Modularity in practice	128
<b>5.4. Real-world experiment: Use in an educational scenario</b>	<b>128</b>
<b>5.5. Developer evaluation of the toolkit</b>	<b>133</b>
<b>5.6. Discussion</b>	<b>136</b>

---

To confirm that the TWICE toolkit's defined requirements have been fulfilled and that the architecture as well as the choice of technology and concepts are valid, we have performed different types of evaluations. In addition to the technical evaluations which show the technical viability of the application development with the toolkit, the toolkit has been used to develop multiple real scenarios and applications proving its effectiveness and reliability. By

observing and interviewing other developers we have tried to evaluate the additional complexity introduced by our toolkit compared to the development of single-user standard web applications using the Google Web Toolkit. Additionally, a more complex show case application is presented which shows how the different components are integrated with each other and how they interact. To ensure that applications developed with our toolkit can also work in real word scenarios, we have applied our show case application in an educational setting to validate end-user acceptance of the system.

## 5.1 Technical evaluation

The technical evaluation of the system is based on experimental testing of different aspects of the system. We want to show that the chosen technology is capable of providing the required functionalities and fulfills the main requirements defined in section 3.2.

### 5.1.1 Performance

The performance of the system depends on multiple factors. The most important ones are: network latency, performance of the JavaScript engine and device capabilities.

#### Network latency

Gutwin et al. show in [30] that usual message rates used for collaborative applications can be handled even by “legacy” network connections (XHR by HTTP POST), especially in local area networks, while the new web socket concept outperforms even plug-in based approaches. For communication from the browser to the server, they used messages with 500 byte payloads (without header overhead) and achieved more than 1000 delivered messages per second with the slowest technology in LAN conditions and 19 delivered messages per second through a wide area network (WAN). One of the most frequently updating components in our toolkit is the remote mouse controller which checks for position changes every 80 milliseconds by default and therefore sends a maximum of 12.5 messages per second. The payload of such mouse position updates is 16 bytes and therefore far below the payload used in the tests of Gutwin et al.

For message rates in the other direction (from the server to the web browser), the message throughput per second depends on the available technology. While most of the tested technologies in [30] achieve a transfer of more than 300 messages per second even in WAN conditions, the long polling technology drops to 20 messages per second over a WAN. Since in a collaborative, distributed system more messages usually arrive at a device than the de-

vice broadcasts (because there is a theoretically infinite number of senders), this could be a serious restriction and therefore devices which only support long polling should not execute components which consume a large amount of events, but rather should mainly be used as data providers with mostly outgoing data transfer.

Additionally, if performance drops – e.g. because of a general overload on the network, poor performance of the JavaScript engine or capabilities that are lacking in the device – a dynamic adaptation of message update rates is possible. Our research group has already started projects to provide means for an add-on to the toolkit’s message system which adapts itself depending on the devices as well as the network conditions, and therefore helps to reduce the overall load for the system.

To conclude, we have seen that all network technologies tested by Gutwin et al. perform well enough to support good network throughput in local area networks and most of them are even applicable for metropolitan area networks (MANs) and WANs. Since our system mainly focuses on colocated scenarios, the use in LANs is the most common use case and we therefore do not expect to face serious network issues. If the system is applied to remote scenarios on the other hand, messaging would need to be reconsidered and more static structures could be introduced (e.g. one server at each location acting as routers and using more performant network protocols).

### **Network throughput on a shared WLAN**

Another restriction is the available bandwidth. If, for example, all colocated clients are interconnected through the same wireless network, they share the provided bandwidth which might vary a lot depending on the characteristics of the physical environment. Nevertheless, the messages exchanged in collaborative environments are usually rather small in size (except for audio and video) and therefore do not consume a lot of bandwidth. Our example of the remote mouse controller shows that a message with 16 bytes of payload ends up as 119 bytes in total (including message overhead and response). At a maximum transfer of 12.5 messages per second, this results in about 1.5 kilobytes (or 12 kilobit/s) of data transfer per second and client.

In [19], average throughput for TCP of a standard wireless LAN has been found to be (depending on the encryption standard) above 13Mbps (or 1664 KB/s). In such a scenario, with the theoretical assumption that no other data transfer is processed at that time, more than 1000 client devices would be able to control the mouse cursors at the same time. Although the scenario of having a network load of only the remote cursors is rather unrealistic and the actual transfer rates vary depending on many different factors, we can conclude that (for standard text messages) the provided bandwidth of a standard wireless network should be

fast enough in most realistic scenarios.

To get an idea of the system performance in a real use case, an evaluation was performed with a school class of 13 students and one teacher (cp. 5.4) which showed that even with a single standard low-cost wireless LAN router, a system can be built that performs well enough for realistic simultaneous use.

### Performance of JavaScript engines and device capabilities

Because our toolkit is mainly based on the execution of code on the client side, the execution performance of our system relies mainly – in addition to the network performance – on the capabilities of JavaScript. This JavaScript performance depends on the actual implementation of the web browser as well as on the capabilities of the device (processor speed, RAM, etc.) that the browser is running on. Because of this device-dependence, general statements about the performance of the overall system are very hard to make and should be differentiated depending on the devices involved. Additionally, web browser implementations have different foci (e.g. string operations, encryption, bit operations, regular expressions, etc.) and therefore measurement and comparison of such performance values is even more difficult.

One of the most widely accepted tools to compare the overall performance of web browsers and their JavaScript engine is the SunSpider JavaScript Benchmark (cp. [121]) which balances the different functionalities of the JavaScript engines and reruns the tests multiple times to provide information about variances during the run of the benchmark (due to other background processes running on the same device which can influence the performance at a specific time).

Although widely accepted to be one of the most appropriate benchmarks, SunSpider still only considers parts of the set of functionalities that a JavaScript engine provides and only gives an idea of differences in performance when comparing different device-browser combinations.

Table 5.1 shows the test results (smaller values are better) of the SunSpider JavaScript benchmark of selected devices and device types. We have mainly focused on representing typical device types in collaborative settings (Desktop-PC, Notebook, Tablet, Smart phone, e-Reader and handheld game consoles) and have benchmarked their combinations with the most common web browsers. The results show that we can basically split the device-browser combinations into three main categories: High performance ( $<500\text{ms}$ ), medium performance ( $<10000\text{ms}$ ) and low performance ( $>10000\text{ms}$ ):

As expected, all **high-performance** devices are either notebooks or desktop PCs. In our experiments, we have seen that these types of devices are ready to properly execute main

	Device	Device type	Browser	Total value
high	HP Pavilion dv6 <sup>1</sup>	Notebook	Internet Explorer 9	199.9ms +/-1.3%
	Dell XPS 420 <sup>2</sup>	Desktop-PC	Chrome 23.0	247.5ms +/-1.3%
	HP Compaq 6730b <sup>3</sup>	Notebook	Firefox 15.0	253.0ms +/-2.5%
	HP Pavilion dv6 <sup>1</sup>	Notebook	Chrome 23.0	260.7ms +/-6.0%
	Dell XPS 420 <sup>2</sup>	Desktop-PC	Firefox 17.0.1	293.0ms +/-1.2%
	HP Compaq 6730b <sup>3</sup>	Notebook	Chrome 18.0	297.0ms +/-0.9%
	Dell XPS 420 <sup>2</sup>	Desktop-PC	Chrome 10.0	303.0ms +/-1.2%
medium	Samsung Galaxy Note 10.1	Tablet	Default browser 4.0.4	1173.8ms +/-0.7%
	Samsung Galaxy S3	Smart phone	Default browser 4.1.2	1220.1ms +/-2.8%
	Samsung Galaxy Note 10.1	Tablet	Firefox 17.0	1331.3ms +/-2.2%
	Samsung Galaxy Note 10.1	Tablet	Chrome 18.0	1351.2ms +/-1.5%
	Samsung Galaxy S3	Smart phone	Chrome 18.0	1376.5ms +/-2.1%
	Samsung Galaxy S3	Smart phone	Firefox 17.0	1400.1ms +/-4.5%
	Apple iPhone 4S	Smart phone	Safari	1771.1ms +/-0.4%
	Apple iPad 2	Tablet	Safari	1812.3ms +/-0.4%
	Dell XPS 420 <sup>2</sup>	Desktop-PC	Firefox 3.6.1	2051.0ms +/-2.3%
	Apple iPad	Tablet	Safari	3342.3 +/-1.5%
	Apple iPhone 3GS	Smart phone	Safari	4671.2ms +/-0.4%
HTC Desire	Smart phone	Default browser (2.2.2)	5747.2ms +/-3.9%	
low	Bookeen Odyssey <sup>4</sup>	e-Reader	Default browser	19245.0ms <sup>5</sup>
	Apple iPhone 3G	Smart phone	Safari	31949.4ms +/-0.7%
	Nintendo 3DS	Handheld game console	Default browser	104366.7ms +/-0.4%
	OYO	e-Reader	Default browser	117964.0ms <sup>5</sup>
	Sony PSP-1000 (6.60-ME)	Handheld game console	Default browser	not executable

<sup>1</sup>4GB RAM, Intel Core i5-2410M (2.3GHz), running Windows 7 Home Premium 64-Bit SP1

<sup>2</sup>4GB RAM, Intel Core2 Quad CPU (4x 2.4GHz), running Ubuntu Linux

<sup>3</sup>4GB RAM, Intel Core2 Duo CPU P8800 (2x 2.66GHz), running Ubuntu Linux with Kernel 3.2.0-30

<sup>4</sup>Cybook Odyssey HD Frontlight, 128MB RAM, Cortex A8 OMAP3611 (800MHz)

<sup>5</sup>This test was only executed once instead of the repeated execution of the tests for reducing variance as it is standard for SunSpider. This was necessary because the crash detection mechanism of the device did not allow us to execute the full test. The measured value nevertheless gives an idea about the approximate performance of the device.

Table 5.1.: SunSpider JavaScript benchmark results of selected devices (total values – lower values are better)

functionalities of a collaborative setting such as control of the shared screen, multi-pointer visualization, etc.

The **medium-performance** category is dominated by tablets and smart phones. We have found these types of devices to be capable of easily handling one or more components at the same time. The more powerful ones were even able to handle multi-pointers (successfully tested e.g. with the Samsung Galaxy Note 10.1 tablet) although they were noticeably slower than their high-performance counterparts. Devices of this type are typically used as powerful controller devices (e.g. for mouse pointer control, text entry, displays of extended or personal information, etc.). Components which make reduced use of JavaScript functionality but contain rather static elements with regular updates have been found to be the most appropriate for these devices. Interestingly enough, the measurement of the desktop PC running the rather old Firefox version 3.6.1 was categorized as a medium-performance device. When this combination is compared to other (more modern) browser implementations running on the exact same machine (which are some of the combinations with the highest performance in this set of measurements), we very clearly see that performance is not only defined by the hardware capabilities of the device, and that the maturity of the browser implementation plays an even more important role.

The devices tested in the **low-performance** category were either standard e-readers or handheld game consoles. This is not surprising since the web browsing functionalities are – compared to the other device types – not part of the main functionalities of the device and therefore not the main focus of improvements and development. Nevertheless, it is worth noticing, that the values of the two e-readers differ enormously. The value of the newer device (the Bookeen Odyssey) is less than a sixth of the value of the older one (OYO) and therefore we can see that even in this area, new device generations are going to improve performance of their web browsers and it seems to only be a question of time before one of these device types moves into the medium-performance category. The devices in this category have been integrated into our testing environment mostly as a proof of concept. Their low performance made it necessary to write special implementations of components which are very static and prevented the execution of JavaScript code whenever possible. Nevertheless, we were able to show that they are capable of fulfilling very simple tasks and can be used for things like text entry or coarse-grained pointing tasks. Additionally, one of the main functionalities of such devices (especially of e-readers) could be to display textual information on the displays.

As the table shows, there was one device which was not able to execute the test at all: the Sony PSP-1000. This first version of the handheld game console from Sony was not even able to load the benchmark test and was also the only device which we were not able to integrate at all into our system. Analysis of the execution of JavaScript code showed that the device provides very little memory for the web browser and even restricts the set of

JavaScript functionalities which makes it very hard to develop applications for this kind of device. Unfortunately, we did not have access to a newer version of this device and were therefore not able to test if these restrictions still apply for newer generations of this console.

Further measurements of different devices and browsers, and therefore hints for the device categorization, can be found on the web or executed manually on the web page [121].

Although any JavaScript benchmark focuses only on some functionalities provided by the JavaScript engine capabilities and the results therefore cannot be seen as strict values defining the overall performance of a web browser, the executed tests give good indications of the approximate performance that can be expected for a specific device, and if the device is capable of executing complex web applications in general. We have seen that measured values vary enormously and that they mostly represent the differences between different device types. Based on the comparison with the tested devices and the experiences gained when using them in our experimental scenarios (cp. 5.2), predictions can be made about what types of component a device will be capable to execute and how much of the functionality of a collaborative system can be off-loaded to the device.

### 5.1.2 Heterogeneity

As we have shown in the last section, the support of a device mainly depends on the capabilities of its JavaScript engine. All tested devices – except for the Sony PSP-1000 – were able to execute at least the fundamental components of the toolkit (mouse control and text input). Based on the tested devices (cp. Table 5.2) we have therefore found strong indications, that we are able to integrate any device that is able to connect to a WLAN network and that runs a web browser, as long as it is able to execute the SunSpider benchmark test suite.

Which components can actually be executed on a specific device depends mainly on the performance restrictions of the device and on whether implementations exist for the input modalities. While especially for low-performance devices (such as e-readers) serious negative performance values were found, we were able to show that they can nevertheless be integrated and that efficiency could be improved with more appropriate implementations for the specific device types.

Thanks to the reuse of standard events (e.g. mouse events) for web application control, immediate compatibility with hardware extensions – such as interactive whiteboards – is possible, which simplifies the integration of such applications even into already existing collaborative setups.

The results show that one of the main goals of our work – to support a broad set of hetero-



Device	Device type	Platform	Support
Dell XPS 420	Desktop-PC	Ubuntu 10.10	yes <sup>1</sup>
Apple MacBook Pro	Notebook	iOS	yes <sup>1</sup>
HP Pavilion dv6	Notebook	Windows 7	yes <sup>1</sup>
HP Compaq 6730b	Notebook	Ubuntu 12.04	yes <sup>1</sup>
Apple iPad	Tablet	iOS 4.3	yes
Apple iPad 2	Tablet	iOS 5.0.1	yes
Samsung Galaxy Note 10.1	Tablet	Android OS 4.0.4	yes
Apple iPhone 3G	Smart phone	iOS	yes
Apple iPhone 3GS	Smart phone	iOS 5.0.1	yes
Apple iPhone 4	Smart phone	iOS	yes <sup>2</sup>
Apple iPhone 4S	Smart phone	iOS	yes <sup>2</sup>
HTC Wildfire	Smart phone	Android OS	yes
HTC Desire	Smart phone	Android OS	yes
HTC Desire Z	Smart phone	Android OS	yes
Samsung Galaxy S2	Smart phone	Android OS	yes
Samsung Galaxy S3	Smart phone	Android OS 4.1.2	yes
OYO	e-Reader	-	yes
Bookeen Odyssey	e-Reader	-	yes
Nintendo 3DS	Handheld game console	-	yes <sup>3</sup>
Sony PSP-1000	Handheld game console	-	no <sup>4</sup>

<sup>1</sup>Successfully tested for executing the server component

<sup>2</sup>Needs internet connection when connecting to WLAN

<sup>3</sup>Special display setup (two screens one touch sensitive)

<sup>4</sup>Restricted JavaScript support

Table 5.2.: Devices tested for basic support of the toolkit

geneous devices which can be integrated configuration- and installation-free – has therefore been achieved.

### 5.1.3 Scalability

The scalability of the system depends on the chosen architecture (and therefore the message throughput) as well as on the involved devices, and therefore varies for the manifold possible scenarios. To validate the scalability of the system in our main focus use case – colocated collaboration – we performed a real scenario scaling test with a group of 13 students and one teacher in a classroom setting (cp. 5.4). The test was performed in a scenario with a single server instance (the teacher’s notebook) and a standard WLAN router based on the 802.11g standard. The test showed, that even with such a simple set-up, performance and system stability are sufficient for class sized groups and therefore for most scenarios of colocated groups.

## 5.2 In-use evaluation

The system has been used to develop several experiment settings to do basic research on user experience, group dynamics and comparative studies of different work processes, as well as for an experimental real-use case scenario. These different applications have proven the functionality and reliability of the overall system containing many different devices and have inspired further development and technical decisions.

### 5.2.1 Initial experiments

The initial experiments of the project were performed during the evaluation phase of the technology and were therefore realized based on the web plugin technology “Adobe Flex”. The experiments were performed on a shared screen, integrating up to 4 cursors controlled by Wiimotes. The experiments contained several tasks to examine – in collaboration with the psychology department of the University of Fribourg – effects of collaborative applications for group dynamics in computer supported collaborative settings.

Besides the execution of the psychological experiments, the main purpose of these experiments were an initial contact with the problem space and getting an idea of the actual issues involved in such scenarios.

#### Tasks

The following tasks were solved by one to four users interacting simultaneously, each equipped with one Nintendo Wii-mote and working in front of a shared screen.

#### *Sudoku*

A multi-player sudoku allowed multiple users to solve the puzzle simultaneously by dragging numbers onto empty fields (Figure 5.1a). In this task, drag and drop functionalities were mainly examined, as well as fundamental structures of “snap” logic (the element repositions itself within a grid depending on its drop position).

#### *Connect the numbers*

The connect the numbers task (Figure 5.1b) showed a large grid on the screen with blocks of numbers between 1 to 60 distributed randomly over the screen. The users were asked to connect the numbers with arrows by dragging an arrow from a block to the one with next higher number and releasing the mouse. This experiment helped examine more fine-grained, and therefore more precise, mouse control. Additionally, different strategies could

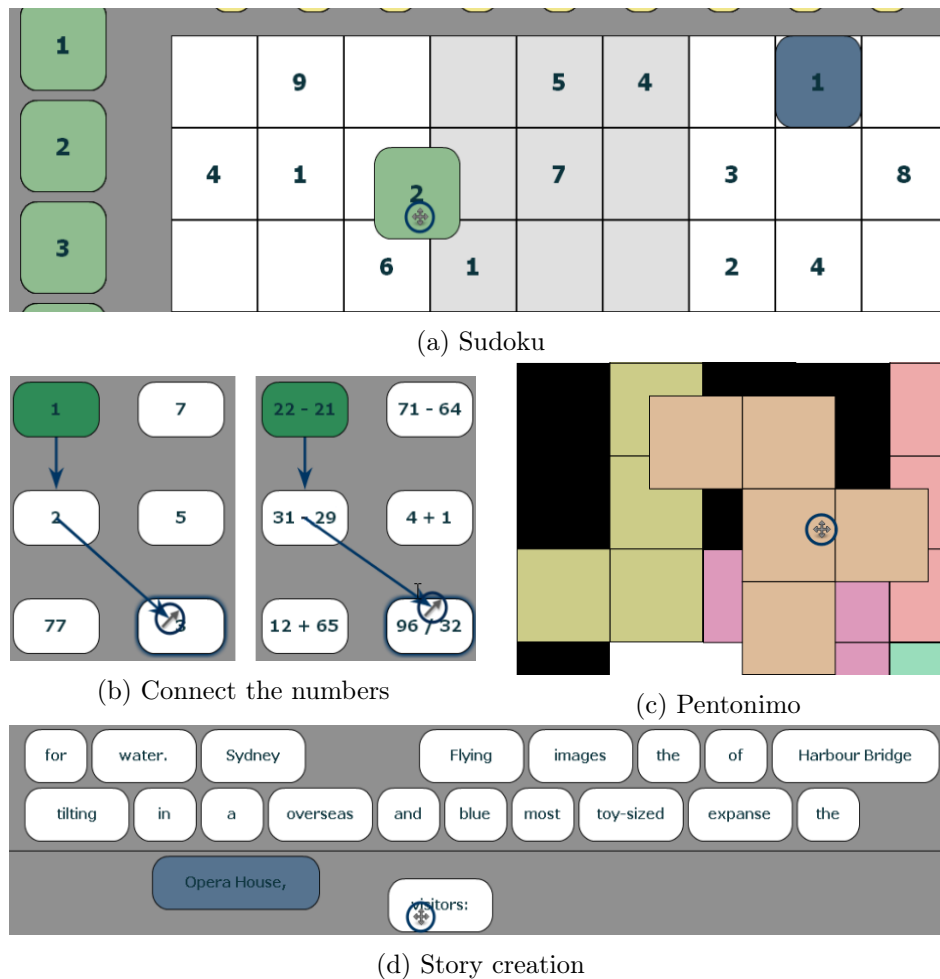


Figure 5.1.: Initial experiments

be applied to solve this problem with multiple users which made it interesting in terms of the inter-personal coordination between users when executing a complementary task. For a more complex version, mathematical equations were represented instead of numbers.

### *Pentomino*

A game where figures built from squares have to be arranged to fill a predefined space (Figure 5.1c). The figures can be dragged by a pointer and rotated by pressing a button. In this experiment, additional control mechanisms (rotation) were introduced and even more coordination between multiple users was necessary.

### *Story creation*

In this task unordered parts of sentences were provided to the users, who had to put them in the right order (Figure 5.1d). This task involves more complexity in terms of task diffi-

culty and can therefore help examine group dynamics and spontaneously arising hierarchies between users.

### **Impacts on the toolkit**

Although these experiments were run with a different technology, the fundamental concepts of concurrency control (user-based locking mechanisms), the integration of multiple controllers and their effect on drag and drop, rotation and selection mechanisms as well as the introduction in functional components such as snapping mechanisms, which have been partially ported to the final toolkit, as well as other fundamental experiences with multi-user functionalities influenced the development of the current toolkit.

#### **5.2.2 A Fitt of distraction**

After the experiences gained from the initial experiments, experiments of more fundamental multi-user interaction were run by Lalanne and Lisowska Masson [46] based on the initial version of our web based toolkit. Here, the amount of distraction caused by other users interacting with the same screen and the effects on the performance on point-and-select tasks were examined and compared between mice and wiimotes (cp. Figure 5.2). The users (in a single-user, 2 or 4 simultaneous users scenario) were asked to click with their pointer as fast as possible on squares with the same color as that which was assigned to them. When a square was clicked on it repositioned itself if the click was triggered by the associated pointer and has to be selected again at the new position. If users clicked on squares which did not belong to them, the square kept its position. After a specified number of clicks, additional simulated pointers with different colors appeared on the screen fulfilling the same task. To simulate multiple users, the implementation of this experiment introduced computer controlled pointers with predefined movement paths. The number of simulated pointers constantly increased and the influence of the additional visual elements on task performance was measured.

### **Impacts on the toolkit**

The experiment proved the feasibility of multi-user applications with standard web technologies and confirmed the effectiveness of the distinguishing between different devices. This allowed us to establish a conflict management strategy based on user-assigned components which allow their manipulation only by predefined or dynamically assigned pointers. Additionally, the results of the experiments gave hints about the potential scalability of a collaborative system in terms of usability aspects. The evaluation showed that there is a limit on the

number of simultaneous mouse pointers on the screen so that the efficiency of task-fulfillment is maintained and that the system therefore should provide concepts to reduce the amount of pointers which are visible on the screen at the same time. One direct effect of this finding therefore inspired additional concepts such as the expiring mouse pointer mechanism.

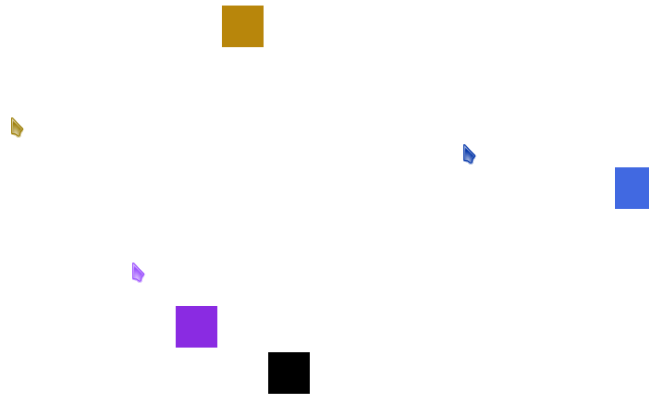


Figure 5.2.: A Fitts of distraction – Point-and-select tasks for distraction measurement

### 5.2.3 Distributed user interface experiments

In collaboration with the psychology department of the University of Fribourg, experiments were executed to examine the task solution strategies and the group dynamic effects of collaboration with distributed user interfaces. Multiple tasks were involved which were executed either by a single user or by two users simultaneously.

#### *Remote text input and mouse control (Figure 5.3a)*

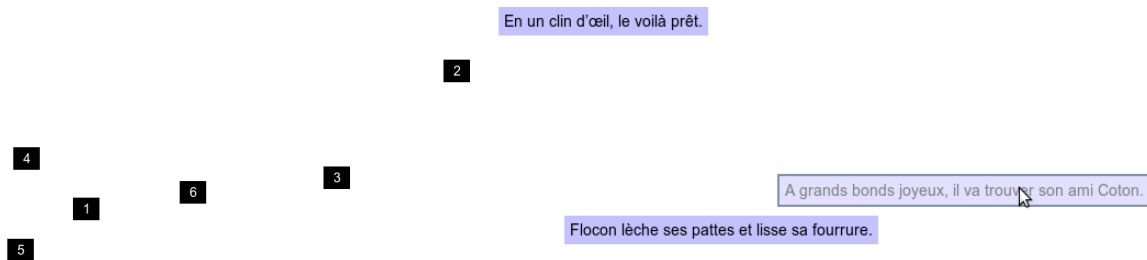
This task involved simultaneous text input by using the software keyboards of smart phones (Apple iPhones) to fill in a text box displayed on the shared screen (A) in combination with the mouse pointer control mechanism to click on the “Done” button (B) when the text input is complete.

#### *Ordered numbers (Figure 5.3b) and text (Figure 5.3c)*

A given set of numbers or snippets of texts were distributed over the screen and had to be rearranged into their correct order (ascending order for the numbers and the correct position to create a grammatical sentence for the text snippets). This task used only the remote mouse control functionality and involved some self-organized coordination between the collaborating users.

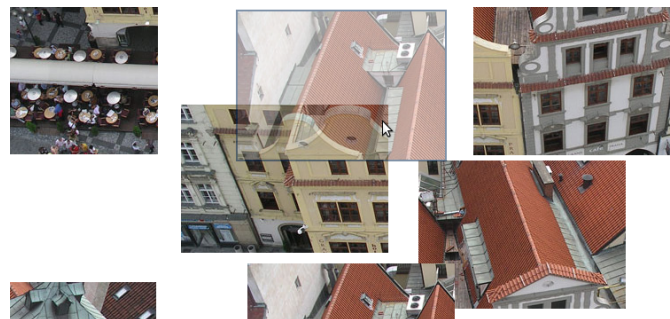


(a) Text input on a shared screen using the keyboard and remote mouse control mechanism executed on iPhones



(b) Order numbers by drag and drop

(c) Order text



(d) Puzzle

Figure 5.3.: Distributed user interface experiments

### *Puzzle (Figure 5.3d)*

The puzzle task asked users to solve a puzzle made out of a picture that had been split into parts. Again, this task was executed only with the remote mouse control.

### **Impacts on the toolkit**

The experiments not only provided us with feedback about how users solve more complex tasks collaboratively, but also introduced new concepts such as the “experiment layout” com-

ponent<sup>1</sup> as well as the automatized component control (the shared screen controls the available functionalities on the client devices depending on the task). The conflict management strategy of user-specific locking developed as part of the initial experiments (5.2.1) was ported to web technologies and was tested within this experiment in real-use the first time. Additionally, the stability and performance was further confirmed and has shown the maturation of the toolkit towards a reliable base for collaborative application development.

### 5.2.4 Computer supported brain storming

In collaboration with the psychology department of the University of Bern, a further experiment was conducted to compare the effects of computer support for brain storming sessions. Here, conventional approaches (such as brain storming with white boards and post-it notes) were compared to the solution with our system (cp. Figure 5.4) where the shared screen (A) provides space to place notes which are simultaneously written and posted by multiple users using a client application (B) executed on notebooks. The client involves two components



Figure 5.4.: Brain storming application

side-by-side: a textbox on the left to create a note widget with an “Add” button to send the note to the shared screen, as well as a mouse control sensitive area on the right that can be used to control the mouse pointer on the shared screen and to rearrange the created notes.

#### Impacts on the toolkit

The distributed user interface as well as the concept of a device specific layout (by introducing components running side-by-side on a split screen) for notebooks was elaborated, which finally resulted in the decision for a very dynamic layouting mechanism (cp. 4.2.4).

<sup>1</sup>In addition to the standard layouts for displaying modules, an experiment runner has been developed which allows to define the flow of pre-configured settings. With this experiment runner, the modules are run sequentially one after the other. It is possible to either define an ending condition (e.g. the fulfillment of a task), a timeout or both.

### 5.2.5 Usability experiments

Other internal research experiments to explore usability aspects – mainly concerning the differences between different input devices – were developed by an external developer (cp. 5.5). Here, performance of text entry, target selection and drag and drop actions as well as mouse pointer movement precision were examined with different input devices (mouse touchpad, iPhone, iPad). The task of text entry included the possibility to add texts to a shared screen using the text input on the personal devices. Target selection was measured by a task in which users had to pick up elements and drag them into a trash bin (cp. Figure 5.5a). The precision of pointer movements was evaluated by asking the users to follow a predefined path (cp. Figure 5.5b).

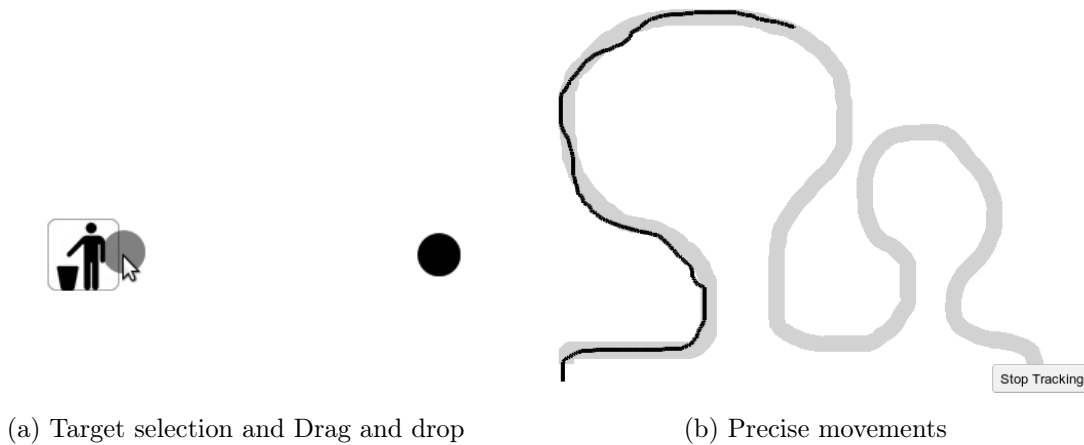


Figure 5.5.: Usability experiments

#### Impacts on the toolkit

These experiments have shown the limits and potential improvements of our remote mouse control implementations. Differences in smoothness of movements, precision, task-fulfillment efficiency and subjective user experience of comfortable use were found between the tested devices. To overcome technical differences, one of the direct results of these experiments was the launch of a bachelor thesis for finding ways in which the mouse control components can adapt themselves based on network latencies and client performance, since noticeable differences were found with constant message update rates between the different devices. Nevertheless, the results of these experiments have shown that some devices are better suited for specific tasks than others and that habituation plays an important role in terms of efficiency as well as comfort.



5.2.6 Multi-Zoom

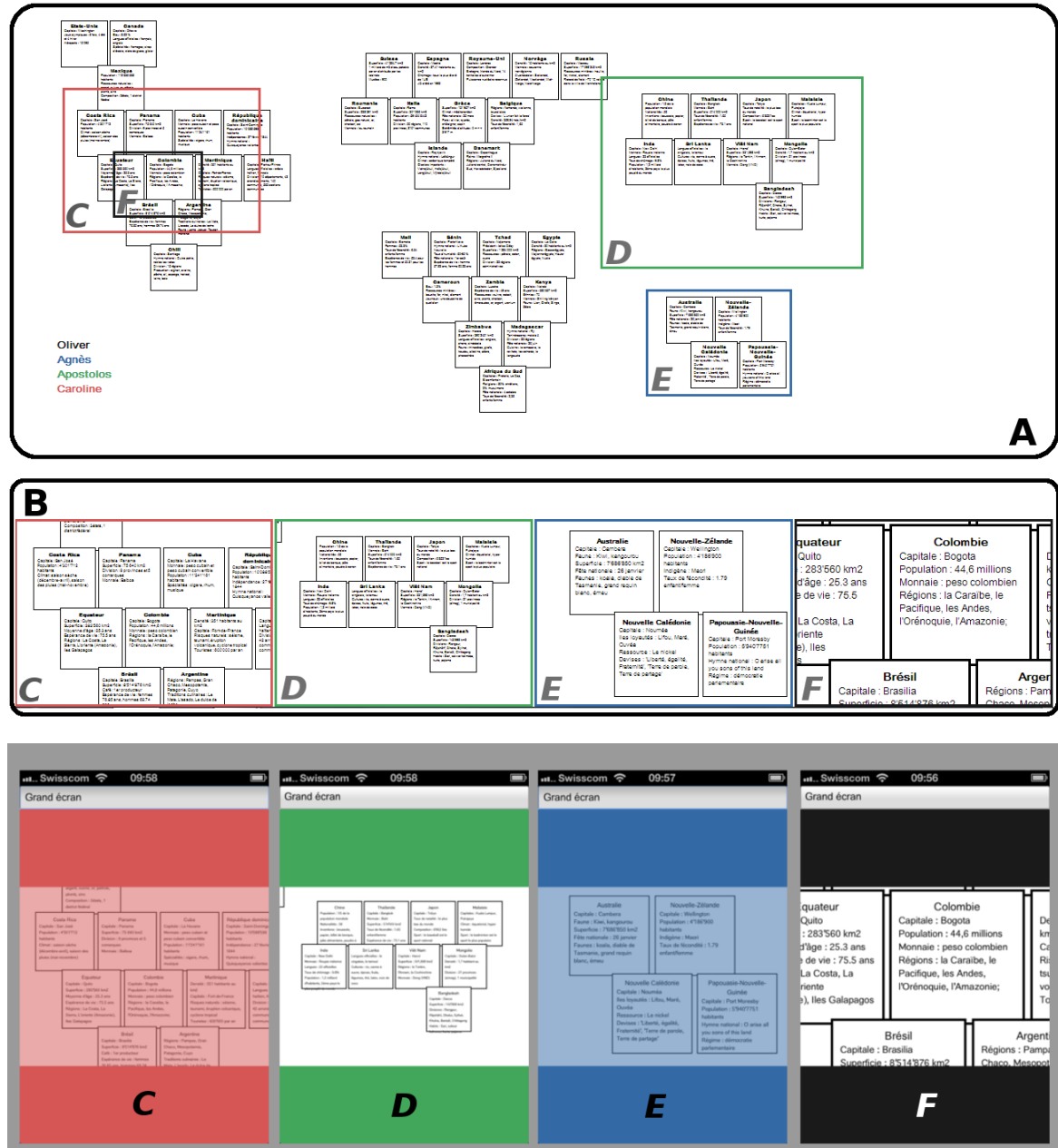


Figure 5.6.: Multi-Zoom

In a multi-zoom experiment (Figure 5.6) performed as a part of a student’s Master’s thesis (cp. 5.5), the awareness of other users in collaborative information gathering was examined. The shared screen showed an overview of a brainstorming session with detailed information (A). Four users were asked to collaboratively find specific information on this screen in three

different scenarios with the help of their iPhones. In a first scenario, the iPhones (C, D, E and F) only displayed the user color, without any additional information. For every user, a detail area (B) was displayed on the shared screen in which the user specific visible area was represented. The users were able to reposition this area by moving their fingers on their iPhones and by zooming in and out with the standard pinch-and-zoom gestures. Besides the personal details (B), the current detail area was also represented in the overall view (A) by squares coloured and sized according to the user specific detail area. In the second scenario, the detailed views (B) were not displayed on the shared screen but on the personal devices (C, D, E and F) instead. The overall view (A) still indicated the currently visible area per user and on the personal devices, the users had the choice between two manipulation modes. The users were allowed to either control the square on the shared screen (indicated on the personal device by a semi-transparent overlay – C and E) or to control the content on the personal device (D and F) which has – depending on the executed mode – the effect that the triggered actions are inverted (e.g. dragging the content in personal device control mode to the left results in a movement of the visual indicator on the shared screen to the right and moving the indicator on the shared screen to the right causes a shift of the content on the personal device to the left). The third scenario, contained the detailed area (B) on the shared screen as well as on the personal device and therefore represents the combination out of the two previous conditions.

### Impacts on the toolkit

Within this experiment, we were able to examine the adaptiveness of existing mechanisms (remote mouse pointers) to more advanced concepts (visible areas with zoom mechanisms). Additionally, we were able to gain experiences with the distribution of user interfaces and the adaptation of the level of detail of information depending on user-specific vs. shared displays.

## 5.3 A modular mindmap application

When starting the development of the toolkit, we decided on a development-leading application that would help us keep focus on the solution that we wanted to achieve. This application would at the same time be a showcase and an example application of how a collaborative application developed with our toolkit could be structured.

For this purpose, we chose to implement a collaborative mindmap application. Mindmaps have always been a very popular type of collaboration application – not only because of their obvious and very generalizable use cases but also because the complexity of such applica-

tions can be varied. A mindmap application contains different elements (e.g. text entry, reorganization of mindmaps, deleting mindmaps, display, etc.) which can be clearly separated although the influence between them has to be addressed. Many of these components need different means of interaction for different devices and we can think of different visual representations depending on the available interacting devices.

During the development of the mindmap application, many different functionalities were developed and were affected by constant changes of integrational concepts. Since the final version of our mindmap application was mainly prepared for our real world experiment (cp. 5.4), not all of the previously implemented components were integrated because they were not useful for this specific scenario. Nevertheless, these additional components are still available and ready to be integrated for any potential future use cases. These additional components therefore shall be presented in this chapter just after the overview of the latest version of the mindmap application.

### 5.3.1 The latest version of the mindmap application

The latest version of the mindmap application is based on the standard modules of the dynamic layout and makes use of the in-built eventing mechanism, remote mouse control, authentication and easy access mechanisms. The application differentiates between two modes: execution as a shared device and as an input-device.

#### The shared device

The shared device (requested with the URL parameter “deviceType=multicursor”) allows multiple clients to interact with the application either by adding notes or by using remote mouse pointers (cp. Figure 5.7). It contains an application specific component (the `MindMapCanvas`) which is the main component and provides space (in red) for the notes (which adapt their size to the length of their content) to be placed, as well as a trash bin (G) to remove notes by dropping them over this area. Besides this, the canvas component also contains a toolbox at the bottom of the screen to control the current settings. The toolbox is only accessible by the native cursor, which allows the person that controls the shared screen to define the different options of the application.

The visible remote cursors are annotated with the capitalized first two letters of the user name (“AL” for Alice and “BO” for Bob) to establish awareness of the cursor-to-user assignments. The chosen drag and drop strategy is a simple locking mechanism (a note which is dragged by one user is locked for others).

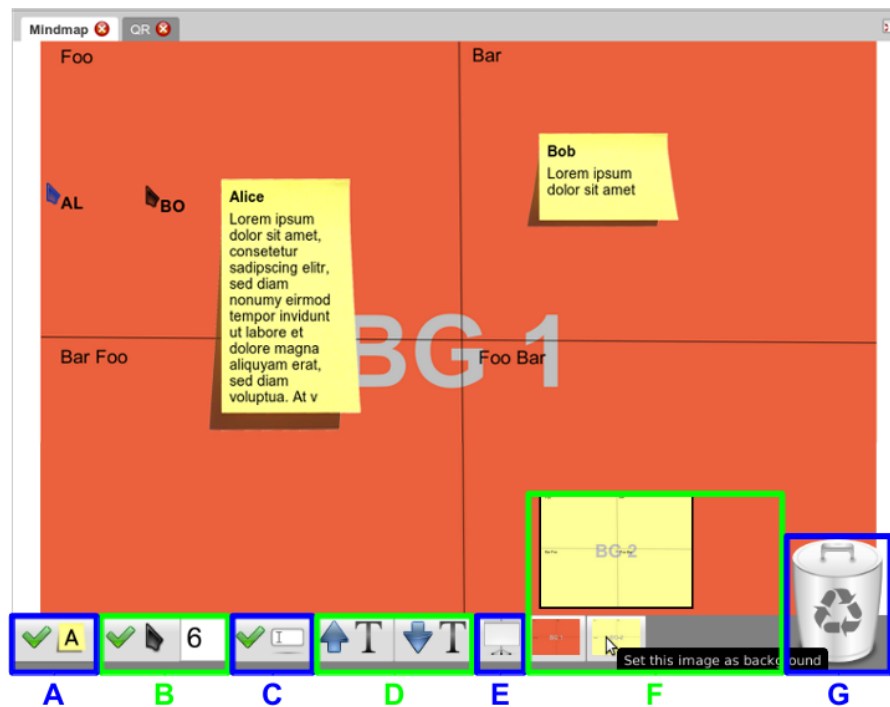


Figure 5.7.: Shared mindmap screen

The toolbox contains a toggle button to show or hide the content of the notes (A). This allows notes to be collected without the collaborators being able to see what other collaborators have written. By toggling the button, all content becomes immediately visible. Additionally, the native cursor of the shared screen can toggle the notes individually by clicking on them.

The multi-pointer functionality can be switched on and off by toggling the button shown in B. This allows users to be prevented or enabled interaction with remote pointers. The text field next to that button is used to set the maximum number of simultaneously visible cursors. By default, this value is set to 6 based on the results of experiments in section 5.2.2. A change of this value causes an immediate adaptation of that limit.

The button indicated by C allows to enable and disable text entry from remote users. By switching off this functionality, no further notes can be added to the shared screen.

The buttons indicated by D allow to increase or decrease the text size of the notes. The size of the notes will adapt accordingly.

With button indicated by E, the background image can be reset to an empty screen while the area of F allows to add images from any part of the system. The pictures then are represented as new buttons which are presented in a preview when hovered over by the mouse pointer. A click on one of these buttons switches the background to the current image. This allows

for example to predefine categories to coordinate the placing and categorization of the notes. Besides this main component, the shared device also contains a “QR” component (visible as a tab at the top of the page) which supports easy access (cp. 4.2.6).

### **The input device**

The client side contains the mouse pointer control component (cp. 4.3.2), a component that contains a text box, as well as an add button to enter the text and send it to the shared screen. The mouse pointer control not only allows to reposition the mouse but also triggers click events and allows users to drag and drop elements on a remote device. To start dragging, users have to touch the screen for a few seconds to switch the device to the drag mode, indicated by a notification on the client device’s screen. If the drag mode was started on top of a draggable element, the user is now able to move the element and to release it at its target position with a short tap. The text input component is the same for mouse based and touch based clients and therefore is implementation independent. That component makes use of the distributed eventing mechanism to send the information to the server instance.

### **5.3.2 Non-integrated components**

Because of the latest use of the application in an educational scenario, several components which were implemented in the past were not used for this specific use case. These components, which are still available in the toolkit include:

#### *Client side representation of the mindmap*

A component that provides a view of the display on the client devices, with a distinction between mouse and touch oriented devices. While mouse oriented devices get a similar representation to what is visible on the shared screen, touch based devices are provided with a list-oriented view of the available mindmap notes. When a note is selected, it is locked and can only be edited with the device that has selected it.

#### *Select and edit*

A component which can be used to select a note using a remote pointer on the shared screen with a single click. The note then appears on the personal device of the user for editing and is locked for other devices. This component is a good example of the bi-directional communication between the client and the server instance.

### ***Image upload***

Instead of text, this component allows to send images from the personal device to the shared screen. Here, the image is uploaded to the server and references are sent to the other devices.

### ***Disclosure on hover***

This component has not been fully developed but is rather at a proof of concept stage. It allows to hover over a note with undisclosed content using a remote pointer disclosing the clear-text content on the personal device. Using “disclosure on hover” can for example protect messages, make their content available only to a specific set of users, or enrich a note with meta-information (e.g. because the available space is restricted on the shared screen).

## **5.3.3 Modularity in practice**

The mindmap application is a good example of what we mean when we are talking about modular applications. Functionalities can be added or removed based on the current needs of the application. Some components might be suitable for a specific scenario while others not. Additionally, combining very generic components (e.g. remote mouse pointer control) and application specific modules (e.g. the note creation component) is easily achieved and the different elements are integrated smoothly. Moreover, the application shows how our dynamic layouting system works with any combination of the different available components. For example, with one further line of code, the note creation component could be added to the shared screen as well and the layout would just add it as another tab that could be rearranged by drag and drop. The visual scalability of our layouting system is therefore validated and the exchangability of the different elements is achieved.

## **5.4 Real-world experiment: Use in an educational scenario**

In addition to the experimental settings which focused on very specific tasks, we wanted to test our system in a more real-world situation. We therefore performed an experimental run of our case study mindmap application in the environment of a school class at the Gymnasium Neufeld high-school in Bern.

### **Environment and setup**

The students were asked to bring their own devices such as smart phones, tablets, etc. For those students who did not have access to appropriate devices, additional devices were provided so that no one was excluded from participation.

The experiment was located in the regular classroom where the shared screen was projected by a fixed projector controlled by the teacher's notebook, which acted as a server at the same time. The network setup was established using a standard wireless LAN router (802.11g) without internet connection, providing an open network connection without password protection and encryption (Figure 5.8).

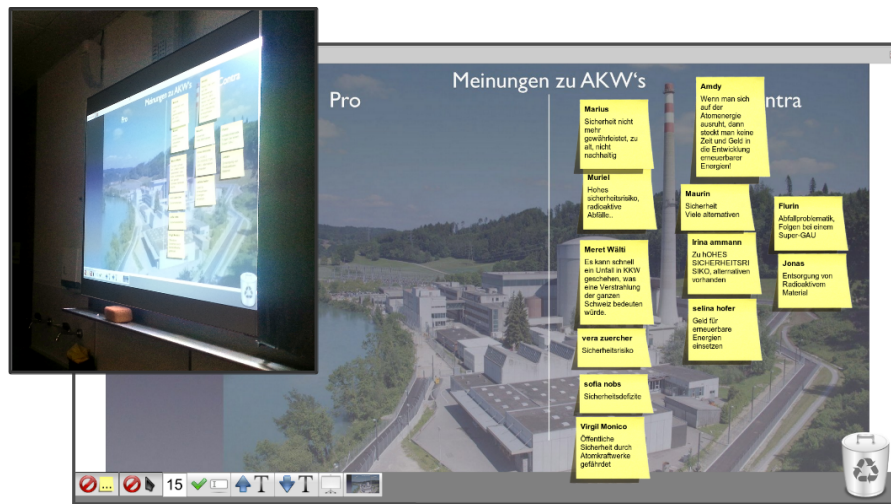


Figure 5.8.: Mindmap of the students' opinions about nuclear power plants

The experiment was run with the latest version of the case study application including the use of the dynamic layout components, encrypted communication by the distributed eventing mechanism and other components to test the suitability of the overall integrated system for a real-world use scenario.

### Experimental run

11 of the 13 students as well as the teacher had their own smart phone. Two of them had to be equipped with our devices (one HTC Desire smart phone and one Samsung Galaxy Note 10.1 tablet). 8 of these smart phones were iPhones, where the versions varied between 3, 3GS, 4 and 4S. Besides these, 3 Android phones from Samsung and HTC were used. Because of (non-system related) technical issues with the HTC smart phone, it was replaced by an iPad 2.

The connection to the open wireless LAN, as well as to the application, worked with all the devices immediately and was considered to be easy or very easy to use by more than 75% (cp. Figure B.3) of the users, and no one found it very difficult to use. This is surprising, especially because they were asked to connect to the system without any means of easy access – they

had to connect to the WLAN and enter the technical URL into the web browser manually.

After they connected to the application, the students were asked to follow a line projected on the shared screen with their remote pointers and to create notes and move them on the screen to become familiar with the provided tools.

The teacher then asked the students to read some facts about nuclear power and to write down their opinions (pros and cons) about nuclear power plants using the system by putting notes onto the screen and dragging them (in an undisclosed state – the content was not visible to the other students) depending on their content to either the “pro” or the “con” section of the screen. The system was configured to not restrict the number of mouse pointers and therefore all 14 users used their remote pointers at the same time.

Since the system allows to delete notes by dragging them to the trash bin, some students started to delete notes of others, which made it necessary for the teacher to intervene by locking the remote pointer mechanism temporarily.

After this initial experiment, the teacher switched back to conventional means of education by presenting information, discussions and showing a video. Then, the students were asked to point to the locations of nuclear power plants in Switzerland collaboratively by moving their cursors to the different places on a map. Since every student only had one pointer but multiple nuclear power plants exist in Switzerland, this implied some coordination between them to make sure that all locations were covered. Here again, all 14 pointers were in action simultaneously.

Finally, the students were asked to add notes to the system with ideas about what they could do to reduce their own energy consumption and therefore to reduce the need for nuclear power, and to order them by the impact the change would have on their personal lives.

## **Results and observations**

During the experiment we observed that the students were very motivated to get the system running and were rather impatient for example when connecting to the wireless LAN although no additional delay to the usual connection time was noticeable. In addition to this, as they were using the system, we observed spontaneous reactions such as “das isch no fräch” which loosely translates to “this is surprising and cool”. Additionally, the students even ignored the ringing of the bell that indicates a break between two lessons. When the teacher called attention to the fact that it was time for a break, they said that they would prefer to continue using of the system with the reasoning “das isch luschtig”, which means “it is fun”.

At the end of the lesson the students were asked to fill in a questionnaire about their experi-



ences with the system. The results, as well as the questionnaire, can be found in Figure B.3 or Figure B.1 and Figure B.2 respectively.

In addition to the already mentioned finding that the users did not have major problems connecting their devices to the system (only 7.7% found it difficult to connect), the text entry also did not cause problems (no participant found it difficult or very difficult to enter text), although some usability aspects such as that the text box on their devices was too small or the extra steps for switching between the mouse pointer control and the text entry, were mentioned as points for potential improvement.

More issues were found with the task to move objects around on the screen, which was rated by 23.1% to be difficult. The issues were mainly to find the personal mouse pointer on the screen (rated by 38.5% to be difficult or very difficult) as well as getting used to the catch and release mechanism of the dragging implementation (cp. 5.3.1). Although the recognition of the personal remote pointer on the shared screen was difficult for a large part of the class, almost a quarter of the participants found it easy to recognize and find their pointers. This might be due to the physical locations of the students which were not the same for all of them and therefore, students with shorter distances to the screen might have been able to better distinguish their pointer from others due to better visibility. In terms of performance, the participants found that the system reacted either ok (76.9%) or smoothly (23.1%) when moving the mouse pointer, and none found that there were delays.

The expectation that many of the students would own a compatible mobile device was fully fulfilled. Only one student did not own a smart phone, whereas one other student did not have the device with her. Although most (8) of the students owned Apple iPhones in different versions (3G to 4S), four devices based on the Android OS produced by HTC or Samsung were used as well. We can therefore see, that the expected heterogeneity of devices is realistic and that a collaborative solution like ours has to support multiple platforms so that no participants are excluded. Additionally, eleven of the twelve device owners use those devices on a daily basis, mostly for calling, SMS and internet.

In terms of privacy, two students (of the 12 which owned their own device) had concerns about using their personal devices in school – besides the undesired permanent use of electronic devices in school, the fear of a loss of data was also mentioned to be an issue. Since our system addresses that issue explicitly, this concern could be addressed by explaining how our system prevents the loss and/or disclosure of personal data.

Most of the students said they would install an additional application on their device for working with a system like this in school although one of them has mentioned the precondition that the application should be available for free. Opinions about whether they would like to use such a system regularly in class were divided: While half of the class would like

to continue the use of the system, the other half of the class would prefer not to. Within the open question of what the students liked the most and the least about the system, we were able to find a hint which could (partially) explain this split. Some of them claimed that the conventional way of teaching would have been at least as successful and therefore have pointed out that the use of the system should be very carefully planned in all collaborative and educational settings to support the actual goals so that the use would not be seen as an end in itself.

Similar impressions were reported by the teacher in an open interview performed after the experiment to explore his opinion about the system and if he felt comfortable using it in an educational context. Based on the experience gained with this experiment, he mentioned the necessity of finding a good balance between using such a system and using traditional teaching since it seemed to him that it is very important that such a system provides an added value in an educational setting. He said that he could definitively see such an added value if the system is applied in the right context.

In response to the question of whether he would be willing to reuse this or a similar system in his classes, he said that he would not only reuse it but also would like to extend its use for other purposes (e.g. voting and other more interactive applications).

His ideas for improvements (such as to clear all notes on the screen with a button or to apply a mode for permanently visible cursors) were more application specific suggestions rather than suggestions of improvements to the fundamental system and even extensions to other fields of education (e.g. economics) as well as interest from other teachers were mentioned. This feedback gave us strong indications that educational settings could potentially be a good use case for our toolkit and our system.

In addition to the general observations about the user experience, and system performance, we measured the network throughput of the system during this experiment. We measured the data flow on the WLAN router to which the clients (the devices of the students and the teacher) but not the server (which was connected by cable) were connected by wireless LAN. The WLAN data input (data coming from the connected devices) reached a maximum peak of about 85KB/s and was rarely above 50KB/s during the experiment (cp. Figure 5.9). This measured value is the actual received data at the WLAN router and therefore includes resent packages (e.g. because of package loss) as well as the complete overhead of the network protocols involved.

Thinking of the average capacity of a wireless LAN network (cp. *Network latency* in 5.1.1) which is about 1.6MB/s, even a standard wireless router seems to be capable of handling much more than 14 devices and is therefore not an actual bottleneck for realistic collaborative scenarios with these types of interaction.

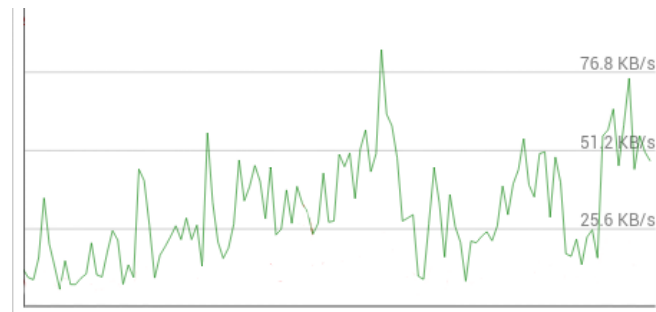


Figure 5.9.: Load of the WLAN interface on the router during the execution of the real-world use experiment (involving text addition and moving objects)

## 5.5 Developer evaluation of the toolkit

Two other developers used an almost final version of the TWICE toolkit to create concrete applications and experimental setups.

One of the developers had experience with Java development but not with the Google Web Toolkit. He worked on the usability experiments (cp. 5.2.5) run in our research group. His feedback – collected in an open interview – was that most of the issues that he faced during the development were not caused by our extensions but rather with the general GWT structures and his unfamiliarity with this technology. Although aware that the application he was developing would finally run in a multi-user scenario (using for example the provided drag and drop library), he developed his applications purely in single-user mode and was not testing the applications in a multi-user scenario until the end.

The integration into multi-user mode worked as expected and very smoothly with the addition of only three lines of code (to launch the multi-pointer module). The developer mentioned that he felt very comfortable not having to think about the complex issues of multi-user use while developing and was instead able to use the standard functionalities of the Google Web Toolkit.

Since he was working with a later version of the Google Web Toolkit than that which was used for the development of the drag and drop library, he faced the issue of the current incompatibility of the interfaces between our drag and drop library and the newly added standard interfaces provided by the Google Web Toolkit (cp. 4.3.1). This enforces our belief that the developer generally expects the toolkit to be compatible the standard interfaces and therefore that the re-use of APIs (cp. 4.1.2) is an essential requirement for simplified use.

The second developer used the toolkit to set up an experiment to examine different zoom strategies on a shared screen in a multi-user context (cp. 5.2.6) for her Master’s thesis. In the

beginning, she was unfamiliar with the Google Web Toolkit and had only basic knowledge of the Java programming language. She was faced more with the actual distributed nature of the final application since new interaction modalities (especially for touch based devices) had to be introduced and therefore existing components had to be extended. In an open interview, she mentioned that in the beginning, it was rather difficult to understand which of the many available modules fulfilled which purposes. Improved documentation of the different modules and especially graphical overviews of the dependencies between the modules, typical combinations for specific scenarios (e.g. “for systems with a shared screen, remote pointers and text input, the modules X, Y and Z are recommended”) and code examples of applications for the most common settings were mentioned to improve support of the developer to gain a better overview of the elements available in the toolkit. She also mentioned some other initial issues she had when starting to use the Google Web Toolkit (GWT) due to the unfamiliarity with the technology and needed some time to understand the concepts and the elements of this technology. Once she understood the general idea of GWT, she said that she thought that it was a comfortable and useful way to implement web applications. In particular, the modular structure introduced by our toolkit and the concept of deferred binding allowed her to focus on the requirements, functionalities and the devices involved in the specific use case she was focusing on, which according to her substantially reduced the complexity of development. Another big advantage that she mentioned was the fact that she was able to build the special functionality (zoom) on top of already existing mechanisms (remote mouse pointers), which allowed the reuse of existing code and therefore reduced the effort of setting up the required fundamental structures (e.g. communication channels, separation of events depending on their origin, etc.). In a general statement, she said that our toolkit was helpful for developing the application and that she had the impression that the toolkit provided the required functionalities for the development of many different scenarios and applications and therefore was well suited as a generic toolkit for collaborative systems.

In a self-evaluation of the toolkit, we have tried to reflect on the achievement of the initial goals in terms of simplified development during the development of the final version of the mindmap application (cp. 5.3). Due to more than 3.5 years experience of working with GWT in industrial projects, the familiarity with the underlying technology was already given, which increased awareness of the traps that make development of web applications hard (e.g. order of instantiation and delayed accessibility of elements’ scaling dimensions). Nevertheless, the reuse of the previously developed components such as the remote mouse pointers as well as the dynamic layouting structures helped to create prototypes of the application, including its main features, in a very short time. This was a big advantage in terms of coordination with non-technical stakeholders (in this case the teacher of the school class that we ran the real-world experiment with – 5.4) since the discussion about specific features (e.g. the tool-

bar at the bottom which is only accessible by the native cursor and which allows to control the different functionalities) was possible based on an already running application and the suitability of specific components was testable through activation or deactivation of them at runtime. Besides the simplified specification phase and therefore a reduction of the risk of creating software that does not cover the actual needs of a specific scenario, the possibility to reuse standard GWT components (e.g. widgets like buttons) – especially because of pre-existing experience with GWT – helped to structure the different elements and therefore to advance rather fast while handling special behavior within the multi-user scenarios with very few additional method calls. Another benefit we experienced was the possibility to test the application without the need to spread it over multiple devices during development. Multiple devices were simulated by simply adding another browser window and therefore multiple clients were executable on the same device. This was especially helpful because the Google Web Toolkit contains a “development mode” which allows to execute the client-side Java code (based on a browser plugin) within the web browser without pre-compilation into JavaScript and enables the developer to use the standard tools for debugging that are available for standard Java development. With the simultaneous execution of the different clients on the development machine it is possible to execute and comfortably debug the communication between the different instances, and therefore the whole life-cycle of the overall system. As the first of the other developers mentioned, we also experienced the issue of the non-standardized API of the drag and drop functionality. Since all other components follow the conventions of standard APIs and the developer is therefore used to not having to worry about learning new interfaces, the impact of this exception seemed to us to even be stronger than it would have if all other components would have had only their own interfaces as well. Nevertheless, in general, we were able to see that our toolkit was very helpful for developing and customizing this application in very short time and that it substantially simplified the development process.

To conclude, it seems like our goal to reduce the additional complexity for the development of standard multi-user applications has, for the most part, been achieved. Components (such as the drag and drop library) appear to be the most comfortable to use if they are compatible with their corresponding standard interfaces and therefore have to be constantly adapted to newly integrated APIs in standard GWT. The complexity of development increases if special behaviour is needed (e.g. additional functionalities are required which are affected by the distributed nature of the system) but can be simplified when basing it on already existing code of similar or related components.

Although general documentation of the different elements in particular could still be improved, it can be said that our toolkit provides an appropriate guideline and frame for the development of standard collaborative applications and allows the extension of functionali-

ties. We have indications that developers with pre-existing GWT skills will quickly become familiar with the newly introduced concepts and APIs and therefore the steepest learning curve is the one to become accustomed to the concepts and elements of standard GWT.

## 5.6 Discussion

The theoretical and technical evaluations indicate that the necessary performance and scalability can be provided for most of the realistic scenarios and show that a broad range of currently available devices are fully or at least partially supported by our system. Additionally, the different experiments performed have not only proven that the toolkit is stable and reliable but at the same time have implied challenges (e.g. handling conflicts between multiple cursors, distribution and “remote control” of user interfaces, overcoming issues of network latencies, calibrating message rates) which had to be solved to provide the needed functionalities for being able to establish the experimental settings. They were therefore useful not only for evaluation but also for driving the development of useful and reusable components and improvements, and at the same time ensured that the developed components were generic enough to be applied to different scenarios. In particular, the collaboration with experts from other fields (e.g. HCI and psychology) helped to avoid a purely technology-driven development approach and brought up explicit needs for real use functionalities which then had to be realized with the available technology. This ensured that the development of the toolkit was headed from the beginning towards a solution whose development did not stop at the specification phase but whose use and functionality was validated in real applications.

To evaluate the suitability of our toolkit for real-life applications, our experiment in an educational setting showed that the system is capable of handling 14 simultaneous users and we have strong indications that it would support even more than that. Additionally, it gave us positive feedback about the usability of the system from an end-user perspective, although the results show that especially in the area of user interface design improvements are still possible and necessary.

Software developers working with our toolkit have mentioned that using our extensions of the underlying standard technology introduced very little additional required knowledge and that they therefore did not have to worry about the complex functionalities needed in a distributed multi-user scenario.

We can therefore conclude, that although improved implementations and more complex scenarios would still have to be elaborated and that additional evaluations (e.g. with experienced GWT developers) would help to further examine if the defined requirements have been fully fulfilled – the available results are good indicators that we have achieved our main goal of

---

creating a toolkit which provides the necessary structures for coordinated and reusable software development for collaborative applications. TWICE has become an important tool for the development of software for experiments in our research group and has shown that it has several interesting use-cases in non-academic settings as well.

# 6

## Conclusion

Computer supported collaboration is an interesting field affected by constant changes. The devices involved, available networking infrastructure and technology awareness and habituation of users has changed a lot within the last years and will further evolve in the future. Technologies, platforms and device types constantly appear and disappear and the leading companies within these fields, and therefore the platforms they provide, may not be the leaders in the future. All this makes the development and the maintenance of heterogeneous systems very difficult and often implies redesign, restructuring and partially even rewriting of whole applications to adapt to these changes.

We have therefore developed a toolkit (TWICE) for the development of collaborative applications based on web technologies in order to avoid the necessary and constant modifications needed to keep existing code running. Using web technologies, we can focus on the new opportunities which are provided by more advanced devices and web browsers and therefore to put work into the extension rather than the maintenance of code.

In TWICE, we explicitly address the issues of device heterogeneity in terms of support (to make sure that no users are excluded from a collaborative setting because they own a non-supported device) as well as in terms of adaptability to device-dependant specificities by providing device and/or device-type specific implementations of functionalities depending on the capabilities and input modalities of a particular device.

To support spontaneous collaboration, another general goal of our toolkit is to establish simplified access to the system for end-users. Particularly in situations of ad-hoc and short-term collaboration, users might not be willing to execute complex installation and configuration procedures to launch a collaborative application. We have therefore found ways to provide access to a collaborative system without any installation and/or configuration requirements.

As a third general goal, we have focused on the simplification of the development process of multi-user and multi-device applications. We did thus recycle the syntax that is provided for



---

single-user application and extend it with additional functionalities which make it applicable for multi-user applications. This reduces the complexity of development, debugging and testing of multi-user applications (since they can often simply be treated in a similar manner to single-user applications).

Finally, our toolkit provides basic functionalities required for most collaborative settings, a modular and extensible structure which allows comfortable replacement of components or implementations of specific functionalities at runtime, and more concrete software modules which are representative of how to integrate further functionalities into the already existing toolkit. The modularity, and therefore the possibility to include, exclude and replace modules dynamically, simplifies experimentation with and comparison between different approaches for similar issues as well as the (re-)combination of different functionalities to examine the most suitable set-ups for specific tasks.

## **Contribution**

The state of the art concerning the main issues related to computer supported collaborative systems (2) showed a need to implement our own toolkit for the development of collaborative applications since our research context (3.1) requires functionalities (3.2) which were not possible to fulfill, which would require extensive customizations and work-arounds or would have ended in unsatisfying compromises when being realized with existing tools.

By choosing web technologies in general and the Google Web Toolkit in particular, we have chosen a technology stack (cp. 3.3) as the base of our toolkit which will likely be future-safe due to its widespread use, its large developer community and its continuous adaptation to new features and concepts, while ensuring backwards compatibility (because the masses of already existing, “legacy” documents and web applications which still have to be supported by browsers). Web technologies are widely accepted by developers in general since they are the base of a very large number of existing applications and are gaining in importance in their role as a dynamic and platform-independent software stack runnable on a heterogeneous set of devices. Although dependent on the actual implementation of a software, web technologies provide native general means to improve their availability (by redundancy) as well as their scalability (by load-balancing) since the capability to handle a large number of users simultaneously has been one of the key requirements of web technologies from its beginnings. Because of the built-in availability of web browsers in almost any modern personal device, the support of web applications is provided in almost any context in an installation and configuration free way. Besides the advantage of being able to create true walk-up-and-use scenarios and therefore to simplify and encourage spontaneous collaborative settings, the execution of application code in a browser sandbox provides fundamental means to ensure privacy. Since

---

the browser sandbox restricts capabilities of applications and allows users to explicitly define permissions for a specific privacy and security critical functionality at run-time, means for precise control of accessible resources (e.g. private data, cameras, etc.) are provided to the user.

In our toolkit (4) we have introduced general concepts such as the reuse of APIs, replaceable and extensible modules (4.1) to be able to ensure simplified development, and the reduction of differences in development between multi-user and single-user applications to establish compatibility with mature and well-known development tools. These concepts not only allow to provide varying implementations depending on device dependant specificities but also enable optimization of the use of the device's resources thanks to the system-managed life-cycles, and therefore the dynamic allocation and release of occupied resources of the different software components involved. Adaptations of the software to the specificities of heterogeneous devices are therefore possible at runtime which allows them to react in a very flexible manner to the incompatibilities, user-preferences and performance issues of the different devices involved.

We have also implemented basic functionalities to provide tools to overcome general issues of collaborative systems (4.2). To reduce delays in the complex network communications involved in distributed systems, a messaging mechanism has been introduced and integrated through the well-known pattern of an application-wide event bus. The centralized message handling not only allows optimization of network traffic but also provides security functionalities such as receiver-specific encryption, the provision of digital signing of messages as well as checks for data integrity. In addition to simplifications of user access to an already existing collaborative setting (through QR codes, WiFi hotspot landing pages, etc.), multi-user support is ensured through remotely controlled multiple mouse pointers, multi-focus functionalities as well as replaceable conflict management strategies (e.g. pure locking mechanisms, optimistic and pessimistic concurrency control). Dynamic layouting mechanisms and concepts of user interface distribution complete the basic set of functionalities for the development of collaborative systems, allowing the development of complex multi-display applications.

The more specific software modules (4.3) that we provide such as drag and drop and collaborative web browsing functionalities are examples of how the general concepts of the TWICE toolkit can be applied in concrete use cases and show how interacting components can be integrated with each other in a collaborative context. Besides their actual functionality, they act as showcases for further extensions and component development by third party developers.

---

In-use evaluations in different experimental scenarios (5.2) and an experiment in a real-world classroom situation (5.4) have shown that applications developed with our toolkit fulfill the requirements of realistic collaborative scenarios such as support for heterogeneous devices, adequate performance and scalability, and functional requirements such as dynamic replacement and combination of application modules. Through abstraction of the complexity of multi-user scenarios within our toolkit, we have achieved the goal of providing extended functionalities without the need for developers to learn new APIs (unless they want to obtain more fine-grained control over an extended functionality). The achievement of the simplification of the development of multi-user functionalities has been validated through open interviews with developers with varying levels of experience with the technologies involved (5.5).

### **Future work**

During this thesis, we were able to develop the necessary concepts, structures and initial components for a generic toolkit. Concrete potential improvements are the adaptation of the drag and drop library to the standard APIs, as well as the continuous update and extension of the documentation according to the changes in the TWICE toolkit as well as the provision of additional examples of code to provide starting points for developers with very specific needs. Other future work would involve further development of concrete components (e.g. instant messaging, video conferencing, document exchange, etc.) which would allow to establish more complex collaborative systems through the combination of powerful application modules. Additionally, device compatibility and further refinements of implementations to device-dependant specificities can further be improved and other actions could be taken to improve user experience.

In addition to the extension of the toolkit functionalities, adaptations to specific collaborative situations and psychological aspects (e.g. group dynamics) should be taken into account. Differences between group-sizes, colocated and remote as well as synchronous and asynchronous collaboration should further be examined to gain knowledge about the factors influencing successful and efficient collaboration. User acceptance of the system in general, and the specific components in particular could therefore be evaluated in different scenarios and actions could be taken to improve the comfort of use of such collaborative systems.

Furthermore, more examinations would be needed in terms of the acceptance of the toolkit by third party developers, as would further improvements and simplifications of APIs to continuously flatten the initial learning curve for less experienced developers and therefore to let them become familiar with the creation of collaborative applications more easily. Additionally, real-world experiments would lead to indications about what is missing in terms

of concrete functionalities. We will therefore stay in contact with the high-school where we were allowed to run our real-use experiment in order to be able to perform further evaluation and develop more specialized applications for educational scenarios. Additionally, we will try to find other fields of potential application (e.g. in business) to get a broad set of different contexts and therefore to improve the generalizability of the fundamental toolkit while providing very specific application modules for reuse in similar situations.

To keep the toolkit alive and to spread its use outside of our research group, we have released the code under a very open license and hope that we will be able to create a developer community with users from academics as well as industry so that our toolkit can evolve with most current knowledge and newly introduced concepts while getting feedback about the suitability of the different components for real-world applications.

## Epilogue

We are convinced that there is a strong need for the integration of the different existing solutions for the issues that collaborative systems face into a unified platform to be able to compare, exchange and re-combine the different components and therefore to execute experiments to find the most appropriate solutions for specific collaborative tasks. By providing our modular and extensible toolkit, we have tried to create a technological base structure which allows the integration of the manifold solutions for the different specific issues related to establishing collaborative systems that we found in the related work.

We believe that web technologies are the best base for such a unified platform due to their broad acceptance. Several leading IT companies also seem to believe that the web technology stack will become a proper software engineering platform for heterogeneous devices and for collaborative applications in the near future. Microsoft, for example submitted new pointer specifications to the W3C very recently ([112]) and Google is working intensively on the standardization of real time communication through web technologies ([124]).

Although we are aware of the fact that our toolkit does not solve all issues related to the development of collaborative applications, we think that we have created a solid base for further development and therefore we hope that our approach, the presented ideas and our technological decisions contribute to the challenging but necessary task of simplification of the development process of distributed collaborative applications, and that at some point in the future our toolkit will become the base of one of the “[...] *“breakthrough” systems that will influence the next generation of toolkits*” [65].

# A

## Acronyms

<b>AJAX</b>	Asynchronous JavaScript And XML
<b>API</b>	Application Programming Interface
<b>CSCW</b>	Computer Supported Collaborative Work
<b>CSS</b>	Cascading Style Sheet
<b>CIA</b>	Confidentiality, Integrity, Availability
<b>DOM</b>	Document Object Model
<b>DUI</b>	Distributed User Interface
<b>GUI</b>	Graphical User Interface
<b>GWT</b>	Google Web Toolkit
<b>HCI</b>	Human Computer Interaction
<b>HID</b>	Human Interface Device
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>ICE</b>	Interactive Collaborative Environments
<b>IDE</b>	Integrated Development Environment
<b>JavaEE</b>	Java Enterprise Edition
<b>JavaME</b>	Java Micro Edition
<b>JavaSE</b>	Java Standard Edition
<b>JSNI</b>	JavaScript Native Interface
<b>JSON</b>	JavaScript Object Notation
<b>JSON-P</b>	JavaScript Object Notation with padding

**LAN** Local Area Network  
**MAN** Metropolitan Area Network  
**MDA** Model Driven Architecture  
**MPG** Mixed Presence Groupware  
**MPX** Multi-Pointer X Server  
**MDG** Multi-Display Groupware  
**NTP** Network Time Protocol  
**OSGi** Open Services Gateway Initiative  
**PDA** Personal Digital Assistant  
**QoS** Quality of Service  
**QR** Quick Response  
**REST** Representational State Transfer  
**RIA** Rich Internet Application  
**SDG** Single Display Groupware  
**SDK** Software Development Toolkit  
**TCP** Transmission Control Protocol  
**TWICE** Toolkit for Web-based Interactive Collaborative Environments  
**UDP** User Datagram Protocol  
**URL** Uniform Resource Locator  
**UUID** Universal Unique Identifier  
**UX** User eXperience  
**VNC** Virtual Network Computing  
**VM** Virtual Machine  
**WAN** Wide Area Network  
**WIMP** Window, Icon, Menu, Pointing device  
**WYSIWIS** What You See Is What I See  
**XHR** XMLHttpRequest

# B

## Resources and extended code extracts

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.
  apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>ch.unifr.pai.twice.mylibrary</groupId>
5   <artifactId>MyLibrary</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>MyLibrary</name>
8   <description />
9   <build>
10    <resources>
11     <resource>
12      <directory>src/main/java</directory>
13      <includes>
14       <include>*/client/*</include>
15       <include>*/public/*</include>
16       <include>*/*.gwt.xml</include>
17      </includes>
18     </resource>
19    </resources>
20    <plugins>
21     <plugin>
22      <artifactId>maven-compiler-plugin</artifactId>
23      <configuration>
24       <source>1.6</source>
25       <target>1.6</target>
26      </configuration>
27     </plugin>
28    </plugins>
29   </build>
30   <dependencies>
31    <dependency>
32     <groupId>com.google.gwt</groupId>
33     <artifactId>gwt-servlet</artifactId>
34     <version>2.4.0</version>
35     <scope>compile</scope>
36    </dependency>
```

```
38 <dependency>  
    <groupId>com.google.gwt</groupId>  
    <artifactId>gwt-user</artifactId>  
40    <version>2.4.0</version>  
    <scope>provided</scope>  
42 </dependency>  
    <dependency>  
44    <groupId>com.google.gwt</groupId>  
    <artifactId>gwt-dev</artifactId>  
46    <version>2.4.0</version>  
    <scope>provided</scope>  
48 </dependency>  
  </dependencies>  
50 </project>
```

Listing B.1: An example POM-file for library components



```
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
  XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven
  -4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
6   <groupId>ch.unifr.pai.twice.myModule</groupId>
  <artifactId>MyModule</artifactId>
8   <version>0.0.1-SNAPSHOT</version>
  <name>MyModule</name>
10  <build>
  <sourceDirectory>${basedir}/src/main/java</sourceDirectory>
12  <outputDirectory>${basedir}/src/main/webapp/WEB-INF/classes</outputDirectory>
  <resources>
14    <resource>
      <directory>${basedir}/src/main/java</directory>
16      <excludes>
        <exclude>/**/*.java</exclude>
18      </excludes>
      </resource>
20    </resources>
  <plugins>
22    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
24      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
26      <configuration>
        <source>1.6</source>
28        <target>1.6</target>
      </configuration>
30    </plugin>
    <plugin>
32      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
34      <version>2.1.1</version>
    </plugin>
36    </plugins>
  </build>
38  <dependencies>
    <dependency>
40      <groupId>com.google.gwt</groupId>
      <artifactId>gwt-servlet</artifactId>
42      <version>2.4.0</version>
      <scope>compile</scope>
44    </dependency>
    <dependency>
46      <groupId>com.google.gwt</groupId>
      <artifactId>gwt-user</artifactId>
48      <version>2.4.0</version>
      <scope>provided</scope>
50    </dependency>
    <dependency>
52      <groupId>javax.validation</groupId>
```

```
54     <artifactId>validation-api</artifactId>
55     <version>1.0.0.GA</version>
56         <classifier>sources</classifier>
57         <scope>provided</scope>
58     </dependency>
59     <dependency>
60         <groupId>ch.unifr.pai.mice.gwt.layout</groupId>
61         <artifactId>DynamicLayout_noConfig</artifactId>
62         <version>0.0.1-SNAPSHOT</version>
63     </dependency>
64 </dependencies>
65 <packaging>war</packaging>
66 </project>
```

Listing B.2: An example POM-file for module components

	<b>Sehr schwierig</b>	<b>Schwierig</b>	<b>Normal</b>	<b>Einfach</b>	<b>Sehr einfach</b>
Wie fandest du die erforderlichen Schritte um dich mit dem System zu verbinden?		<b>1</b>	<b>2</b>	<b>3</b>	<b>7</b>
Wie fandest du die Eingabe von Text mit dem System?			<b>2</b>	<b>7</b>	<b>4</b>
Wie fandest du das Verschieben der Objekte auf dem Bildschirm?		<b>3</b>	<b>4</b>	<b>4</b>	<b>2</b>
Wie schwierig war es, deinen Mauszeiger auf dem Bildschirm zu erkennen?	<b>1</b>	<b>4</b>	<b>5</b>	<b>3</b>	
Als Objekte auf dem Bildschirm verschoben wurden, reagierte das System deiner Meinung nach...	<b>3 ... flüssig</b>	<b>10 ... akzeptabel</b>	<b>0 ... verzögert</b>		
Was war schwieriger: Das Eingeben von Text oder das Verschieben von Objekten und wieso?	<b>5 Texteingabe</b>	<b>8 Verschieben von Objekten</b>			
	<b>Verschieben: bisschen verzögert, schwerfällig, erkennen welcher cursor mir gehört, Zettel schlecht loslassen nach dem Verschieben, weil der Kontakt nicht immer 100% hergestellt war, Loslassen erschwert</b>				
	<b>Texteingabe: Sichtbarkeit Text (kleine Anzeige), Eingabe sehr klein, Umstellen von Texteingabe zu Touchpad zu umständlich, noch andere Schritte notwendig um Text einzugeben</b>				
Welches Gerät hast du verwendet?	<b>11 Smart-Phone (+1 mit iPad 2 getauscht)</b> <b>2 Tablet</b>				
Welches ist die Typ-Bezeichnung des verwendeten Gerätes (z.B. Apple iPhone 4, Samsung Galaxy S3, etc.)	HTC Desire, HTC Desire Z (mit iPad 2 getauscht), iPad 2, iPhone 3G, iPhone 3GS, 5x iPhone 4, iPhone 4S, Samsung Galaxy, Samsung Galaxy S2, Samsung Galaxy Note 10.1				
Gehört das verwendete Gerät dir, ist es also dein privates Gerät?	<b>11 Ja</b>		<b>2 Nein</b>		
Besitzt du ein Smart-Phone, Tablet oder ähnliches Gerät?	<b>12 Ja</b>		<b>1 Nein</b>		
Wenn ja: Wie oft verwendest du es und wofür?	<b>11 Täglich</b>	<b>1 Ab und zu</b>	<b>0 Selten bis nie</b>		
	<b>2 Games</b>	<b>7 Telefonie</b>	<b>9 SMS</b>	<b>11 Internet</b>	
I Hast / Hättest du Bedenken dein privates Gerät in der Schule für den Unterricht einzusetzen und wenn ja, welche?	<b>2 Ja (+1 "habe keins")</b>		<b>10 Nein</b>		
	Smart-Phone, Datenverlust, "man hängt nur noch vor Elektrogeräten rum"				
J Wärest du bereit, eine zusätzliche Applikation auf dem Gerät (z.B. über den Apple-Store / Google Play) zu installieren um mit einem solchen System in der Schule zu arbeiten?	<b>11 Ja</b>		<b>1 Nein</b>		
	<b>(1: wenn es gratis wäre)</b>		<b>(1 Enthaltung)</b>		
K Würdest du ein solches System in der Schule gerne regelmässig verwenden?	<b>6 Ja</b>		<b>6 Nein (1 Enthaltung)</b>		
Was mochtest du am meisten / am wenigsten an der Benutzung des Systems?	Wenigsten: Schrift zu klein, nicht schreibgeschützt ( <i>Zettel wurden von anderen Benutzern gelöscht</i> ), kaum lesbar, wäre einfacher gewesen auf die normale Art und Weise, umständlich eher spielerisch, Der soziale Austausch der Klasse wird geschwächt, zum Teil langsam				
	Meisten: Übersicht, es ist mal etwas anderes, Dass man seine Meinng an die Wand projizieren kann. "isch gäbig", mit iPhone steuern				
Bemerkungen	Text verschwunden wenn ADD gedrückt.				

*Grey, italic annotations are made by the author and are not part of the responses*

Figure B.1.: Results of the original questionnaire in its original version

	<b>Very difficult</b>	<b>Difficult</b>	<b>Neutral</b>	<b>Simple</b>	<b>Very simple</b>
How did you find connecting to the system?		<b>1</b>	<b>2</b>	<b>3</b>	<b>7</b>
How did you find adding text in the system?			<b>2</b>	<b>7</b>	<b>4</b>
How did you find moving around objects on the screen?		<b>3</b>	<b>4</b>	<b>4</b>	<b>2</b>
How did you find recognizing your own mouse pointer on the screen?	<b>1</b>	<b>4</b>	<b>5</b>	<b>3</b>	
When you moved around objects on the screen, was it...	<b>3 ... smooth</b>	<b>10 ... ok</b>	<b>0 ... delayed</b>		
Which was more difficult, adding text or moving things around, and why.	<b>5 adding text</b>		<b>8 moving things around</b>		
<b><i>Moving things around: a little delayed, labored, finding my cursor, releasing the notes after moving, 'contact' was not 100% there, releasing a note was hard</i></b>					
<b><i>Adding text: Visibility of text (small display), text entry was very small, changing from text entry to touchpad was laborious, other steps were necessary to add texts</i></b>					
Which device did you use	<b>11 smart-phones (+1 replaced by an iPad)</b> <b>2 tablets</b>				
What is the type name of the used device (e.g. (z.B. Apple iPhone 4, Samsung Galaxy S3, etc.))	HTC Desire, HTC Desire Z ( <i>replaced by iPad 2</i> ), iPad 2, iPhone 3G, iPhone 3GS, 5x iPhone 4, iPhone 4S, Samsung Galaxy, Samsung Galaxy S2, Samsung Galaxy Note 10.1				
Does the used device belong to you – is it therefore your device?	<b>11 yes</b>		<b>2 no</b>		
Do you own a smart phone, tablet or a similar device?	<b>12 yes</b>		<b>1 no</b>		
If yes: How often do you use it and for what?	<b>11 daily</b>	<b>1 from time to time</b>	<b>0 rarely or never</b>		
	<b>2 games</b>	<b>7 telephony</b>	<b>9 SMS</b>	<b>11 internet</b>	
Do / Would you have concerns to use your private device for class in school and if so which?	<b>2 yes (+1 "I don't have one")</b>		<b>10 no</b>		
"Smart-Phone" (?), loss of data, permanent use of electronic devices					
Would you install an additional application on the device (e.g. by Apple-Store / Google Play) to work with such a system in class?	<b>11 yes</b>		<b>1 no</b>		
	<b>(1: if it would be for free)</b>		<b>(1 abstention)</b>		
Would you like to use such a system regularly in class?	<b>6 yes</b>		<b>6 no (1 abstention)</b>		
What did you like most / least when using the system?	Least: The font was too small, was not write protected ( <i>notes were deleted by others</i> ), almost impossible to read, would have been easier with the normal way, laborious – rather playful, the social interaction in the class was weakened, partly slow  Most: overview, it's something different, to be able to project the personal opinion at the wall, it's useful, to control it by the iPhone				
Additional remarks					
Text disappeared when pressing ADD					

*Grey, italic annotations are made by the author and are not part of the responses*

Figure B.2.: Translation of the results of the original questionnaire

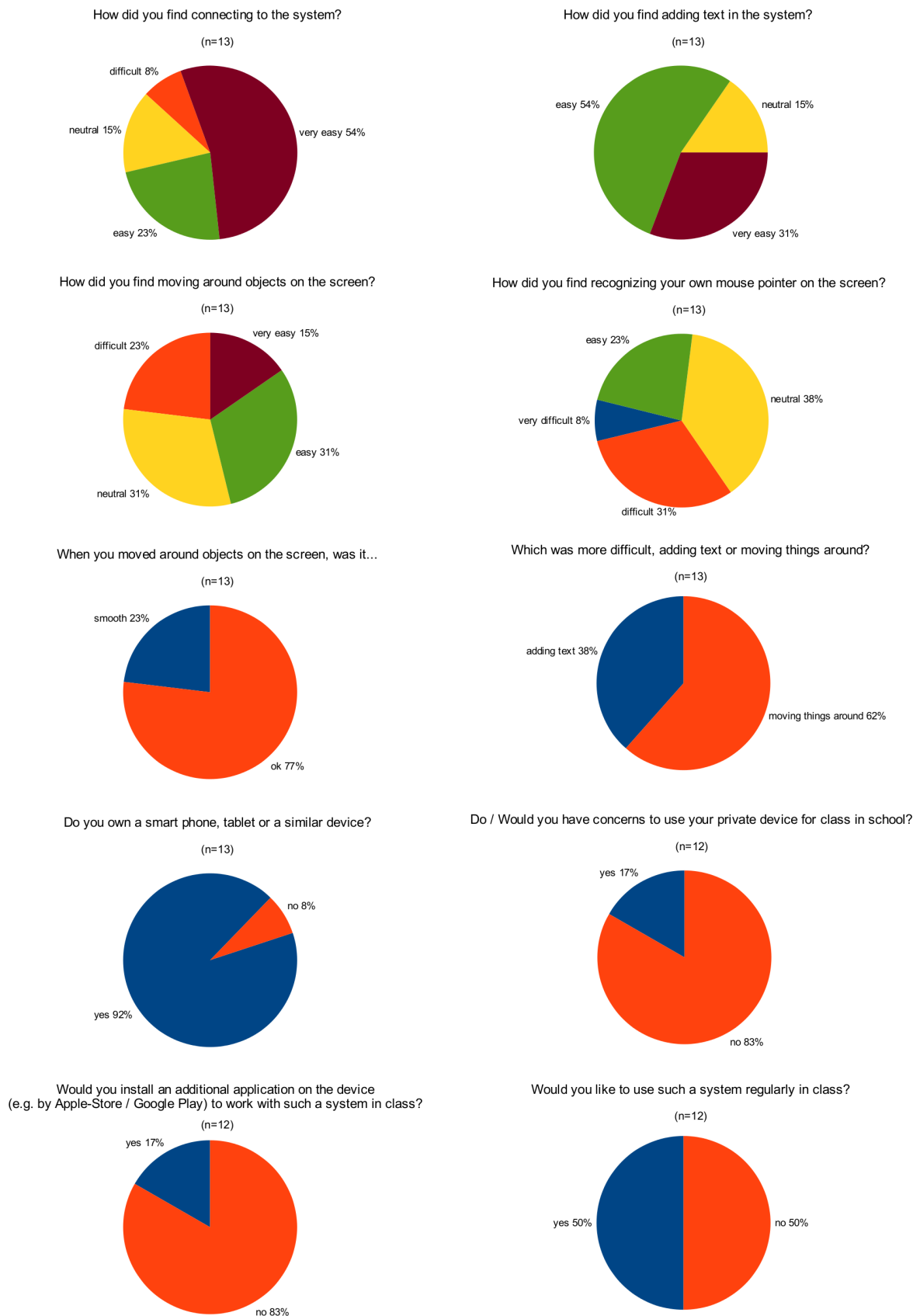


Figure B.3.: Results of the user evaluation of the system in the real world use case



## Website of the Project

A web-page was created for this project: <http://olinux.github.com/twice/>

- The API of the project.
- The binaries and sources of the toolkit code.

# Bibliography

- [1] Paulo Sérgio Almeida, Carlos Baquero, and Victor Fonte. Interval tree clocks. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*, OPODIS '08, page 259–274, Berlin, Heidelberg, 2008. Springer-Verlag.
- [2] F. Almenarez, A. Marin, D. Diaz, A. Cortes, C. Campo, and C. Garcia-Rubio. A trust-based middleware for providing security to ad-hoc peer-to-peer applications. In *Sixth Annual IEEE International Conference on Pervasive Computing and Communications, 2008. PerCom 2008*, pages 531–536, March 2008.
- [3] Magnus Ingmarsson Anders Larsson. A development platform for distributed user interfaces. pages 704–, 2007.
- [4] J. Aycock. A brief history of just-in-time. *ACM Computing Surveys (CSUR)*, 35(2):97–113, 2003.
- [5] Peter Backx, Tim Wauters, Bart Dhoedt, and Piet Demeester. A comparison of peer-to-peer architectures. In *In EURESCOM*, 2002.
- [6] Ronald M. Baecker, Jonathan Grudin, William Buxton, and Saul Greenberg. *Readings in Human-Computer Interaction: Toward the Year 2000, Second Edition*. Morgan Kaufmann, 2nd edition, April 1995.
- [7] R. Bentley, W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor, and G. Woetzel. Basic support for cooperative work on the world wide web. *Int. J. Hum.-Comput. Stud.*, 46(6):827–846, June 1997.
- [8] R. Bentley, T. Horstmann, and J. Trevor. The world wide web as enabling technology for CSCW: the case of BSCW. *Computer Supported Cooperative Work (CSCW)*, 6(2):111–134, 1997.
- [9] Sumeer Bhola, Guruduth Banavar, and Mustaque Ahamad. Responsiveness and consistency tradeoffs in interactive groupware. In *Proceedings of the 1998 ACM conference*

- on Computer supported cooperative work*, CSCW '98, page 79–88, New York, NY, USA, 1998. ACM.
- [10] Kellogg S. Booth, Brian D. Fisher, Chi Jui Raymond Lin, and Ritchie Argue. The "mighty mouse" multi-screen collaboration tool. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, UIST '02, page 209–212, New York, NY, USA, 2002. ACM.
- [11] Uwe M. Borghoff and Johann H. Schlichter. *Computer-Supported Cooperative Work: Introduction to Distributed Applications*. Springer, September 2000.
- [12] Tony Bourke. *Server Load Balancing*. O'Reilly Media, Inc., August 2001.
- [13] V. Cahill, E. Gray, J.-M. Seigneur, C.D. Jensen, Yong Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Di Marzo Seruendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielson. Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 2(3):52 – 61, September 2003.
- [14] K. Mani Chandy, Joseph Kiniry, Adam Rifkin, and Daniel Zimmerman. Webs of archived distributed computations for asynchronous collaboration. *The Journal of Supercomputing*, 11(2):101–118, 1997.
- [15] P. H. Chia, Y. Yamamoto, and N. Asokan. Is this app safe?: a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web*, page 311–320, 2012.
- [16] Eric Cronin, Anthony R. Kurc, Burton Filstrup, and Sugih Jamin. An efficient synchronization mechanism for mirrored game architectures. In *Architectures", Multimedia Tools and Applications*, page 67–73. ACM Press, 2003.
- [17] James R. Dabrowski and Ethan V. Munson. Is 100 milliseconds too fast? In *CHI '01 extended abstracts on Human factors in computing systems*, CHI EA '01, page 317–318, New York, NY, USA, 2001. ACM.
- [18] Brian de Alwis, Carl Gutwin, and Saul Greenberg. GT/SD: performance and simplicity in a groupware toolkit. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09, page 265–274, New York, NY, USA, 2009. ACM.
- [19] J. A. R. P. de Carvalho, H. Veiga, C. F. R. Pacheco, and A. D. Reis. Performance evaluation of wi-fi IEEE 802.11 a, g WPA2 PTP links: a case study. *Lecture Notes in Engineering and Computer Science*, 2198, 2012.
- [20] A. Dix and C. Sas. Mobile personal devices meet situated public displays: Synergies and opportunities. *International Journal of Ubiquitous Computing*, pages 11–28, 2010.



- 
- [21] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta. Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the 6th international conference on Intelligent user interfaces*, IUI '01, page 69–76, New York, NY, USA, 2001. ACM.
- [22] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, January 1991.
- [23] Alan W. Esenther. Instant co-browsing: Lightweight real-time collaborative web browsing. In *In Proc. Of the 11th Int*, page 7–11. Press, 2002.
- [24] Colin Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *11th Australian Computer Science Conference*, pages 55–66, 1988.
- [25] David Flanagan. *JavaScript: the definitive guide*. O'Reilly, Beijing, 6th ed edition, 2011.
- [26] Saul Greenberg, Michael Boyle, and Jason Laberge. PDAs and shared public displays: Making personal information public, and public information personal. Technical report, Personal Technologies, 1999.
- [27] Saul Greenberg and David Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, page 207–217, New York, NY, USA, 1994. ACM.
- [28] D. Grolaux, P. Van Roy, and J. Vanderdonckt. Migratable user interfaces: beyond migratory interfaces. In *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004*, pages 422 – 430, August 2004.
- [29] Carl Gutwin. The effects of network delays on group work in real-time groupware. In *In Proceedings of European Conference on Computer-Supported Cooperative Work*, page 299–318, 2001.
- [30] Carl A. Gutwin, Michael Lippold, and T. C. Nicholas Graham. Real-time groupware in the browser: testing the performance of web-based networking. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, page 167–176, New York, NY, USA, 2011. ACM.
- [31] Richard Han, Veronique Perret, and Mahmoud Naghshineh. WebSplitter: a unified XML framework for multi-device collaborative web browsing. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, CSCW '00, page 221–230, New York, NY, USA, 2000. ACM.

- 
- [32] Jason Hill and Carl Gutwin. The MAUI toolkit: Groupware widgets for group awareness. *Computer-Supported Cooperative Work*, page 5–6, 2004.
- [33] Pamela J. Hinds and Sara Kiesler. *Distributed Work*. MIT Press, May 2002.
- [34] Juan Pablo Hourcade and Benjamin B. Bederson. Architecture and implementation of a java package for multiple input devices (MID). Technical report, 1999.
- [35] Qi Huang, Daniel A. Freedman, Ymir Vigfusson, Ken Birman, and Bo Peng. Kevlar: a flexible infrastructure for wide-area collaborative applications. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, page 148–168, Berlin, Heidelberg, 2010. Springer-Verlag.
- [36] Y. Huang and C. Kintala. Software fault tolerance in the application layer. *Software Fault Tolerance*, 3:231–248, 1995.
- [37] Peter Hutterer and Bruce H. Thomas. Groupware support in the windowing system. In *Proceedings of the eight Australasian conference on User interface - Volume 64*, AUIC '07, page 39–46, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.
- [38] Hans-Christian Jetter, Michael Zöllner, Jens Gerken, and Harald Reiterer. Design and implementation of post-WIMP distributed user interfaces with ZOIL. *International Journal of Human-Computer Interaction*, 28(11):737–747, November 2012.
- [39] Brad Johanson, O. Fox, and Terry Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1:67–74, 2002.
- [40] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, page 640–651, New York, NY, USA, 2003. ACM.
- [41] Yugo Kaneda, Mika Minematsu, Masato Saito, Hiroto Aida, and Hideyuki Tokuda. AN-GEL: a hierarchical state synchronization middleware for mobile ad-hoc group gaming. In *In Proceedings of International Workshop on Pervasive Gaming Applications at Pervasive (Pergames)*, pages 30–35, 2004.
- [42] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network security: private communication in a public world, second edition*. Prentice Hall Press, Upper Saddle River, NJ, USA, second edition, 2002.
- [43] Ned Kock, editor. *Encyclopedia of E-Collaboration*. IGI Global, December 2007.
- [44] Gerd Kortuem, Jay Schneider, Dustin Preuitt, Thaddeus G. C. Thompson, Stephen Fickas, and Zary Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In *Proceedings of the First International*

- Conference on Peer-to-Peer Computing*, P2P '01, page 75–, Washington, DC, USA, 2001. IEEE Computer Society.
- [45] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, April 2008.
- [46] Denis Lalanne and Agnes Lisowska Masson. A fitt of distraction: measuring the impact of distracters and multi-users on pointing efficiency. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, page 2125–2130, New York, NY, USA, 2011. ACM.
- [47] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.
- [48] Hemang Lavana and Franc Brglez. CollabWiseTk: a toolkit for rendering stand-alone applications collaborative. In *In Seventh Annual Tcl/Tk Conference. USENIX*, 2000.
- [49] Yao-Nan Lien, Hung-Chin Jang, and Tzu-Chieh Tsai. A MANET based emergency communication and information system for catastrophic natural disasters. In *29th IEEE International Conference on Distributed Computing Systems Workshops, 2009. ICDCS Workshops '09*, pages 412–417, June 2009.
- [50] Matthew Liotine. *Mission-Critical Network Planning*. Artech House, 2003.
- [51] Marc Loy, Robert Eckstein, and Dave Wood. *Java Swing*. O'Reilly Media, 0002 edition, November 2002.
- [52] Andrés Lucero, Jaakko Keränen, and Tero Jokela. Social and spatial interactions: shared co-located mobile phone use. In *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, CHI EA '10, page 3223–3228, New York, NY, USA, 2010. ACM.
- [53] K. Luyten and K. Coninx. Distributed user interface elements to support smart interaction spaces. In *Seventh IEEE International Symposium on Multimedia*, page 8 pp., December 2005.
- [54] Jérémie Melchior, Donatien Grolaux, Jean Vanderdonckt, and Peter Van Roy. A toolkit for peer-to-peer distributed user interfaces: concepts, implementation, and applications. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09, page 69–78, New York, NY, USA, 2009. ACM.
- [55] Jérémie Melchior, Jean Vanderdonckt, and Peter Van Roy. A model-based approach for distributed user interfaces. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '11, page 11–20, New York, NY, USA, 2011. ACM.

- [56] B. Michotte and J. Vanderdonckt. GrafXML, a multi-target user interface builder based on UsiXML. In *Fourth International Conference on Autonomic and Autonomous Systems, 2008. ICAS 2008*, pages 15–22, March 2008.
- [57] David L. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39:1482–1493, 1991.
- [58] S. Mogan and Weigang Wang. The impact of web 2.0 developments on real-time groupware. In *2010 IEEE Second International Conference on Social Computing (SocialCom)*, pages 534–539, August 2010.
- [59] G. Mori, F. Paterno, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering*, 30(8):507–520, August 2004.
- [60] Meredith Ringel Morris. A survey of collaborative web search practices. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, CHI '08*, page 1657–1660, New York, NY, USA, 2008. ACM.
- [61] B.C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.
- [62] C.E. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Rocchetti. On maintaining interactivity in event delivery synchronization for mirrored game architectures. In *IEEE Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004*, pages 157–165, December 2004.
- [63] Sushil K. Prasad, Vijay Madiseti, Shamkant B. Navathe, Raj Sunderraman, Erdogan Dogdu, Anu Bourgeois, Michael Weeks, Bing Liu, Janaka Balasooriya, Arthi Hariharan, Praveen Madiraju, Srilaxmi Malladi, Raghupathy Sivakumar, Alex Zelikovsky, Yanqing Zhang, and Saied Belkasim. Syd: A middleware testbed for collaborative applications over small heterogeneous devices and data stores. In *In 5th ACM/IFIP/USENIX International Middleware Conference*, page 22, 2004.
- [64] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, December 2002.
- [65] Mark Roseman and Saul Greenberg. Building real-time groupware with GroupKit, a groupware toolkit. *ACM Trans. Comput.-Hum. Interact.*, 3(1):66–106, March 1996.
- [66] Martina Angela Sasse and Chris W. Johnson. *Human-computer Interaction, INTERACT '99: IFIP TC.13 International Conference on Human-Computer Interaction, 30th August -3rd September 1999, Edinburgh, UK*. IOS Press, 1999.

- 
- [67] Robert W. Scheifler and Jim Gettys. The x window system. *ACM Trans. Graph.*, 5(2):79–109, April 1986.
- [68] Oliver Schmid, Agnes Lisowska Masson, and Béat Hirsbrunner. Collaborative web browsing: multiple users, multiple pages, concurrent access, one display. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '12, page 141–150, New York, NY, USA, 2012. ACM.
- [69] Günter Schäfer. *Security in Fixed and Wireless Networks: An Introduction to Securing Data Communications*. John Wiley & Sons, March 2004.
- [70] Bin Shao, Du Li, Tun Lu, and Ning Gu. An operational transformation based synchronization protocol for web 2.0 applications. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, page 563–572, New York, NY, USA, 2011. ACM.
- [71] Haifeng Shen and Chengzheng Sun. Achieving data consistency by contextualization in web-based collaborative applications. *ACM Trans. Internet Technol.*, 10(4):13:1–13:37, March 2011.
- [72] Thomas Springer, Daniel Schuster, Iris Braun, Jordan Janeiro, Markus Endler, and Antonio A. F. Loureiro. A flexible architecture for mobile collaboration services. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, page 118–120, New York, NY, USA, 2008. ACM.
- [73] William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 1999.
- [74] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning, and D. Tatar. WYSIWIS revised: early experiences with multiuser interfaces. *ACM Trans. Inf. Syst.*, 5(2):147–167, April 1987.
- [75] Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Applications*. Springer, October 2005.
- [76] Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, page 59–68, New York, NY, USA, 1998. ACM.
- [77] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2 edition, October 2006.
- [78] Anthony Tang, Carman Neustaedter, and Saul Greenberg. VideoArms: embodiments in mixed presence groupware. In *In Proc. of BCS HCI*, page 85–102, 2006.
- [79] Jenifer Tidwell. *Designing Interfaces*. O'Reilly Media, Inc., December 2010.

- [80] Edward Tse and Saul Greenberg. Rapidly prototyping single display groupware through the SDGToolkit. In *Proceedings of the fifth conference on Australasian user interface - Volume 28*, AUIC '04, page 101–110, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.
- [81] Paulo Verissimo and Luís Rodrigues. *Distributed Systems for System Architects*. Springer, 2001.
- [82] Nicolas Vidot, Michelle Cart, Jean Ferrié, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. In *In Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, page 171–180, 2000.
- [83] Pedro G. Villanueva, José A. Gallud, and Ricardo Tesoriero. WallShare: a multi-pointer system for portable devices. In *Proceedings of the International Conference on Advanced Visual Interfaces, AVI '10*, page 416–416, New York, NY, USA, 2010. ACM.
- [84] James R. Wallace, Stacey D. Scott, Taryn Stutz, Tricia Enns, and Kori Inkpen. Investigating teamwork and taskwork in single- and multi-display groupware systems. *Personal Ubiquitous Comput.*, 13(8):569–581, November 2009.
- [85] Jean Walrand and Pravin Varaiya. *High-Performance Communication Networks*. Morgan Kaufmann, October 1999.
- [86] Weigang Wang. Powermeeting on common ground: web based synchronous groupware with rich user experience. In *Proceedings of the hypertext 2008 workshop on Collaboration and collective intelligence*, WebScience '08, page 35–39, New York, NY, USA, 2008. ACM.
- [87] Yao Wang and Julita Vassileva. Trust and reputation model in peer-to-peer networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing, P2P '03*, page 150–, Washington, DC, USA, 2003. IEEE Computer Society.
- [88] Daniel Wigdor, Hao Jiang, Clifton Forlines, Michelle Borkin, and Chia Shen. WeSpace: the design development and deployment of a walk-up and share multi-surface visual collaboration system. In *Proceedings of the 27th international conference on Human factors in computing systems, CHI '09*, page 1237–1246, New York, NY, USA, 2009. ACM.
- [89] J. Xu, J. Zhang, T. Harvey, and J. Young. A survey of asynchronous collaboration tools. *Information Technology Journal*, 7(8):1182–1187, 2008.
- [90] Jiang-ming Yang, Hai-xun Wang, Yi-ming Liu, and Chun-song Wang. Lock-free consistency control for web 2.0 applications. In *In Proc. 17th Intl. Conference on World Wide Web (WWW)*, page 725–734, 2008.

- 
- [91] Jin Yu, B. Benatallah, F. Casati, and F. Daniel. Understanding mashup development. *IEEE Internet Computing*, 12(5):44–52, October 2008.
- [92] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.
- [93] Philip R. Zimmermann. The official PGP user’s guide. May 1995.

## Referenced Web Resources

- [94] Appcelerator Titanium. <http://www.appcelerator.com> (accessed December 23, 2012).
- [95] The Atmosphere Framework. <https://github.com/Atmosphere/atmosphere> (accessed November 02, 2012).
- [96] Wikimedia Commons CSCW matrix. <http://en.wikipedia.org/wiki/File:Cscwmatrix.jpg> (accessed December 23, 2012).
- [97] ECMAScript. <http://www.ecmascript.org/> (accessed September 27, 2012).
- [98] The GNU Compiler Collection (GCC) web page. <http://gcc.gnu.org/> (accessed September 27, 2012).
- [99] Website for the GIMP. <http://www.gimp.org/> (accessed September 13, 2012).
- [100] Website of the GTK+ project. <http://www.gtk.org/> (accessed September 13, 2012).
- [101] Website for the Integration of MPX into GTK+. <https://live.gnome.org/GTK+/MPX> (accessed September 13, 2012).
- [102] Deferred Binding in the Google Web Toolkit (GWT). <https://developers.google.com/web-toolkit/doc/latest/DevGuideCodingBasicsDeferred> (accessed September 27, 2012).
- [103] GWT docs: Declarative Layout with UI Binder. <https://developers.google.com/web-toolkit/doc/latest/DevGuideUiBinder> (accessed November 02, 2012).
- [104] HTTP/1.1 Specification web page. <http://www.w3.org/Protocols/rfc2616/rfc2616.html> (accessed September 22, 2012).
- [105] I-Jetty: webserver for the android mobile platform. <http://code.google.com/p/i-jetty/> (accessed December 23, 2012).



- 
- [106] Java web page. <http://www.oracle.com/technetwork/java/javase/overview/index.html> (accessed September 27, 2012).
- [107] Java Mobile Edition (ME) web page. <http://www.oracle.com/technetwork/java/javame/index.html> (accessed September 27, 2012).
- [108] Java Performance Documentation web page. <http://www.oracle.com/technetwork/java/performance-138178.html> (accessed September 23, 2012).
- [109] JSON-P. <http://json-p.org/> (accessed December 26, 2012).
- [110] Maven. <http://maven.apache.org/> (accessed December 09, 2012).
- [111] Website for the Windows MultiPoint Mouse SDK. <http://www.microsoft.com/multipoint/mouse-sdk/> (accessed September 13, 2012).
- [112] Pointer Events Specification. <http://www.w3.org/Submission/pointer-events/> (accessed December 13, 2012).
- [113] OSGi Alliance. <http://www.osgi.org/> (accessed December 23, 2012).
- [114] The Perl Programming Language. <http://www.perl.org/> (accessed September 27, 2012).
- [115] PhoneGap. <http://phonegap.com/> (accessed October 07, 2012).
- [116] Python Programming Language. <http://www.python.org/> (accessed September 27, 2012).
- [117] RAW Input web page. [http://msdn.microsoft.com/en-us/library/windows/desktop/ms645536\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms645536(v=vs.85).aspx) (accessed September 22, 2012).
- [118] Ruby Programming Language. <http://www.ruby-lang.org/> (accessed September 27, 2012).
- [119] Smartphones ownership in Switzerland in March 2012. <http://www.comparis.ch/~media/files/mediencorner/medienmitteilungen/2012/telecom/verbreitung-smartphone.pdf> (accessed December 26, 2012).
- [120] Smartphones ownership in USA in September 2012. <http://pewinternet.org/Reports/2012/Smartphone-Update-Sept-2012/Findings.aspx> (accessed December 26, 2012).
- [121] Sunspider JavaScript Benchmark. <http://www.webkit.org/perf/sunspider/sunspider.html> (accessed December 22, 2012).
- [122] Tcl Developer Site. <http://www.tcl.tk/> (accessed September 27, 2012).
- [123] TWICE on Github.com. <http://olinux.github.com/twice/> (accessed December 23, 2012).

[124] Website of WebRTC. <http://www.webrtc.org/> (accessed January 16, 2012).

[125] Website of the X.Org project. <http://www.x.org/> (accessed September 13, 2012).

# Curriculum Vitae

OLIVER SCHMID

## Personal information

**Date of birth** November 22, 1984  
**Nationality** Swiss  
**Languages** German (mother tongue), English, French

## Academic qualifications

**2009 - today** PhD candidate in Computer Science, PAI Research Group, University of Fribourg

**2006 - 2009** MA in Information Management, University of Fribourg. Thesis: *“Interactive learning applications within a MHP environment”*, Supervisor: Prof. Dr. Béat Hirsbrunner

**2003 - 2006** BSc in Educational Sciences and Information Management, University of Fribourg. Thesis: *“Computergestützte Instruktion – Cognitive Apprenticeship und Anchored Instruction im Vergleich”*, Supervisor: Prof. Dr. Dr. h.c. Fritz Oser

## Professional experience

- 2009 - today** Work as Software Engineer (60-70%) at Puzzle ITC in Bern
- 2005 - 2012** Department of Educational Sciences of the University of Fribourg: Development of a media based diagnose software for further education for teachers
- 2007** Internship for the studies in Information Management at UBS Investment Bank as Junior Software Developer. Tasks: Extension of a JavaScript web application, ORM mapping (Hibernate, Ibatis) for data base access through stored procedures only, distributed caching (RMI, Jgroups, JMS, Terracotta), JMail, JMS-notification (Tibco Rendezvous) in Spring, extension of Struts application for the validation and processing of XML
- 2005** Internship for the studies in Educational Sciences at Lehrwerkstätte Bern, creation and integration of a virtual communication- and education platform
- 2000 - 2009** Freelancer: PC support, web design, small software engineering projects

## List of publications

- Schmid, O., A. Lisowska Masson, and B. Hirsbrunner, "*Collaborative Web Browsing: Multiple Users, Multiple Pages, Concurrent Access, One Display*", 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12), Copenhagen, Denmark, 06/2012
- Schmid, O., and B. Hirsbrunner, "*Middleware for distributed collaborative ad-hoc environments*", Work in Progress session at PerCom 2012 (WIP of PerCom 2012), Lugano, Switzerland, 2012.
- Bowie, M., O. Schmid, A. Lisowska Masson, and B. Hirsbrunner, "*Web-Based Multi-pointer Interaction on Shared Displays.*", Proceedings of the Interactive Papers session of the ACM Conference on Computer Supported Collaborative work (CSCW 2011)., March, 2011.

- 
- Schmid, O., A. Lisowska, M. Courant, and B. Hirsbrunner, "*Robust and reliable solutions for middle cost large multi-touch displays.*", Proceedings of the workshop on Engineering Patterns for Multi-Touch Interfaces 2010 at the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2010), June, 2010.
  - Schmid, O., M. Khadraoui, and B. Hirsbrunner, "*Video indexation by subtitles and its usage within the language learning process*", 13th IBIMA conference on Knowledge Management and Innovation in Advancing Economies, November, 2009.