Matthias Heinrich

Enriching Web Applications Efficiently with Real-Time Collaboration Capabilities

Doctoral Dissertations in Web Engineering and Web Science
Volume 1

Prof. Dr.-Ing. Martin Gaedke (Series Editor)

Matthias Heinrich

Enriching Web Applications Efficiently with Real-Time Collaboration Capabilities

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Universitätsverlag Chemnitz
2014

Imprint

Bibliographical Information of the German National Library

The German National Library lists this publication in the German National Bibliography; detailed bibliographic data is available online at http://dnb.d-nb.de.

Technische Universität Chemnitz/Universitätsbibliothek
Universitätsverlag Chemnitz
09107 Chemnitz
GERMANY
http://www.tu-chemnitz.de/ub/univerlag

Production and Distribution
Verlagshaus Monsenstein und Vannerdat OHG
Am Hawerkamp 31
48155 Münster
GERMANY
http://www.mv-verlag.de

ISSN 2199-5354 print - ISSN 2199-5362 online
ISBN 978-3-944640-25-9

http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-149948

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Enriching Web Applications Efficiently with Real-Time Collaboration Capabilities

Dissertation
zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

vorgelegt
der Fakultät für Informatik
der Technischen Universität Chemnitz

von
Dipl.-Medieninf. Matthias Heinrich
geboren am 21. August 1979 in Leisnig

Gutachter	Prof. Dr. Martin Gaedke
	Prof. Dr. Alexander Schill
	Prof. Dr. Maximilian Eibl
Eingereicht am	4. November 2013
Verteidigt am	2. April 2014

Abstract

Web applications offering real-time collaboration support (e.g. Google Docs) allow geographically dispersed users to edit the very same document simultaneously, which is appealing to end-users mainly because of two application characteristics. On the one hand, provided real-time capabilities supersede traditional document merging and document locking techniques that distract users from the content creation process. On the other hand, web applications free end-users from lengthy setup procedures and allow for instant application access. However, implementing collaborative web applications is a time-consuming and complex endeavor since offering real-time collaboration support requires two specific collaboration services. First, a concurrency control service has to ensure that documents are synchronized in real-time and that emerging editing conflicts (e.g. if two users change the very same word concurrently) are resolved automatically. Second, a workspace awareness service has to inform the local user about actions and activities of other participants (e.g. who joined the session or where are other participants working). Implementing and integrating these two collaboration services is largely inefficient due to (1) the lack of necessary collaboration functionality in existing libraries, (2) incompatibilities of collaboration frameworks with widespread web development approaches as well as (3) the need for massive source code changes to anchor collaboration support.

Therefore, we propose a Generic Collaboration Infrastructure (GCI) that supports the efficient development of web-based groupware in various ways. First, the GCI provides reusable concurrency control functionality and generic workspace awareness support. Second, the GCI exposes numerous interfaces to consume these collaboration services in a flexible manner and without requiring invasive source code changes. And third, the GCI is linked to a development methodology that efficiently guides developers through the development of web-based groupware. To demonstrate the improved development efficiency induced by the GCI, we conducted three user studies encompassing developers and end-users. We show that the development efficiency can be increased in terms of development time when adopting the GCI. Moreover, we also demonstrate that implemented collaborative web applications satisfy end-user needs with respect to established software quality characteristics (e.g. usability, reliability, etc.).

Acknowledgments

Writing this dissertation has been a demanding as well as a rewarding experience. Concluding this three-year project, I would like to thank a number of people I had the opportunity to work with.

First of all, I would like to thank my supervisor Prof. Martin Gaedke who guided me on this journey. Martin's enthusiastic and caring nature always stimulated me to set ambitious goals and to actually accomplish those goals. The friendly and challenging atmosphere established by Martin nurtured confidence to successfully tackle unprecedented tasks and endeavors. I am also grateful to Prof. Alexander Schill for supporting me in his role as a co-advisor and for promoting the fruitful cooperation with the chair of computer networks that resulted in numerous supervisions of award-winning thesis projects.

Dr. Johannes Meinecke and Dr. Thomas Springer also deserve a great deal of thanks for mentoring me during the entire dissertation period. In our bi-weekly meetings Johannes insisted on, consistently repeated and finally successfully conveyed the idea that good research starts out with structure. Moreover, Johannes diligently read a myriad of drafted publications as well as dissertation chapters and always accurately and sometimes painfully disclosed deficiencies. Being able to do dozens of things in parallel, Thomas not only coached me and co-authored publications, but we also managed to supervise four thesis projects cooperatively. I will certainly miss our bi-weekly research discussions and our paper-acceptance celebrations.

I would also like to thank the great group of students I could closely work with throughout the last three years. Besides contributing to prototypical implementations and writing excellent master theses, Franz Lehmann, Franz Josef Grüneberger and Philipp Hauer have been a true inspiration for my work on real-time collaboration support. On a daily basis, we discussed related work, prototypes, publications, patents or their thesis documents which represented a vastly intense and an immensely productive time. Franz, Franz Josef and Philipp showed extraordinary passion and they are exceptional people I really enjoyed working with.

Furthermore, I would like to express my gratitude to my colleagues Dr. Thomas Hettel, Dr. Tobias Nestler, Dr. Steffen Göbel and Dr. Jochen Rode. Thomas was the former lead architect of SAP Gravity, which rep-

resents SAP's operational transformation engine, and he actually sparked my interest for the research field of real-time collaboration. Tobias in his role as the OMELETTE project lead always strived to embrace the topic of real-time collaboration which allowed me to largely focus on my research interests. Steffen and Jochen gave me management support and trust which was critical in order to finish this dissertation.

Likewise, I am thankful to the entire VSR research group including Dr. Jörg Anders, Olexiy Chudnovskyy, Hendrik Gebhardt, Sebastian Heil, Michael Krug, Ralph Sontag, Luise Steinbach, Frank Weinhold, Fabian Wiedemann and Stefan Wild. They all have been extremely supportive managing technical or organizational issues. But more importantly, I immediately felt welcome in their research group.

A huge thanks goes to my friends. They have put up with me not meeting obligations when a paper deadline approached. They cheered me up when a notification about a rejected paper arrived. They brought me down to earth when I was floating in some research cloud. Particularly, I thank Joachim and Matthias who accepted to proofread the entire dissertation.

Finally, I would like thank my family and Kathi. Without your constant support, this dissertation would simply not have been possible. I owe a special thanks to my sister Manuela who has thoroughly proofread a multitude of research articles.

Publications

Material, ideas and figures from this dissertation have appeared previously in the following publications.

Refereed Journal and Conference Articles

[1] Matthias Heinrich, Franz Lehmann, Franz Josef Grüneberger, Thomas Springer, Martin Gaedke, and Alexander Schill. Enriching Single-User Web Applications Non-Invasively with Shared Editing Support. *Science of Computer Programming*, 2013.

[2] Matthias Heinrich, Franz Lehmann, Thomas Springer, and Martin Gaedke. Exploiting Single-User Web Applications for Shared Editing – A Generic Transformation Approach. In *International World Wide Web Conference (WWW '12)*, pages 1057-1066, 2012.

[3] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Exploiting Annotations for the Rapid Development of Collaborative Web Applications. In *International World Wide Web Conference (WWW '13)*, pages 551-560, 2013.

[4] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Reusable Awareness Widgets for Collaborative Web Applications – A Non-invasive Approach[1]. In *International Conference on Web Engineering (ICWE '12)*, pages 1-15, 2012.

[5] Matthias Heinrich and Martin Gaedke. Data Binding for Standard-based Web Applications. In *ACM Symposium on Applied Computing (SAC '12)*, pages 652-657, 2012.

[6] Matthias Heinrich, Franz Lehmann, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Analyzing the Suitability of Web Applications for a Single-User to Multi-User Transformation. In *International World Wide Web Conference (WWW '13)*, pages 249-252, 2013.

[1] This publication received the *Best Research Paper Award* at the International Conference on Web Engineering 2012.

[7] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, Philipp Hauer, and Martin Gaedke. GAwI: A Comprehensive Workspace Awareness Library for Collaborative Web Applications[2]. In *International Conference on Web Engineering (ICWE '13)*, pages 482-485, 2013.

[8] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Enriching Web Applications with Collaboration Support Using Dependency Injection. In *International Conference on Web Engineering (ICWE '12)*, pages 473-476, 2012.

[9] Matthias Heinrich and Martin Gaedke. WebSoDa: A Tailored Data Binding Framework for Web Programmers Leveraging the WebSocket Protocol and HTML5 Microdata. In *International Conference on Web Engineering (ICWE '11)*, pages 387-390, 2011.

Industry Articles

[10] Matthias Heinrich and Thomas Hettel. Synchronisationsalgorithmen verstehen – Das Geheimnis der Echtzeit-Kollaboration. *heise Developer*, 2012.

[11] Matthias Heinrich and Thomas Hettel. Browser-Entwicklungsumgebungen im Vergleich – Desktop ade, tut scheiden weh? *iX Developer – Programmieren heute*, 2012.

Patents

[12] Matthias Heinrich and Franz Lehmann. Universal Collaboration Adapter for Web Editors (Application Number 13/304,418; Reference Number 2011P00293US), 2011.

[13] Matthias Heinrich and Franz Josef Grüneberger. Generic Workspace Awareness Support for Collaborative Web Applications (Application Number 13/490,058; Reference Number 2012P00399US), 2012.

[14] Matthias Heinrich and Franz Josef Grüneberger. Collaboration Adapter for Single-User Model-View-Controller Applications (Application Number 13/892,305; Reference Number 130186US01), 2013.

[2] This publication received the *Best Demo and Poster Award* at the International Conference on Web Engineering 2013.

Contents

Chapter 1

Introduction

In the last decade, the World Wide Web evolved from a document provisioning infrastructure to an application platform giving rise to interactive web applications ranging from basic text editors to sophisticated office suites, multi-player games or even development environments. In contrast to traditional application platforms, the highly distributed nature of the World Wide Web reaching billions of people represents a unique opportunity to foster collaboration. Examples like Facebook, Google Docs or Wikipedia demonstrate that software corporations as well as nonprofit organizations leverage this unmatched collaboration potential to create web-based solutions that are now used by hundreds of millions of end-users.

1.1 Real-Time Groupware

Synchronous distributed interaction is a specific collaboration form facilitated by real-time groupware [1] that allows geographically dispersed users to edit the very same document simultaneously. Google Docs [2], for example, is one prominent real-time groupware application enabling numerous collaborators to author a rich text document in parallel. Thereby, real-time groupware systems have to expose two crucial capabilities, namely, *concurrency control* and *workspace awareness*. Concurrency control, on the one hand, ensures that all users work on a consistent document [3] which includes assuring that document copies are synchronized and that editing conflicts are resolved (e.g. if two users change the very same word concurrently). On the other hand, workspace awareness defined as "the up-to-the-moment understanding of another person's interaction with the shared workspace" [4] sheds light on activities of remote users (e.g. who joined the session or where are other participants working).

Real-time groupware represents an established research discipline pioneered by Douglas C. Engelbart, who demonstrated the first multi-user text editor in 1968 at the Stanford Research Institute in Menlo Park [5]. Afterwards, even though the globalization era entailed a strong need for capable collaborative applications in multinational enterprises and organizations, real-time groupware systems remained to a large extent a research topic. Various research groups devised advanced concurrency control algorithms (e.g. operational transformation [3], differential synchronization [6]), explored new approaches to support workspace awareness (e.g. telepointers [7], radar views [8], creation coloring widgets [4]) or investigated novel groupware solutions (e.g. GROVE [9], CoWord [10]). In 2004, after more than thirty years of diligent research, Noël and Robert stated that Computer-Supported Cooperative Work (CSCW) systems in general have not been successful [11]. The identified problems were mainly installation issues [12] and the adoption barrier asking users to invest substantial effort to learn the new groupware applications [13].

Leveraging the web as application platform could represent an inflection point in the adoption of groupware systems because of two reasons. First, end-users no longer have to carry out time-consuming setup procedures. And second, end-users are already familiar with basic browser interaction patterns (e.g. browser navigation) which lowers the adoption hurdle [14, 15]. Moreover, prominent examples support the assumption that the usage of real-time groupware increases. For instance, the Google Docs office suite is ranked number 68 in Netcraft's most visited web sites statistic [16].

1.2 Motivation

The demand for real-time groupware, in particular, for web-based solutions offering ease of consumption and a minimal adoption hurdle, is constantly growing within as well as across organizations. This trend is reinforced by three major factors: *operational savings*, *end-user demand* and *the plethora of use cases*.

Operational savings can be realized replacing legacy systems that hamper collaborative work. For example, collaboratively authoring documents using traditional single-user applications (e.g. Microsoft Word) requires the use of revision control systems distracting knowledge workers from the actual document creation process. If revision control is not available and document versions are distributed for instance via email, the collaboration efficiency is reduced even further. In 2012, management consulting firm McKinsey estimated that the use of social communication and collaboration technologies in enterprises can unlock up to $860 billion in annual savings [17].

Besides economical drivers, the majority of end-users are also asking for multi-user capabilities in their office application of choice. For instance, the

Forrester Research report "A Look at the Improvements and Shortcomings of Microsoft Office 2007 Desktop Applications" [18] reveals that 51 percent of surveyed end-users want improved collaboration support, which represents the number one feature request. The "Empirical Study on Collaborative Writing" [11] yielded similar results disclosing that synchronous document access is the most requested functionality when asking participants for the ideal collaborative writing tool.

A third factor contributing to the growing demand for web-based groupware is the plethora of use cases for collaborative work. Apparently, the most common domain is collaborative writing that is adopted for producing project proposals, scientific publications, etc. Taking into account that a large part of documents is created by more than one author (e.g. a study carried out by Ede and Lunsford identified that 87 percent of all documents had at least two authors [19]) emphasizes the need for collaborative word processors. Besides the established collaborative writing domain, further use cases range from jointly authoring spreadsheets and presentations to collaboratively creating source code files, CAD models or even circuit diagrams.

1.3 Problems

The central problem that is addressed by this dissertation is the inefficiency in the development process of web-based real-time groupware. Efficient development processes and methodologies have been devised for the specific field of web engineering [20]. In particular, web engineering research explored advanced model-driven approaches (e.g. WebML [21], UWE [22], OOHDM [23]) as well as capable component-based approaches (e.g. WebComposition [24, 25, 26]). These web engineering approaches are tailored for data-driven web applications where data repositories are leveraged to construct conventional web pages on the server-side. However, these high-level approaches are not suited capturing the required client-side interactivity of real-time groupware.

Client-side interactivity is commonly implemented leveraging client-side JavaScript libraries (e.g. jQuery [27]). Nevertheless, low-level JavaScript libraries are also not able to promote development efficiency to its full extent due to (1) the lack of out-of-the-box workspace awareness support, (2) the deficiencies of concurrency control libraries and (3) the invasiveness of existing approaches, i.e. adopting a library entails changing the application's source code. In the following, we will elaborate on the three identified problems:

1. Groupware applications necessitate dedicated workspace awareness widgets such as participant lists, telepointers, radar views, etc., to inform the local user about the actions of other users in the shared space. Collaborative web applications like Google Docs [2], Etherpad [28],

SAP Process Flow [29], etc., incorporate a set of workspace awareness widgets that mostly overlaps (e.g. each groupware application requires a participant list). However, there is neither a comprehensive awareness widget library for the web nor a systematic approach how to adequately couple the actual application with a workspace awareness infrastructure. Hence, each web-based groupware development endeavor entails implementing a set of awareness widgets including a platform to distribute awareness information (e.g. the new coordinates of a telepointer). Re-implementing basic components and maintaining numerous components of the same type (e.g. various participant lists) unnecessarily inflates development expenditures as well as maintenance costs.

2. Besides the lack of proper workspace awareness support, implementing concurrency control in new groupware development projects also induces major implementation effort that unnecessarily reduces development efficiency. First, creating web-based groupware applications from scratch commonly requires the use of general-purpose web frameworks (e.g. jQuery [27], Knockout [30], etc.). These general-purpose web frameworks do not support concurrency control at all and thus, web developers have to get familiar with yet another concurrency control library (e.g. ShareJS [31] or Apache Wave [32]). Second, incompatibilities of development approaches limit the use of concurrency control libraries. For example, Apache Wave relies on the Google Web Toolkit development approach [33] urging programmers to implement their web applications in Java and neglecting the native web programming language JavaScript. Third, the offered feature set of concurrency control libraries does not always meet the requirements of common groupware applications. For instance, ShareJS cannot synchronize graph data models which is required for collaborative business process editors like SAP Process Flow [29].

3. A third efficiency problem emerges when existing single-user web applications should be leveraged for collaborative work and hence, are subject to a single-user to multi-user transformation. Enhancing single-user applications with multi-user capabilities is a promising approach taking into account the familiarity end-users already gained using the original single-user application and considering the myriad of existing single-user web applications (e.g. the Chrome Web Store [34] lists thousands of single-user applications). However, introducing concurrency control and workspace awareness features is currently by no means efficiently supported since collaboration libraries entail the need for invasive source code changes. Having to change the editor's implementation requires getting acquainted with the application's source

code which is a time-consuming and complex task. This is particularly true for mature single-user web applications such as the text editor CKEditor [35] consisting of 110 000 lines of JavaScript code or the graphics editor SVG-edit [36] encompassing more than 30 000 lines of code.

1.4 Research Objectives

The goal of this dissertation is to increase development efficiency in the process of devising and implementing real-time groupware for the web. The development efficiency goal is divided into the following three concrete research objectives:

1. Analyze to what extent workspace awareness support for web-based groupware can be supplied in a reusable and non-invasive fashion.

2. Investigate and devise efficient means for integrating concurrency control support into widely-adopted web development frameworks to ease groupware development.

3. Explore generic single-user to multi-user transformation approaches that allow to non-invasively incorporate multi-user capabilities into existing single-user web applications.

1.5 Research Contributions

Besides carving out a methodology for enhancing web applications with workspace awareness and concurrency control support, the main contributions are threefold and in line with the listed research objectives.

- Research objective 1 is addressed by identifying web application commonalities that allow anchoring an awareness system in a non-invasive manner, i.e. the source code of the original web application is not subject to changes. These web application commonalities are leveraged to design and implement a *workspace awareness adapter* including a set of ready-to-use workspace awareness widgets (e.g. telepointer, participant list, radar view, etc.). Furthermore, the non-invasiveness and the reuse properties of the proposed approach are validated by integrating the workspace awareness adapter into numerous single-user web applications (the graphics editor SVG-edit [36] or the text editor CKEditor [35]) and by verifying functional completeness. Additionally, a usability study encompassing 20 participants shows the suitability of the workspace awareness adapter for collaborative work.

- Research objective 2 is embraced by analyzing a variety of general-purpose web frameworks (e.g. Backbone [37], Knockout [30], etc.) with the goal of finding exploitable similarities for a lightweight concurrency control integration. Similarities serve as the foundation for facilitating concurrency control generically in general-purpose web frameworks. Eventually, combining popular web development frameworks with concurrency control capabilities eases the task of implementing web-based groupware applications. We demonstrate the feasibility of the approach by implementing the so called *framework adapter* that enhances existing web frameworks (e.g. Knockout [30] or SAPUI5 [38]) with concurrency control support. Furthermore, we show the efficiency induced by the framework adapter approach by conducting a developer study that compares a conventional programming library with the proposed framework adapter approach.

- Research objective 3 is approached by advancing the transparent adaptation approach [39] that allows for single-user to multi-user transformations, i.e. collaboration capabilities are introduced in a non-invasive fashion. However, the application-specific nature of the transparent adaptation approach entails re-implementing collaboration system components. Thus, we generalize this approach for standards-based web applications that are built on top of the Document Object Model (DOM) [40]. To this end, we devise the *DOM adapter* which frees programmers from the time-consuming task of re-implementing concurrency control components. To validate our approach, we transform various single-user web editors (CKEditor [35] and SVG-edit [36]) applying the DOM adapter implementation. Moreover, a usability study with 30 participants assesses typical software quality characteristics of converted editors and demonstrates end-user acceptance.

1.6 Structure

This dissertation is structured as depicted in Figure 1.1. While Chapters 1–3 introduce the topic of web-based real-time groupware, Chapters 4–7 present the main research contributions and Chapters 8–9 evaluate and summarize these contributions. Subsequently, we briefly elaborate on the content of each individual chapter.

Chapter 2 establishes the groupware terminology, introduces relevant technical foundations and motivates the research objectives introducing two exemplary use cases. While the first use case illustrates a transformation scenario where an existing single-user web editor is converted into a multi-user version, the second use case demonstrates the from-scratch implementation of web-based groupware. Taking into account the two use cases, we highlight essential characteristics of web-based real-time groupware and derive

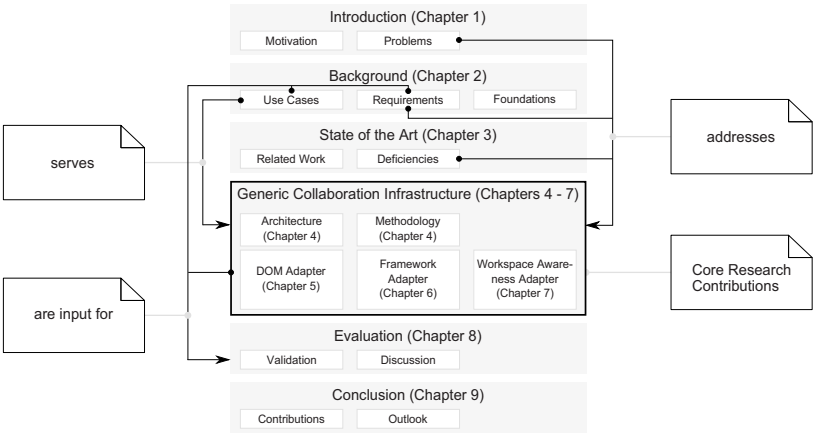


Figure 1.1: Thesis outline

requirements for collaboration components that allow to efficiently support the from-scratch development as well as the single-user to multi-user transformation.

Chapter 3 discusses the state of the art and identifies associated deficiencies. The state of the art discussion includes groupware development libraries and frameworks, transformation approaches as well as web engineering approaches. Leveraging the set of requirements compiled in Chapter 2, we assess to what extent the considered development approaches are suitable to efficiently develop web-based groupware.

Chapter 4 introduces a collaboration system called Generic Collaboration Infrastructure (GCI) that is devoted to increase development productivity for web-based groupware. The GCI encompasses three main components: (1) the DOM adapter, (2) the framework adapter and (3) the workspace awareness adapter. While Chapter 4 briefly introduces these components, Chapters 5–7 are dedicated to discuss these GCI components in detail. In addition to the GCI introduction, a groupware development methodology is established supporting developers to select efficient means when implementing collaborative web applications.

Chapter 5 elaborates on the DOM adapter that allows introducing concurrency control capabilities non-invasively into existing single-user web applications. The DOM adapter discussion focuses on the architecture and a usability study using two converted editors. Moreover, the relevance of the approach is analyzed, i.e. we investigate how many existing applications can leverage the DOM adapter approach.

Chapter 6 specifies the framework adapter that supports injecting concurrency control capabilities when developing web-based groupware from-

scratch. Thereby, we present two viable approaches: the wrapper approach and the annotation-based approach. While the wrapper approach enhances an existing single-user API with multi-user functionality, the annotation-based approach adopts source code annotations to tag data model elements that should be synchronized. Finally, a developer study compares the framework adapter in terms of efficiency with a traditional concurrency control library.

Chapter 7 sheds light on the workspace awareness adapter. In contrast to the DOM adapter and the framework adapter anchoring concurrency control support, the workspace awareness adapter is in charge of incorporating workspace awareness features that represent the second crucial collaboration service. Besides exploring the architecture, we expose the widget library comprising six awareness widgets and we discuss a usability study showing the suitability of the awareness adapter for collaborative work.

Chapter 8 represents the overall evaluation. On the basis of the defined requirements in Chapter 2, we assess to what extent the GCI is appropriate for implementing collaborative web applications.

Chapter 9 summarizes the main research contributions and draws conclusions. Additionally, we provide an outlook regarding future work in the field of web-based groupware development.

Chapter 2

Background

In this chapter, we discuss essential background aspects regarding the development of web-based real-time groupware covering concrete uses cases, associated challenges, technological foundations and derived efficiency requirements. To focus the foundations discussion on concrete examples and to illustrate the problem scope, we first introduce two exemplary use cases adopting established development approaches for real-time groupware. These use cases, highlighting the *from-scratch development* as well as *the single-user to multi-user transformation*, are revisited throughout this dissertation to clarify encountered research challenges and to convey proposed solutions. Moreover, we describe the foundations of web-based real-time groupware including their distinct multi-user capabilities (concurrency control and workspace awareness) as well as their architectural building blocks imposed by the web application platform. We conclude the chapter with a definition of requirements for an efficient groupware development approach.

2.1 Groupware Development Use Cases

Use cases are a descriptive means to explore research challenges, demonstrate proposed approaches, verify anticipated benefits and highlight entailed limitations. Hence, we define two differing use cases for the development of web-based real-time groupware and constantly revisit them to clarify issues or to transform theoretical questions into a tangible context.

The development of web-based real-time groupware can be divided into two main scenarios: the from-scratch development as well as the single-user to multi-user transformation. While the from-scratch development assumes that the demand for multi-user capabilities is already known at the beginning of the software development lifecycle, the transformation approach incorporates multi-user capabilities as an evolutionary feature at a later phase of the software development lifecycle. Figure 2.1 depicts the coarse-grained development lifecycle according to Gaedke [26] and highlights the phases

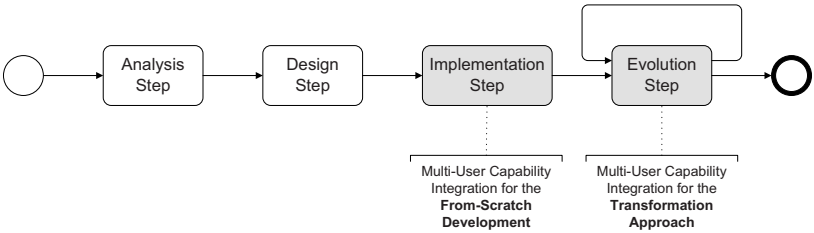


Figure 2.1: The software development lifecycle and anchor points to incorporate multi-user capabilities

where multi-user facilities can be incorporated. On the one hand, the implementation phase is the dedicated step to anchor multi-user features in from-scratch development projects; on the other hand, the evolution step is devoted to integrate multi-user features in transformation projects.

In the following, we define one concrete use case for each of the two prevalent development approaches for real-time groupware and identify associated inefficiency challenges that developers nowadays face.

2.1.1 SVG-edit Transformation: Converting the Single-User Editor into a Multi-User Version

The large majority of web applications, offered for instance in the Chrome Web Store [34] or the Firefox Marketplace [41], support solely single-user scenarios even though the targeted application domain (e.g. text editing) is also suited for collaborative work. For example, there are web-based graphics editors (SVG-edit [36]), word processors (CKEditor [35], TinyMCE [42]), development environments (Eclipse Orion [43]) and circuit modelers (CircuitLab [44]) that lack native multi-user support. To unfold the potential of single-user web applications for collaborative work, another evolution step is necessary to incorporate shared editing capabilities (concurrency control and workspace awareness). In this use case discussion, we leverage the SVG-edit graphics editor to explain why multi-user capabilities are of value and what requirements have to be implemented to adopt SVG-edit for collaborative work.

The SVG-edit Graphics Editor

The graphics editor SVG-edit [36] is a popular single-user web application that allows creating scalable vector graphics offering an intuitive user interface (cf. Figure 2.2). Just like established vector graphic tools for the desktop (e.g. Adobe Illustrator [45] or Inkscape [46]), SVG-edit supports draw operations for various kinds of shapes such as lines, circles, rectangles

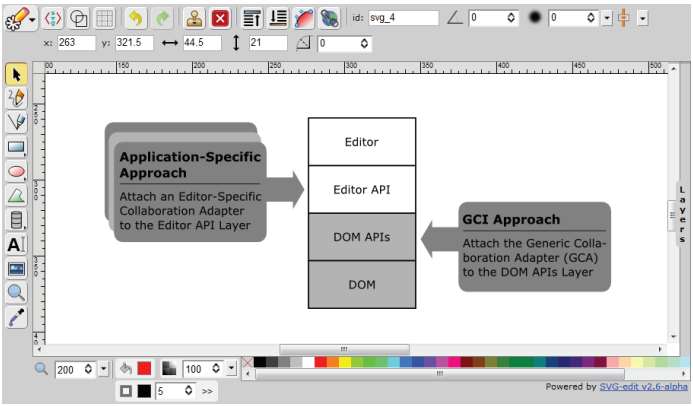


Figure 2.2: A screenshot of the single-user graphics editor SVG-edit [36]

or polygons. Additionally, SVG-edit also offers a variety of shape manipulation tools to resize, fill, move or group graphical objects. To distribute created graphics, SVG-edit allows storing documents in the standardized Scalable Vector Graphics (SVG) format [47].

SVG-edit was initiated as an open-source project in 2009 and today a team of approximately 20 committed individuals is maintaining and advancing the graphics editor. The latest stable release (SVG-edit version 2.5.1) surpassed 45 000 downloads demonstrating the broad adoption of the single-user editor. SVG-edit is primarily implemented in JavaScript and the current code base adds up to more than 30 000 lines of code representing a rather large project in the web application ecosystem.

Functional Requirements for the Collaborative SVG-edit

A collaborative SVG-edit could broaden the application's scope and promote the editor's proliferation by allowing end-users to collaboratively sketch floor plans, draw application mockups, capture ideas in mind maps, etc. Multiple users could edit the very same graphic in parallel without being distracted by requesting edit tokens or merging multiple document versions. Leveraging a multi-user SVG-edit that supports synchronous collaboration would allow users to focus on the collaborative graphic creation process without having to deal with technology deficiencies.

Converting the single-user SVG-edit into a collaborative version requires adding numerous features. To formally record these features, we specify functional requirements. These include Concurrency Control Requirements (CCR) as well as Workspace Awareness Requirements (WAR). We refer to the converted SVG-edit as *the system*.

- CCR1: The system shall allow users to join or leave a collaborative session at any point in time.
- CCR2: The system shall allow users of a collaborative session to employ all original single-user tools (e.g. create, move or resize shape tools).
- CCR3: The system shall allow users to manipulate the shared document in an unconstrained fashion (i.e. the document manipulation of one user must not limit the document manipulation options of other users).
- CCR4: The system shall synchronize all document manipulations of all users sharing the same session.
- CCR5: If two document manipulations are in conflict (e.g. two users change the fill color of the very same rectangle at the same time), the system shall automatically resolve these editing conflicts.
- WAR1: The system shall present all participants of the collaborative session in a dedicated participant list widget.
- WAR2: The system shall show further presence information in form of a creation coloring widget, telepointer and radar view.

In addition to the application-specific functional requirements, software quality characteristics such as reliability, usability, efficiency, etc., also have to be considered and assessed. Various established software quality models capture these aspects (e.g. ISO/IEC 25010 standard [48]) and therefore, we will omit a detailed discussion of software quality models. However, in the evaluation of the collaborative SVG-edit we will also consider these quality characteristics (cf. Section 5.3).

2.1.2 CoBAT From-Scratch Development: Implementing a Collaborative Cost-Benefit Analysis Tool (CoBAT)

Besides the transformation scenario, we identified a second use case representing the from-scratch development. We selected an exemplary application supporting the decision making process for major investments. Larger investments like introducing a customer relationship management system or purchasing a truck for the company's transportation fleet have to be reviewed systematically. A Cost-Benefit Analysis (CBA) application supports this process by analyzing positive and negative factors. Moreover, a CBA tool also allows comparing various investments and identifying the most profitable one.

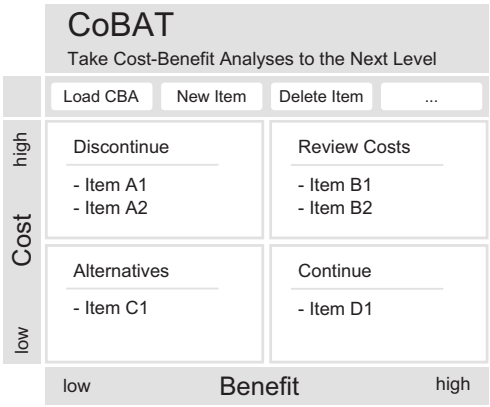


Figure 2.3: A mockup of the web-based CoBAT editor

In the following, we introduce the envisioned Cost-Benefit Analysis Tool (CoBAT). This includes a mockup of a UI sketch as well as a specification of the functional requirements.

The Envisioned CoBAT Editor

CoBAT should represent a simple collaborative web application[1] that provides an overview of all influencing CBA factors supporting an investment decision. Collecting crucial factors and quantifying their associated costs and promised benefits are the main CoBAT tasks. Since assembling influencing factors is a highly collaborative assignment with various people contributing, end-users should be able to simultaneously edit the CoBAT document. In particular, larger investments rely on the input and feedback of a variety of stakeholders which emphasizes the need for real-time collaboration.

From a technology point of view CoBAT is supposed to shed light on the development of web-based real-time groupware and thus, the application should be built exclusively on top of standardized web technologies (e.g. HMTL5 [49], JavaScript 5.1 [50], etc.).

A CoBAT mockup is depicted in Figure 2.3 dividing the application into a toolbar section and a cost-benefit matrix. While the toolbar provides convenient tools to load existing matrices, create or delete new CBA factors,

[1] Designing this from-scratch development use case, we consciously selected a moderate-sized development scenario since CoBAT should serve as a means to analyze development efficiency for web-based real-time groupware. Thus, a thorough evaluation requires that numerous programmers individually develop the application using multiple comparable approaches. Urging developers to implement an application using various approaches requires limiting the application's feature set due to the limited evaluation resources.

etc., the 2 x 2 matrix gives an overview of all considered factors. Each matrix cell can accommodate numerous items and may contain a heading. Additionally, all textual content (e.g. headings, CBA factors, etc.) may be edited in a collaborative manner.

Functional Requirements for CoBAT

To clarify the anticipated CoBAT functionality, we again specify the functional requirements. In addition to concurrency control and workspace awareness requirements, the from-scratch development also includes Single-User Feature Requirements (SUFR).

1. SUFR1: The system shall allow users clustering CBA factors in a 2 x 2 matrix.

2. SUFR2: The system shall allow users to add, remove and reorder CBA factors.

3. SUFR3: The system's user interface shall contain a CBA matrix including a heading and matrix axes labels as well as a heading for each matrix cell.

4. CCR1: The system shall allow users to join or leave a collaborative session at any point in time.

5. CCR2: The system shall allow users of a collaborative session to simultaneously edit CBA factors, labels and headings as well as to reorder and move CBA factors concurrently.

6. CCR3: The system shall synchronize all document manipulations of all users sharing the same session.

7. CCR4: If two editing operations are in conflict (e.g. two users change the very same heading), the system shall automatically resolve these editing conflicts.

8. WAR1: The system shall present all participants of the collaborative session in a dedicated participant list widget.

Note that we recorded the key single-user feature requirements for the sake of completeness. In essence, we focus on assessing efficiency for the development of collaborative web applications which primarily analyzes the implementation of concurrency control and workspace awareness. Again, software quality requirements that are for example documented in the ISO/IEC 25010 standard [48] are neglected in this section.

2.1.3 Inefficiency Challenges

Both presented use cases, the SVG-edit single-user to multi-user transformation as well as the from-scratch CoBAT development, require to incorporate concurrency control and workspace awareness features. Incorporating these multi-user capabilities results in major development effort using conventional development approaches. We identified the following crucial development challenges that drive inefficiency in conventional development projects for web-based real-time groupware.

Massively Invasive Integration: Modern concurrency control libraries adopt the notion of anchoring multi-user capabilities directly in the single-user source code which leads to a multitude of source code changes. For example, in order to integrate concurrency control, each function implementation modifying the application's data model has to be adapted to ensure that the concurrency control system is informed about model changes. Thus, leveraging a conventional library to synchronize SVG-edit or CoBAT model instances might incur hundreds or even thousands of source code changes. Since the number of code changes is commonly proportional to the lines of code, large-size projects like, for instance, the open-source project SVG-edit with more than 30 000 lines of code, require a particularly high investment to integrate multi-user capabilities.

Lack of Separation of Concerns: Modern libraries for groupware applications promote intermingling single-user and multi-user capabilities in the application's implementation which contradicts the separation of concerns principle and eventually increases software maintenance costs. For instance, if a concurrency control library has to be replaced by another concurrency control library, not only numerous source code changes have to be accomplished but due to the scattered nature of these code changes various software components are typically affected. In contrast to maintaining one encapsulated concurrency control and one separated workspace awareness module, updating a variety of software components is time-consuming and increases maintenance effort.

Substantial Familiarization Effort: Libraries offering multi-user capabilities are commonly *special-purpose* libraries, i.e. web developers that are acquainted with general-purpose web frameworks like jQuery [27], Knockout [30], etc., cannot leverage their existing knowledge. Instead, they have to learn an extra programming library providing concurrency control and workspace awareness functionality. Additionally, for the single-user to multi-user transformation approach, developers have to get familiar with the editor's code base if they were not involved in the initial single-user implementation project. This entails substantial familiarization effort, in particular, for editor's like SVG-edit that consist of a large code base encompassing 30 000 lines of JavaScript code.

Neglected Workspace Awareness Reuse: As of today, workspace awareness frameworks are not tailored for the web application platform, i.e. their functionality cannot be leveraged by standards-based web applications. Hence, workspace awareness widgets (e.g. participant lists, telepointers, radar views, etc.) as well as the underlying infrastructure (distributing awareness information, etc.) have to be re-implemented from-scratch for each and every web-based groupware application which unnecessarily increases development expenses.

2.2 Foundations

The presented use cases demonstrate the crucial technology fields for the development of web-based real-time groupware: concurrency control, workspace awareness and web technology. To establish a common terminology and understanding, we introduce the foundations of the three pillars of web-based real-time groupware.

2.2.1 Concurrency Control

Groupware applications support a variety of interaction forms that are compiled in the time-space groupware matrix [51] in Figure 2.4. Concurrency control techniques are necessary for the highlighted class of *synchronous remote* groupware systems and we will refer to this application class as *real-time groupware*. This section defines concurrency control for real-time groupware, summarizes concurrency control strategies and introduces the operational transformation algorithm which is the prevalent concurrency control mechanism.

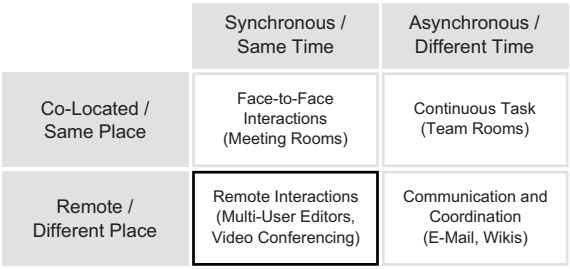


Figure 2.4: Time-space groupware matrix [51]

Concurrency control originates from the domain of database management systems [52, 53] and allows handling conflicts emanating from concurrent database operations (e.g. simultaneous write access). The objective of concurrency control mechanics is to preserve data integrity. Applied to real-time groupware, we give the following concurrency control definition.

Definition 2.1 – Concurrency Control: *Concurrency Control for real-time groupware is a means to automatically resolve editing conflicts emerging from simultaneous document modifications whereas the conflict resolution guarantees lazy document consistency.*

That means that the conflict resolution is carried out without requiring user interaction and that all document instances are synchronized while maintaining consistency in a lazy fashion.

Concurrency Control Strategies

Numerous strategies have been devised to realize automatic conflict resolution as well as lazy document consistency. These strategies are divided into *pessimistic* and *optimistic* approaches [54]. Pessimistic approaches (e.g. locking mechanisms [55], turn-taking protocols [56], etc.), on the one hand, maintain consistency by establishing locks on documents that are being manipulated. On the other hand, optimistic approaches (e.g. operational transformation [3], differential synchronization [6], etc.) dismiss document locks and enable direct document changes. Even though document copies temporarily diverge and editing conflicts occur, optimistic approaches eventually preserve document consistency in a lazy manner and allow for automatic conflict resolution.

For real-time groupware applications, optimistic techniques are predominantly adopted due to their responsive nature. Optimistic concurrency control techniques allow end-users to instantly change a document accompanied by immediate system feedback. According to Nielsen [57], a response time of up to 0.1 second is the limit to perceive system reactions as *instantaneous*. This response time limit is satisfied by optimistic approaches and consequently, the end-user's flow of thought is not interrupted. In contrast, pessimistic approaches might lock documents for minutes or even hours and thus, fall short of providing a viable concurrency control approach for real-time groupware.

Operational Transformation

The predominant optimistic concurrency control algorithm in terms of industry adoption and research exploration is the Operational Transformation (OT) algorithm that was introduced by Ellis and Gibbs in 1989 [3]. Various modern groupware applications such as Google Docs [2], Etherpad [28] or SAP Gravity [29] incorporate some variant of the OT algorithm. Moreover, numerous research groups have advanced OT to tackle consistency issues [58], to suit differing document structures (e.g. SGML [59]) or to support sophisticated concurrency control operations (e.g. undo [60], operation compression [61]).

In contrast to optimistic concurrency control approaches that are *state-based*, i.e. document changes are expressed via calculated document differences (e.g. differential synchronization [6]), OT is *operation-based*. That means OT relies on a set of well-defined document operations and thus, document changes are represented as a set of operations.

To explain the OT mechanics, we introduce a simple example. Let's assume a text editor provides only a minimal set of operations: insert character and delete character. The operation insert character is expressed as $ins(c, i)$ where c denotes the character to insert and i the insertion index starting with 1. Correspondingly, the operation delete character is expressed as $del(i)$ where i denotes the deletion index also starting with 1.

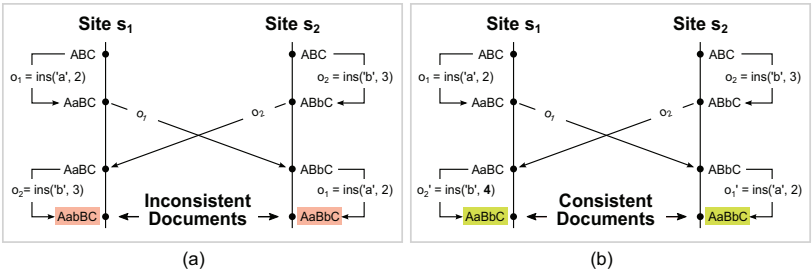


Figure 2.5: a) Document inconsistency emerging from a naive document merge b) Document consistency emerging from adopting the OT algorithm

In the scenario depicted in Figure 2.5 two users simultaneously insert a character in the consistent document ABC at different index positions. After the local operations $o_1 = ins('a', 2)$ and $o_2 = ins('b', 3)$ were applied to the local document copy, the operations o_1, o_2 are propagated to the other site in order to reconcile document copies. The naive replay of the local operations o_1, o_2 at the remote sites would result in two diverging documents containing $AabBC$ and $AaBbC$ (cf. Figure 2.5a). Adopting the OT conflict resolution scheme leads to a transformation of the operations o_1 and o_2 into o_1' and o_2' according to the transformation function f_1:

$$f_1(o_1, o_2) = \{o_1', o_2'\}$$

$$f_1(\text{ins}(c_1,\, i_1),\, \text{ins}(c_2,\, i_2)) = \begin{cases} i_1 > i_2 & \{\text{ins}(c_1,\, i_1 + 1),\, \text{ins}(c_2,\, i_2)\} \\ i_1 \leq i_2 & \{\text{ins}(c_1,\, i_1),\, \text{ins}(c_2,\, i_2 + 1)\} \end{cases}$$

Executing the remote operations o_1' and o_2' produces two documents, both containing the identical string $AaBbC$ (cf. Figure 2.5b). These two documents have reached a consistent state demonstrating that the editing conflict was successfully resolved.

2.2.2 Workspace Awareness

Besides concurrency control, workspace awareness represents the second important pillar for real-time groupware. The fundamentals of workspace awareness including definitions, critical elements supporting awareness as well as common workspace awareness widgets are presented in this section.

A key prerequisite for a successful collaboration in a shared environment is that people have access to information about the actions and intentions of their teammates. The term *awareness* captures the entirety of knowledge about other collaborators. The following awareness definition was given by Dourish and Bellotti.

Definition 2.2 – Awareness: *"Awareness is an understanding of the activities of others, which provides a context for your own activity" [62].*

Gutwin and Greenberg adapted this awareness definition for shared virtual workspaces which results in the following definition.

Definition 2.3 – Workspace Awareness: *"Workspace awareness is the up-to-the-moment understanding of another person's interaction with the shared workspace" [4].*

According to this definition, providing workspace awareness requires capturing interactions of other users with the virtual space. These interactions are multifaceted and include, for example, the re-positioning of mouse pointers or text cursors, gaze changes, viewport modifications, etc.

Workspace Awareness Elements

The fundamental interactions that contribute to workspace awareness were formalized by Carl Gutwin and Saul Greenberg in a dedicated framework [63]. In essence, this framework captures ten critical elements that are key to maintaining workspace awareness. These elements are compiled in Table 2.1 and can be divided into the *who*, *what* and *where* category. To make workspace awareness elements tangible, the associated question allowing to retrieve knowledge about the specific element is also listed in Table 2.1.

Category	Element	Question
Who	Presence	Is anyone in the workspace?
	Identity	Who is participating? Who is that?
	Authorship	Who is doing that?
What	Action	What are they doing?
	Intention	What goal is that action part of?
	Artifact	What object are they working on?
Where	Location	Where are they working?
	Gaze	Where are they looking?
	View	Where can they see?
	Reach	Where can they reach?

Table 2.1: Elements of workspace awareness [63]

Workspace Awareness Widgets

To materialize workspace awareness in real-time groupware, numerous workspace awareness widgets have been devised and described in the literature [4, 7, 8]. This set ranges from established workspace awareness widgets like participant lists, telepointers, telecarets or radar views to peculiar widgets like multi-user scrollbars or heat maps.

A collection of common awareness widgets is shown in Figure 2.6. The screenshots were taken from popular web-based groupware applications such as Google Docs [2] and Etherpad [28] as well as from the desktop groupware tool Codoxware [64] that enhances Microsoft Word with multi-user capabilities.

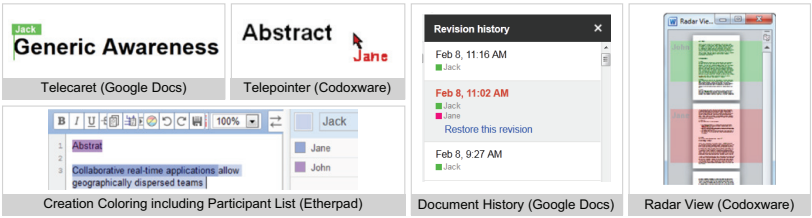


Figure 2.6: Common workspace awareness widgets

2.2.3 Web Technology

The third and last technological pillar of web-based real-time groupware is the web technology pillar. To understand the fundamentals of the web application platform, we introduce major buildings blocks that are of importance for real-time groupware including principles for communication (HTTP, WebSocket protocol), document naming (URI), document description (HTML) and document access (DOM). These web building blocks are an essential asset for developing real-time groupware since they are standardized and thus, allow the creation of application-agnostic and vendor-independent collaboration systems.

Building Blocks of the Web Application Platform

The web relies on major architectural building blocks encompassing a network protocol (HTTP), a markup language (HTML), an address system (URI) and a client-server model. All these essential blocks were already established by Tim Berners-Lee at the beginning of the web era and discussed in the seminal article “The World-Wide Web” [65].

The interplay and the relations of the network protocol, address system, markup language and the client-server architecture are summarized

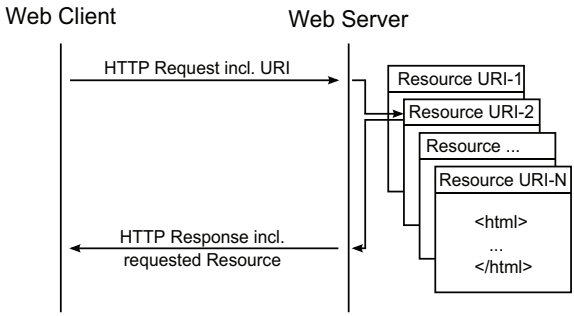


Figure 2.7: HTTP request/response cycle

in Figure 2.7. Interactions in the web are traditionally triggered by a web client requesting a web resource. Such a web resource is identified by a Uniform Resource Identifier (URI) [66] and resource requests are carried out via the Hypertext Transfer Protocol (HTTP) [67]. A web server is in charge of selecting the correct web resource from the set of published resources. Once selected, the server responds to the request by returning the desired resource that is commonly described with the Hypertext Markup Language (HTML) [49].

Besides providing standardized means for identifying, transmitting and describing documents, the document access is also covered by standards (i.e. the Document Object Model (DOM) standard [40] and the ECMAScript language specification [50]). While the DOM represents a language- and platform-independent document model, the ECMAScript language, which is commonly denoted as JavaScript, provides function libraries to alter DOM instances (e.g. to change the background color or to dynamically add artifacts). The original DOM is created from the HTML representation that was retrieved from the web server. Figure 2.8 shows a document represented as HTML, DOM and in form of the rendered web site.

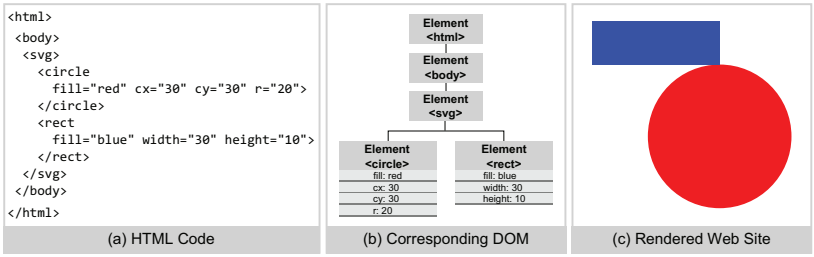


Figure 2.8: Document represented as HTML, DOM and rendered web site

Bidirectional Communication

In addition to the established web building blocks, bidirectional communication is essential for real-time groupware since it represents a prerequisite for document synchronization. Among all collaborating clients, shared documents have to be synchronized and the server, as the central communication hub, must also be able to push data to clients. Nevertheless, the traditional HTTP request/response cycle (cf. Figure 2.8) solely allows unidirectional communication triggered by the client. Therefore, various HTTP techniques using the XMLHttpRequest interface [68] have been devised to circumvent this restriction. These HTTP techniques depicted in Figure 2.9 include *polling*, *long polling* and *HTTP streaming*.

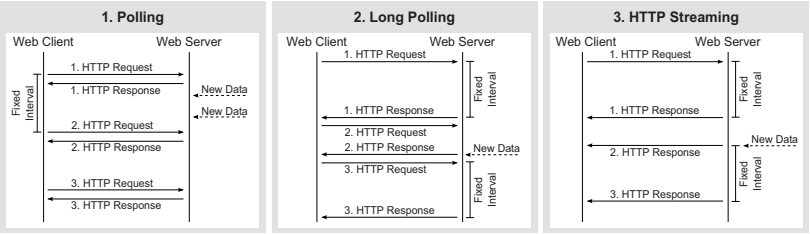


Figure 2.9: Bidirectional HTTP communication techniques [69, 70]

The polling technique is the naive approach whereas a client sends requests periodically to the server in order to check for server-side updates. However, the generated network load is high because the client always sends HTTP requests no matter if a server-side update is available. Long polling and HTTP streaming are more sophisticated techniques. While long polling instructs the server to delay responses until server-side changes are at hand or until a fixed interval elapsed, the HTTP streaming technique uses a persistent connection opened by the client that is not closed because the server keeps sending data packages to the client.

The presented HTTP techniques for two-way communication suffer from high network latency and from substantial network overhead induced by verbose HTTP header fields. As a consequence, the bidirectional *WebSocket protocol* was created and standardized in 2011 [71]. Since the WebSocket protocol is layered over TCP instead of HTTP, the generated network overhead as well as the network latency are drastically reduced. Moreover, a JavaScript language binding, providing event handlers such as `onopen`, `onclose` and `onmessage`, allows to natively leverage the WebSocket protocol in arbitrary web applications. The properties *minimal overhead*, *low latency* and *native language binding* qualify the WebSocket protocol as the primary communication means for real-time groupware.

2.3 Development Requirements

In this section, after having presented two concrete use cases for the development of real-time groupware, associated inefficiencies and technological foundations, we derive requirements for a development approach that addresses the identified inefficiency challenges and that promotes an efficient development of web-based real-time groupware. Therefore, we categorize the requirements into functional and efficiency requirements.

2.3.1 Functional Requirements

As exposed in the functional requirements discussion of the collaborative SVG-edit and the CoBAT web application, dedicated multi-user services providing concurrency control and workspace awareness features are indispensable. Consequently, a development approach for web-based real-time groupware has to facilitate concurrency control and workspace awareness.

Concurrency Control: Shared editing applications (e.g. Google Docs, Etherpad or the presented use case scenarios) have to ensure that simultaneous edits of geographically dispersed team members are synchronized in real-time, i.e. without notable delay, and that editing conflicts are resolved automatically, i.e. without requiring end-user interaction. Thus, a development approach for web-based real-time groupware has to properly support the integration of concurrency control features.

Workspace Awareness: The listed functional requirements of the converted multi-user SVG-edit and the CoBAT web application demonstrated the need for workspace awareness support allowing to understand the actions and intentions of others. To effectively promote workspace awareness, all defined elements of workspace awareness (cf. Section 2.2.2) have to be addressed. Hence, a tailored development methodology for collaborative web applications has to streamline the incorporation of workspace awareness capabilities.

2.3.2 Efficiency Requirements

Besides the straightforward functional requirements, the focus of this dissertation is to design the development process of web-based real-time groupware in an efficient manner and therefore, we also derive efficiency requirements.

Minimal Invasiveness: Since an invasive approach requires developers to get familiar with the existing source code, which is especially costly for single-user to multi-user transformation projects, an efficient approach should be non-invasive or at least reduce source code changes to the required minimum.

Encapsulation: To support efficiency particularly in the evolution of collaborative web applications, concurrency control and workspace awareness features should be encapsulated in dedicated components and obey the separation of concerns principle. Consequently, enhancing or replacing multi-user features exclusively affects specialized components and can be carried out in a streamlined fashion.

Learnability: The mechanics to introduce multi-user capabilities in web applications should be easy to learn, so developers can rapidly adopt the tailored approach. A low entry barrier does not only reduce the overall effort to integrate multi-user capabilities but it also increases the acceptance and adoption.

Reuse: Building upon reusable workspace awareness and concurrency control components instead of re-implementing multi-user modules for each and every collaborative application is a major efficiency driver. It does not only promote cost reductions for the initial development but also for the evolution since it is more efficient to maintain one dedicated multi-user component that is incorporated in various collaborative applications than to maintain one extra component for each groupware application.

Universality: The dedicated development approach for multi-user web applications shall support from-scratch as well as single-user to multi-user scenarios. Hence, the collaboration system accompanying the development approach has to be devised in a generic, application-agnostic fashion.

2.4 Summary

In this chapter, we established a common understanding for web-based real-time groupware and its development processes. First, we presented the two prevalent development approaches introducing the transformation use case with SVG-edit and the from-scratch use case with the CoBAT development scenario. These use cases were critical to derive common functional requirements of multi-user applications and to expose inefficiencies in the development process. To explain the foundations of web-based real-time groupware, we furthermore introduced the three technological pillars comprising concurrency control, workspace awareness and web technology. We concluded the chapter by discussing development process requirements that allow addressing the identified inefficiencies. These requirements – including functional aspects as well as efficiency factors – are used in the state of the art discussion in the next chapter where we analyze existing approaches to create real-time groupware for the web.

Chapter 3

State of the Art

After identifying the set of requirements for the efficient development of web-based real-time groupware, this chapter discusses state of the art development approaches grouped into programming libraries and frameworks, transformation techniques as well as web engineering approaches. The development approaches analysis reveals the basic mechanics of each and every development technique and also assesses their groupware development qualities. We conclude the chapter by summarizing to what extent the listed approaches fulfill the specified development requirements.

3.1 Groupware Development Libraries

A traditional way to implement shared editing applications for the web is facilitated via development libraries like SAP Gravity or ShareJS. These libraries support programmers by providing a set of functions easing the implementation of concurrency control and workspace awareness features.

3.1.1 SAP Gravity

SAP Gravity is a JavaScript library for creating shared editing applications for the web. Originally, SAP Gravity served as the concurrency control component for the collaborative business process modeler SAP Process Flow [29]. Due to the clear separation of the concurrency control layer (SAP Gravity) and the application layer (SAP Process Flow), SAP opened up the Gravity API to support the development of arbitrary collaborative web applications.

The SAP Gravity API consists of a set of JavaScript functions that allow manipulating a shared data model. Changing the shared data model automatically triggers model synchronization and conflict resolution. The integrated concurrency control uses a variant of the operational transformation algorithm. Developers are shielded from the OT complexity since

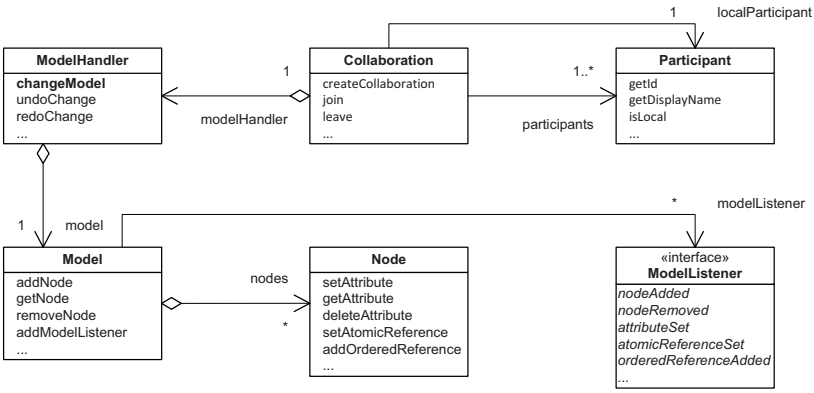


Figure 3.1: UML class diagram of the SAP Gravity JavaScript API

they only leverage the JavaScript API shown in Figure 3.1. Once a model change triggered by an API call occurred, the required OT sync and conflict resolution are autonomously processed by the SAP Gravity engine.

To actually manipulate the shared data model, developers leverage the `changeModel(command)` function belonging to the `ModelHandler` class. The `changeModel` function allows passing in a `command` function. Model changes are defined in the body of this `command` function. The Gravity implementation ensures that all model changes defined in the `command` function are executed transactionally, i.e. either all changes are applied to the shared model or, in case of a conflicting modification, changes are not applied at all. For example, if a user moves a rectangle in a graphics editor, the x and y attributes change. Because the move-rectangle change is one logical operation, the two changes would be handled in a transactional manner by the SAP Gravity engine. To initiate model changes via the `command` function, the `Model` and `Node` classes offer functions like `addNode`, `removeNode`, `setAttribute`, `setAtomicReference`, etc.

The Gravity data model is represented by a graph that comprises nodes with attributes that are interconnected with atomic or ordered references. While an atomic reference is a unary reference to exactly one node, ordered references maintain an ordered set of nodes. SAP Gravity also offers a notification mechanism that allows installing `ModelListeners` on model nodes.

In order to leverage the SAP Gravity API, web developers only have to embed the `gravityClient.js` file and the associated dependency file `com.sap.gravity.deps.js`. At runtime, the Gravity client connects to a dedicated Gravity server using either HTTP or the WebSocket protocol (cf. Section 2.2.3). The Gravity architecture adheres to the web communication principles and thus, clients can only communicate by means of a server hub.

In essence, the SAP Gravity API is suitable for from-scratch implementations requiring comprehensive concurrency control support. Since SAP Gravity relies on open standards (e.g. JavaScript, HTML, HTTP, etc.) and because the Gravity API is clear and concise, developers can easily learn and adopt the collaboration library. However, the support for workspace awareness is minimal (e.g. remote and local participants can be differentiated). Also reuse and minimal invasiveness, which is essential for transformation approaches, are poorly supported.

3.1.2 ShareJS

ShareJS [31] is an open-source concurrency control library that is implemented in CoffeeScript. Since CoffeeScript applications are compiled to JavaScript code, all modern browser engines can natively run ShareJS applications.

In contrast to SAP Gravity supporting a comprehensive graph model, ShareJS is currently limited to simple string objects and thus, not suited for sophisticated data structures like business process models or rich text documents. An excerpt of the ShareJS API is depicted in Figure 3.2. A collaboration session starts by establishing a `Connection`. The functions `open` or `openExisting` return an editable `Document`. Moreover, the `Connection` class allows disconnecting and setting up model listeners. Registering a listener is accomplished with the `on(eventname, callback)` function whereas valid event names are listed in the `Events` enumeration. The argument `callback` expects a function encapsulating the listener logic. All change operations are defined by the `Document` class. In addition to high-level functions like `insert(position, text)` or `del(position, length)`, ShareJS also offers low-level operations like `submitOp(operation)` suggesting that ShareJS also relies on the OT algorithm. Besides the ShareJS API for simple string objects, there is also an API for JSON objects. However, since the JSON API is still experimental, we omit a detailed discussion.

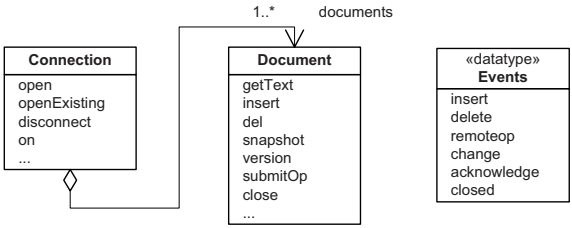


Figure 3.2: UML class diagram of the ShareJS API

In addition to the ShareJS client and the associated API, there is the ShareJS server component that is also implemented in CoffeeScript and

compiled to JavaScript. To execute the server-side JavaScript code, the Node.js platform [72] represents one viable option.

The lightweight ShareJS library encompassing only 4 kB in its minified version offers a compelling API for simple web applications that, for instance, require synching text input fields. Moreover, ShareJS is beneficial in terms of learnability and universality since it is concise and based on web standards. For advanced multi-user applications like word processors, graphics editors or BPMN modelers the limitations (e.g. no tree or graph model support, missing undo functionality, etc.) disqualify the ShareJS adoption. Again, just like SAP Gravity, the workspace awareness support is insufficient and the transformation of single-user web applications entails substantial effort due to the multitude of required source code changes.

3.2 Groupware Development Frameworks

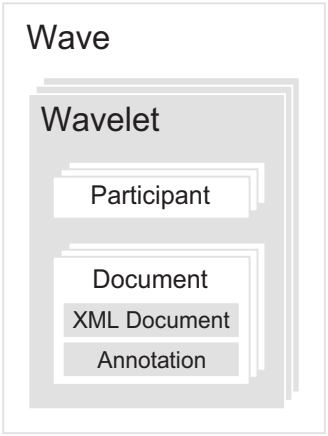
After discussing programming libraries, we analyze various common groupware frameworks such as Apache Wave [32], the MAUI Toolkit [73] and GroupKit [74]. While libraries offer a set of functions that return control after execution, frameworks embody an abstract design and provide a higher degree of built-in behavior. To leverage framework capabilities, developers have to complete a fixed skeleton, i.e. inherit from certain classes, implement specific interfaces, etc.

3.2.1 Apache Wave

Apache Wave [32], formerly known as Google Wave, is an open-source framework for collaborative web applications. The project was started as a Google product that was handed over to the Apache Software Foundation in 2010.

The Apache Wave architecture adopts the well-established client-server model, i.e. Wave clients communicate with other Wave clients using a server hub. Moreover, Apache Wave supports federation by means of the dedicated Wave Federation Protocol [75] representing an extension to XMPP [76]. Thus, the open Apache Wave architecture allows incorporating Wave servers from different ISVs.

The open-source project is unique in terms of communication capabilities supporting e-mail, instant messaging, shared editing, etc. One universal container called *Wave*, which is depicted in Figure 3.3a, is able to encapsulate all supported interaction formats. A Wave consists of a set of *Wavelets* that accommodate authorized participants and documents. Documents are divided into XML artifacts capturing document structure and content as well as annotation artifacts representing document formatting. In contrast to traditional communication protocols (e.g. POP3 [77]), Apache Wave persists all Wave documents on the server. The synchronization and conflict




```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
<ModulePrefs title="Increment Counter" >
  <Require feature="wave" />
</ModulePrefs>
<Content type="html">
<![CDATA[

    <script type="text/javascript">
    function incrementIt() {
      var value =
          parseInt(wave.getState().get('count', '0'));
      wave.getState().
          submitDelta({'count': value + 1});
    }
    </script>
    <input type=button value="Increment!"
       onClick="incrementIt()">

  ]]>
  </Content>
</Module>
```

(a) (b)

Figure 3.3: a) Wave container structure and b) Minimal Wave Gadget

resolution of Wave containers is guaranteed using a tailored implementation of the operational transformation algorithm (cf. Section 2.2.1).

To extend the existing Apache Wave platform with individual groupware applications, programmers have to either adopt the *Wave Gadget* approach [78] or directly leverage the Java API. While Wave Gadgets are suitable for small add-ons, the comprehensive Java API represents the means of choice for sophisticated applications. Wave Gadgets are derived from Open Social Gadgets [79] and are also composed of XML, HTML and JavaScript building blocks. The Wave structure can only be accessed using a shared state object which represents a key-value string map. The JavaScript API for Wave Gadgets contains numerous functions (e.g. `getState`, `submitDelta` or `setStateCallback`) to manipulate this shared state object. For example, Figure 3.3b depicts a minimal Wave Gadget where the shared state object stores a synched counter variable.

To directly manipulate Wave containers, developers have to leverage the Java API providing methods like `updateAttributes`, `deleteCharacters`, etc. Leveraging the comprehensive Java API requires using the Google Web Toolkit (GWT) [33] that compiles the Java application to JavaScript and HTML code.

Due to the Wave container abstraction that natively supports simultaneous document edits, developers may conveniently add concurrency control to web applications. In particular, the development of lightweight applications is well supported by the Wave Gadget approach. However, advanced groupware applications require the use of the Java API that suffers from poor

documentation since solely the actual source code is provided. The sparse documentation and the sheer size of the Wave project surpassing 200 000 lines of code substantially impair learnability. Additionally, the reuse is also limited, specifically because the framework lacks support for out-of-the-box workspace awareness widgets. Moreover, transforming single-user web applications into their collaborative counterparts is only feasible for GWT applications and requires invasive source code changes.

3.2.2 MAUI Toolkit

In contrast to the presented libraries and frameworks, the Multi-User Awareness UI (MAUI) toolkit [73] emphasizes workspace awareness support. On the one hand, workspace awareness is achieved by traditional single-user UI components (e.g. buttons, combo boxes or menus) that offer built-in awareness. On the other hand, dedicated multi-user widgets (e.g. telepointers) are provided. Built-in awareness means remote participants can retrace local user interactions involving conventional single-user UI components since the visual feedback is also delivered to remote participants. For instance, if a local user clicked a button or expanded a drop-down menu, the button-clicked state or the menu expansion is also visualized remotely. The remote visualization is adapted in color and style to be able to distinguish local and remote interactions.

The MAUI architecture is based on the Java Runtime Environment (JRE) [80] which ensures that MAUI applications can run on multiple platforms since the JRE is available for a variety of operating systems (e.g. Linux, Solaris or Windows). MAUI applications furthermore rely on specific Java and MAUI components that are depicted in Figure 3.4. Instead

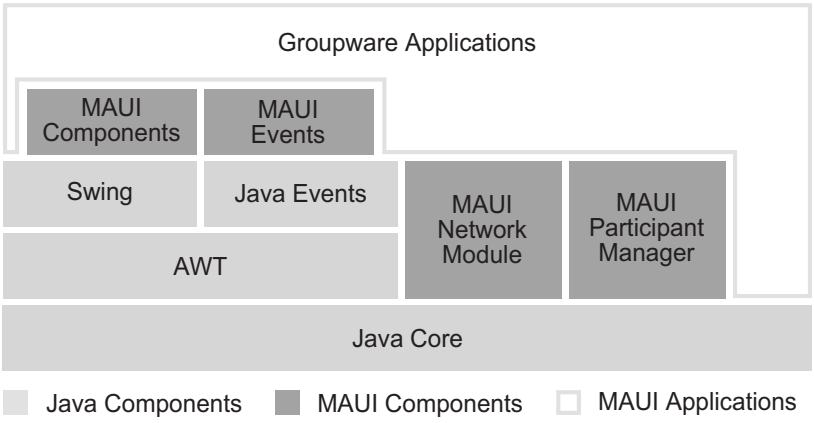


Figure 3.4: MAUI development stack [73]

of building UI components from scratch, MAUI components (e.g. `GButton`, `GMenu` or `GTextField`) enhance Swing components inheriting from the corresponding Swing classes (e.g. `JButton`, `JMenu` or `JTextField`). Besides enriching Swing widgets with awareness features, MAUI components also encompass specific multi-user widgets such as telepointers, participant lists and chat tools. To ease the implementation of workspace awareness support, MAUI provides an event component offering publish-subscribe for collaboration-specific events (e.g. `GUserArrivedEvent`, `GUserLeftEvent`, etc.). Moreover, user management and communication are also facilitated by MAUI via special-purpose modules like the participant manager and the network module.

All UI widgets contained in the MAUI toolkit can be used to compose collaborative applications leveraging a tailored GUI builder. The MAUI GUI builder is an extension of the JBuilder IDE [81] that adopts the established What You See Is What You Get (WYSIWYG) metaphor and allows to drag MAUI components from the toolbar to the actual UI. Hence, developers can easily get acquainted with the MAUI toolkit. In addition to the convenient development support, MAUI-based applications are packaged as standardized JavaBean components [82] promoting software reuse.

The strengths of the MAUI toolkit are the workspace awareness support and the development efficiency that is promoted by the IDE, the UI widgets and the adoption of standards (e.g. the JavaBean standard). However, the set of MAUI widgets suited for form-based applications does not cover advanced widgets such as a rich text editor component or a collaborative graphics editing pane that are required for advanced collaborative applications. MAUI is also limited in terms of concurrency control because even though UI widgets are synchronized, there is no built-in support for conflict resolution. A major drawback of MAUI is the Java runtime prerequisite. Solely browsers with a specific Java plugin can execute Java applications. Thus, there is no support for major mobile platforms (e.g. Android, iOS, etc.). Transformation scenarios considering pure web applications (e.g. the single-user to multi-user conversion of the SVG-edit graphics editor) are also not supported due to the incompatibility of the browser and the Java runtime engine.

3.2.3 GroupKit

After having implemented numerous groupware applications like GroupSketch, GroupDraw or Share, groupware pioneer Saul Greenberg and his research team designed the GroupKit framework [74] for synchronous groupware applications. GroupKit is intended to ease intricate and reoccurring groupware implementation tasks. The GroupKit framework consists of four major building blocks: (1) the runtime infrastructure, (2) a programmers API, (3) a set of groupware widgets and (4) a session management module.

Runtime Infrastructure: GroupKit's runtime infrastructure provides a multitude of process- and communication-related services for distributed systems. For example, the runtime manages the process lifecycle (creation, teardown, etc.), message exchange and inter-process communication. In each GroupKit application, the runtime coordinates three types of processes: the registrar, the session manager and the conference application. Figure 3.5 depicts an exemplary GroupKit scenario with two users and two conferences. The central registrar process is the first process that the GroupKit runtime creates. Afterwards, the session manager is constructed and immediately connects to the registrar to retrieve information about existing users and conferences. Besides requesting session information, the session manager can also direct the registrar to add/delete conferences and users. In contrast to the automatically created registrar and session manager processes, conference processes representing the actual groupware applications are controlled by the developer using the programmer API.

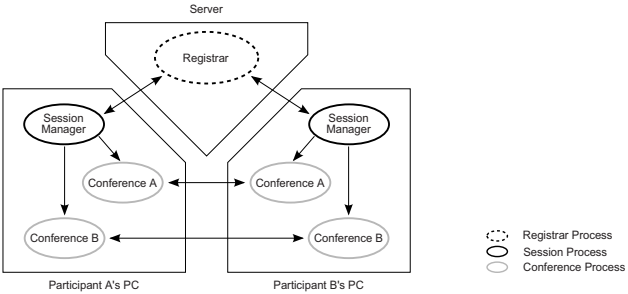


Figure 3.5: Exemplary GroupKit session [74]

Programmer API: While some services are automatically managed by the runtime infrastructure, a dedicated API allows programmers to build custom groupware applications. The GroupKit API therefore offers Remote Procedure Calls (RPCs), events and environments. GroupKit RPCs provide functions like `gk_toAll` or `gk_toOthers` to transport messages to other processes. The specific RPC library is convenient since developers do not have to deal with addresses and the process lifecycle. Moreover, programmers can leverage built-in or custom events to attach callbacks. For instance, events like `newUserArrived` or `userDeleted` are automatically fired enabling developers to adapt the application accordingly. A third API artifact is the environment object representing the application's data model. An environment data structure can be changed using key-value pairs and developers can control whether environments are automatically synchronized.

Groupware Widgets: In addition to the low-level programmer API, GroupKit provides predefined widgets including participant lists, telepointers, multi-user scrollbars and radar views. All widgets are based on the

Tcl/Tk library [83] and GroupKit also offers support for custom widgets providing a rudimentary class builder. In contrast to the MAUI toolkit, GroupKit does not contain common single-user UI widgets (e.g. button, combo box, etc.) that are collaboration-aware.

Session Management: Since Greenberg et al. "strongly believe that one of the obstacles to groupware use is the difficulty of starting up a groupware session" [74], GroupKit provides sophisticated support for session management. Hence, GroupKit offers functionality for creating, naming, deleting and locating conferences as well as adding and deleting participants. For example, the following built-in events are available: `foundNewConf`, `foundNewUser`, `newUserApproved`, etc.

GroupKit is an early groupware framework representative that was implemented in 1989. In general, GroupKit paved the way for efficient groupware development due to its support for workspace awareness, shared data structures and advanced session management. Moreover, GroupKit promotes reuse offering groupware widgets and a broad set of groupware-specific functions. However, in terms of concurrency control support, GroupKit is limited to the synchronization of shared data models and lacks conflict resolution. Also regarding platform support, GroupKit applications targeting common Unix environments (e.g. Solaris, AIX, HP/UX) neglect support for web applications which narrows the potential for end-user adoption. While GroupKit is suited for the from-scratch groupware development, the transformation of existing single-user editors was not a main objective of GroupKit. Thus, converting applications is cumbersome.

3.3 Transformation Approaches

Analyzing libraries and frameworks for groupware development showed that from-scratch implementations are conveniently supported but also exposed the insufficient assistance for single-user to multi-user application transformations. In this section, we identify, describe and assess transformation approaches.

3.3.1 Transparent Adaptation

Chengzheng Sun has been advocating transformation approaches since 2004 and introduced *transparent adaptation* [39] which is the prominent technique to transform "existing single-user applications into collaborative ones, without changing the source code of the original application" [39]. The transparent adaptation approach was adopted to convert various single-user applications (e.g. Microsoft Word, PowerPoint and Visio as well as Autodesk Maya) into their collaborative counterparts (e.g. CoWord [10], CoPowerPoint [39], CoVisio [84] and CoMaya [85]). For end-users, transformed applications are appealing since they are already familiar with the

application's look-and-feel, the interaction patterns, etc. Thus, the adoption barrier for the converted multi-user applications is minimal. From the developer's point of view, transparent adaptation is also convenient since the source code of the original application does not require any changes which limits the conversion effort.

The core idea of transparent adaptation is to monitor, propagate and replay each document change operation that emanated from the original Single-user Application (SA). Document synchronization and conflict resolution are facilitated by a reusable Generic Collaboration Engine (GCE). As depicted in Figure 3.6a, the Collaboration Adapter (CA) bridges the gap between the SA and the GCE, i.e. the CA maps the SA data model to the GCE data model and translates SA operations to GCE operations. The operation mapping is a two-step adaptation. First, the SA operations are translated to Adapted Operations (AOs) and second, AOs are mapped to Primitive Operations (POs) that can be processed by the GCE.

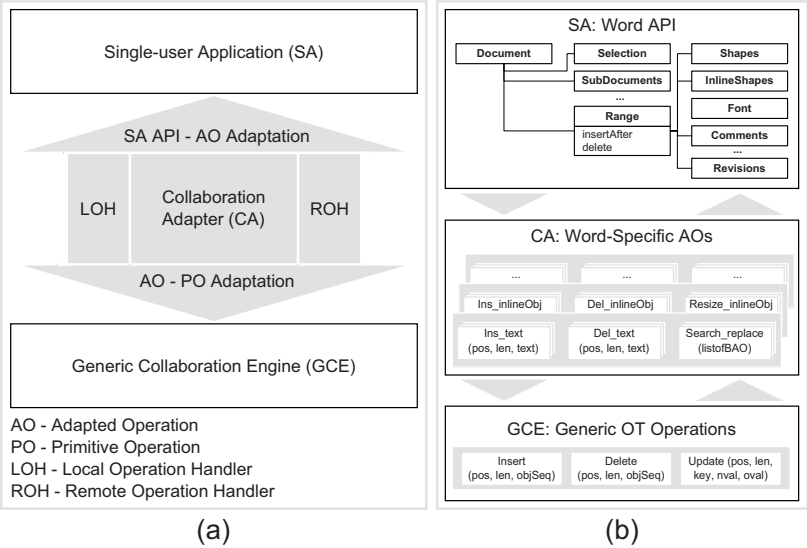


Figure 3.6: a) Transparent adaption architecture [39] and b) CoWord-specific implementation [10]

Figure 3.6b shows the collaboration adapter for Microsoft Word including original Word operations, tailored adapted operations as well as generic primitive operations. Note that the AO-PO mapping is a 1-N relationship, i.e. an AO can accommodate numerous POs. The translation of SA-specific operations into AOs inherently accomplishes the data model mapping where the application-specific data model is mapped to a linearly addressable data

model. For instance, Microsoft Word exhibits a hierarchical data model that is translated into a linear sequence of objects which allows adopting a universal GCE. In addition to the two adaptation modules, the CA also encapsulates the Local Operation Handler (LOH) and the Remote Operation Handler (ROH). While the LOH is responsible for intercepting local events, retrieving context information and creating a corresponding AO, the ROH receives and replays remote operations. Moreover, the CA also provides a set of workspace awareness widgets that for the Microsoft Word implementation includes a radar view, a telepointer and a creation coloring widget.

Transparent adaptation provides solid concurrency control support because GCE operations are realized by OT operations. In terms of workspace awareness, transparent adaptation is suited for devising synchronous groupware applications since it offers common awareness widgets. Reuse is provided by the GCE because the GCE supports multiple applications. However, the collaboration adapter is application-specific which impairs reuse. The application-specific CA implementation also lowers development efficiency. Although developers do not have to change the application's source code, which fulfills the minimal invasiveness requirement, the implementation of the CA is demanding. Evidently, transparent adaptation is suited for transformation approaches. The from-scratch development is also possible but the development effort increases significantly in comparison to groupware libraries or frameworks.

3.3.2 Flexible JAMM

The transformation approach Flexible Java Applets Made Multiuser (Flexible JAMM) [86] was introduced by Begole et al. Like transparent adaptation, Flexible JAMM represents another collaboration transparency approach. Collaboration transparency means the single-user application is turned into a collaborative application through mechanisms that are unknown or *transparent* to the original single-user application and its developers [87]. To adopt the Flexible JAMM approach, target platforms have to offer capabilities for (1) process migration, (2) runtime component replacement, (3) dynamic binding as well as (4) event interception and event introduction. Since the Java platform provides these required features, Begole et al. implemented the Flexible JAMM prototype using the Java platform. This allows converting existing single-user Java Swing applications into collaborative ones.

Process Migration: Flexible JAMM is a replicated system, i.e. the single-user application is executed on every client machine. This replicated system supports *late-joiners* which are clients that were not part of the initial collaboration session. Updating late-joiner processes is accomplished via process migration. In order to successfully migrate a process, information about the address space, used system resources and the execution state have

to be gathered and propagated to the joining client. Flexible JAMM carries out process migration by means of Java Object Serialization (JOS). The JOS feature allows converting an in-memory object into a serialized form (i.e. a text representation) that can be transferred over the network. Hence, Flexible JAMM is able to for example transfer a `JTextField` component and its associated input data to joining clients.

Runtime Component Replacement: An essential Flexible JAMM aspect is the runtime replacement that allows exchanging a class definition with a varying class definition at runtime[1]. This mechanism is leveraged to replace single-user UI widgets with multi-user versions that are enhanced with concurrency control and workspace awareness features. Figure 3.7 shows an example where the Swing class `JScrollPane` is replaced by the `FJammScrollPane` that is enriched with a radar view widget. Again, the Java platform provides an API to carry out runtime replacements. However, a replacement is only possible if the original class and the replacement class implement the same interface or inherit from the same class.

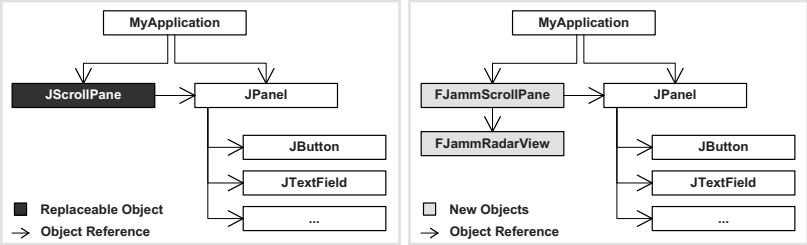


Figure 3.7: Runtime objects before and after the component replacement

Dynamic Binding: The runtime replacement feature depends on Flexible JAMM's dynamic binding capability. Instead of resolving references at compile time, dynamic binding delays resolving references to the actual runtime. Assuming the application depicted in Figure 3.7 is in the state of creating a new pane object to accommodate various UI components, dynamic binding for example allows that the constructor method of the novel component (`FJammScrollPane`) is called instead of calling the original constructor (`JScrollPane`). While private, static and final methods in Java are statically bound, overridden methods are bound dynamically and are thus adopted by Flexible JAMM.

Event Interception and Event Introduction: The last crucial characteristic of Flexible JAMM is the ability to intercept local events and to introduce remotely generated events into local event queues. In order to maintain the interaction of all participants consistent, Flexible JAMM in-

[1]Note that changing class definitions at design time is not an option since the Flexible JAMM approach bans direct source code changes.

tercepts and holds back local events until a total order of all concurrent local and remote events is calculated. Once the event order is computed, the events are applied accordingly. For example, if one user clicks the `Open File` button and another user selects the `Print` button, both interactions would result in showing a modal dialog and the interaction would diverge. Holding back the events and computing a total order ensures that one of the concurrent events is applied first (e.g. show the open file dialog) and then the interaction is in sync. To implement event interception and introduction, Flexible JAMM adopts the Java Swing UI component library since it offers means to intercept events and to manipulate event queues.

In essence, Flexible JAMM is a sophisticated approach to convert Java Swing applications transparently into their collaborative counterparts. Flexible JAMM incorporates an OT-based consistency maintenance engine delivering sound concurrency control. Workspace awareness support is fair since Flexible JAMM offers a dedicated telepointer and a radar view. However, replays evoked by remote events are not distinguishable from user interactions that were initiated locally (e.g. the animation of a pushed button) and thus, end-user awareness is reduced. Due to the requirement of non-invasive approaches to not to change the existing source code, Flexible JAMM allows for proper encapsulation of single-user and multi-user functionality. From a learnability point of view, the Flexible JAMM approach is demanding because in order to know what classes can be exchanged developers have to understand the inheritance and interface implementation hierarchies exposed by the single-user application. Moreover, developers have to get familiar with advanced Java concepts such as runtime component replacement or dynamic binding. These learnability challenges also represent a major impediment for adopting this approach for from-scratch development projects.

3.3.3 JEIS

Lowet and Goergen introduced the JavaScript Engine Input Synchronization (JEIS) approach [88] to primarily realize co-browsing scenarios for TV platforms. JEIS allows users of various networked television sets to, for example, co-browse pictures on Flickr, simultaneously watch videos on YouTube or to collaboratively shop on Amazon. Since televisions are usually shipped with a predefined set of software (e.g. a web browser) that cannot easily be updated or enriched with plugin technologies such as Adobe Flash or Microsoft Silverlight, JEIS exclusively relies on established web standards (e.g. HTTP, HTML and JavaScript). Moreover, to support co-browsing for a multitude of web sites and web applications, Lowet and Goergen selected a non-invasive approach that did not require changing the JavaScript code of the original web application.

In order to sync various web application instances that are executed in different browser engines, JEIS aims to maintain all instances of the document object model consistent. DOM consistency is achieved through input synchronization taking into account (1) UI events and (2) incoming server-side data. While UI events (e.g. `mousedown`, `mouseover`, `click`, `scroll`, etc.) are generated by end-users, server-side data results from synchronous HTTP requests (e.g. clicking hyperlinks) or from asynchronous JavaScript requests (XMLHttpRequest-based communication [68]).To handle both input types, JEIS adopts varying strategies to maintain DOM consistency. On the one hand, UI events are intercepted and resulting DOM changes are held back to sync concurrent UI events that originated from various browser engines. Figure 3.8 depicts an exemplary scenario where two browsers propagate incoming UI events to a UI event ordering service that returns an ordered set of UI events. Applying the ordered set to all DOM instances ensures DOM consistency. On the other hand, when data requested from a web server arrives at the client-side, the DOM commonly also changes since the results are visualized on the UI. In today's complex web applications, an HTTP request with the same URI commonly returns differing data if initiated from different browser engines because the request often includes browser-specific Cookie data. To still guarantee that the same data is returned for each browser and that the entailed DOM manipulation is equal at all sites, JEIS uses a proxy server as depicted in Figure 3.8. So instead of requesting the same URI from each browser, one browser triggers the request over the proxy and once the response arrives at the proxy server, the proxy distributes server-side data to all collaborating browsers.

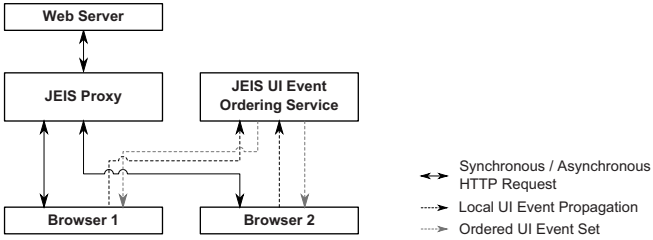


Figure 3.8: Exemplary JEIS scenario

The implementation of JEIS consists of three major components: the proxy, the UI event ordering service and a component to implement the UI event synchronization. While the implementation of the former two is straightforward, the latter is interesting since the client-side UI sync component has to be built in a non-invasive fashion. Therefore, in [88] the authors offer three options: (1) code insertion by means of a proxy, (2) a dedicated browser add-on or (3) code injection using an iframe. The first option utilizes the proxy to anchor the UI event sync code in the original web application

during the initial download. The second option equips the browser with an add-on such as Greasemonkey [89] that allows enhancing browser engines with JavaScript logic. Thus, the enhanced browser can adapt the behavior of web applications. The last option is to call all web applications from an enriched iframe. Such an iframe can inject JavaScript code when special privileges for cross-domain scripting are granted.

The strength of the JEIS approach is the universality and the non-invasiveness which allows adapting arbitrary web applications. Moreover, JEIS as illustrated in [88] works out-of-the box without requiring any application-specific adaptation and consequently, developers can rapidly adopt the approach. The weaknesses of JEIS are concurrency control and workspace awareness support. Even though JEIS offers concurrency control providing a central UI event ordering service, the adopted mechanism represents a pessimistic approach (cf. Section 2.2.1) impairing immediate end-user feedback. The feedback is delivered after the central ordering service was contacted and thus, the collaboration in real-time is not guaranteed. Furthermore, Lowet and Goergen do not discuss how workspace awareness capabilities could enhance a JEIS co-browsing session.

3.4 Web Engineering Approaches

After having discussed libraries, frameworks as well as transformation techniques for the efficient development of real-time groupware, we present established web engineering approaches. Since the discipline of web engineering is devoted to providing methodologies, frameworks as well as best practices for the development and maintenance of web applications, their suitability for building web applications is unquestionable. Nevertheless, in our discussion we analyze to what extent they can conveniently support the development of web-based synchronous groupware including concurrency control and workspace awareness capabilities.

3.4.1 WebML

The Web Modeling Language (WebML) [21] was introduced in 1998 and aimed to bridge the gap between the design and the implementation phase of data-intensive web applications. This gap between the paper-based design specification and the actual source code implementation entails a number of issues, e.g. the application's requirements and the corresponding implementation may diverge or reuse opportunities translating specification artifacts into a code representation may be neglected. Hence, Ceri et al. created WebML, a platform-independent language for specifying web applications that allows generating the application's implementation. Thus, WebML preserves the link between specification and implementation.

WebML accommodates multiple models to describe the various facets of a web application. The crucial models are (1) the structural, (2) the hypertext, (3) the presentation and (4) the personalization model. The structural model expresses the data dimension of a web application, i.e. data entities and their relations. To this end, WebML adopts existing languages such as ER models [91] or UML class diagrams [92]. The hypertext model specifies so called *site views* that capture information about the individual pages, associated content and interconnecting navigation links. Moreover, the content is subdivided into content units (e.g. data, index, filter, etc.) that reference entities from the structural model. The presentation model defines the layout of pages by means of a device-independent XML language. Designers can determine graphic properties of a page by attaching a page-specific presentation model or assigning a generic presentation definition. The last essential model is the personalization model grouping users in dedicated groups and associating content or presentation customizations to specific users or user groups.

To efficiently author WebML-based applications, the company WebRatio offers the WebRatio IDE [90]. The WebRatio development environment is depicted in Figure 3.9 and comprises a set of editors to conveniently author the various models, a repository to persist models, a code generator to translate models to source code and a runtime component to support the execution of generated WebML applications. To successfully run WebML applications, connectivity to the data sources is required.

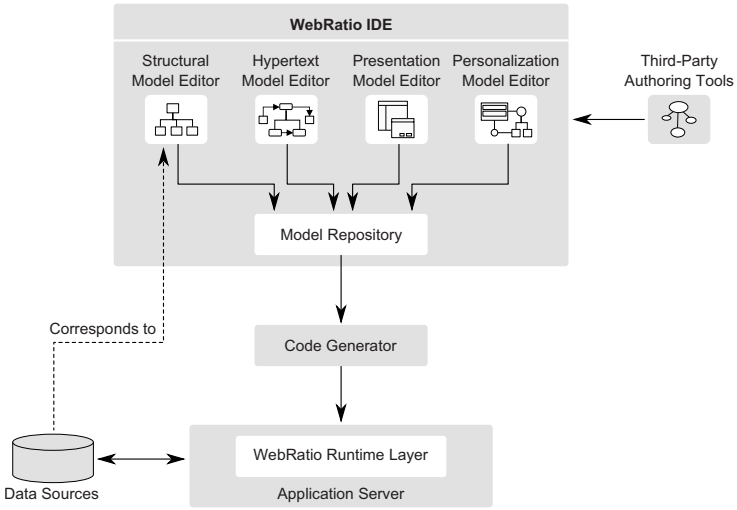


Figure 3.9: The WebRatio architecture [90]

The model-driven development furnished by WebML and the WebRatio IDE supports the development productivity, in particular for realizing data-intensive web applications that are created from-scratch. Productivity improvements primarily stem from the ability to reuse specification documents for the actual application implementation. The WebML approach can furthermore be adopted with moderate effort due to the advanced IDE support and the reuse of established modeling languages (e.g. UML or ER diagrams). However, developers also have to get familiar with a substantial set of novel model types and notations (e.g. the hypertext, presentation and personalization model). While WebML is suited for data-intensive web applications, the support for synchronous groupware applications requires adopting third-party authoring tools. Leveraging the provided interface for linking third-party authoring tools allows effectively anchoring concurrency control and workspace awareness capabilities. WebML is not designed to transform existing web applications into collaborative ones since the approach relies on numerous WebML-specific models that are not available for arbitrary web applications.

3.4.2 UWE

UML-based Web Engineering (UWE) [22] is another model-driven approach for web application development that is exclusively built upon open standards such as UML [93], XMI [94], MOF [95] and QVT [96]. While the Unified Modeling Language (UML) is used to describe content, navigation and presentation; the XML Metadata Interchange (XMI) format is leveraged for model persistence, the Meta-Object Facility (MOF) to define metamodels and the Query/View/Transformation (QVT) language to express model transformations.

UWE's support for web developers spans across various application development dimensions that are depicted in Figure 3.10. Besides supporting multiple development phases, UWE considers structural as well as behavioral aspects and also covers a number of views. At the core of UWE are (1) the requirements, (2) the content, (3) the navigation structure, (4) the process and (5) the presentation model. The requirements model captures application functionalities using UML use case and activity diagrams. UML class diagrams are used to specify content models representing application entities and their relationships (e.g. regarding content, context or user characteristics). The requirements and content model form the basis for the navigation structure that is defined as a stereotyped UML class diagram, i.e. the UML metamodel is extended to provide navigation and process classes. Each navigation structure is refined in a process model specified as a UML activity diagram. To finalize the application design, the presentation model in form of stereotyped UML class diagrams has to be created. The presentation model defines the basic UI structure including UI containers

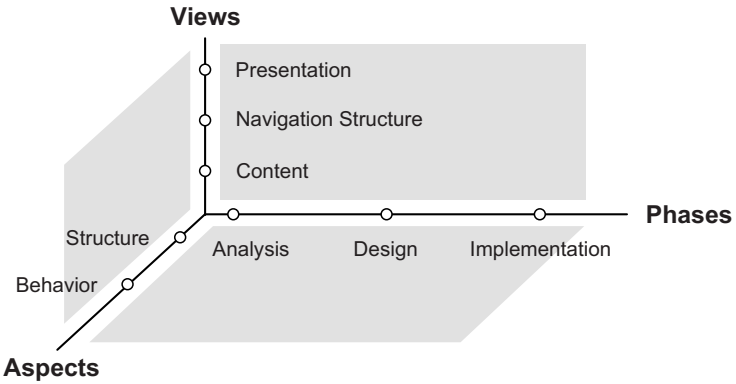


Figure 3.10: Application development dimensions covered by UWE [22]

and primitive elements (e.g. buttons, input fields, etc.) while abstracting from concrete properties (e.g. fonts, colors, etc.).

To devise web applications efficiently, UWE offers model transformation and tooling support. The approach for model transformations adheres to the Model-Driven Architecture (MDA) methodology [97] that allows generating and refining models on different abstraction levels. The actual implementation of the application represented as source code is also generated. Thereby, model-to-model and model-to-code transformations are defined using QVT. To provide proper tooling, ArgoUWE [98], an extension of the open-source ArgoUML tool [99], supports web application developers. ArgoUWE offers tailored model editors, integrates transformation facilities and provides means for model validation.

Like WebML, UWE also brings together the different software development phases (analysis, design and implementation) and thus, modeled artifacts from one phase can be reused in a following phase since transformations generate skeletons for the subsequent model. Another benefit of UWE is the clear separation of concerns which reduces software maintenance costs. Even though reuse and separation of concerns are well facilitated by UWE, the remaining specified requirements (cf. Section 2.3) lack adequate support regarding the development of collaborative web applications. Concurrency control and workspace awareness capabilities are not provided out-of-the-box. Transforming existing single-user web applications to collaborative ones creates a substantial overhead since the existing application has to be described in terms of UWE models. Finally, the effort to get familiar with the UWE approach is significant due to the vast set of models, technologies and tools.

3.4.3 OOHDM

The Object Oriented Hypermedia Design Method (OOHDM) [23] is one of the early model-based web development methodologies that was introduced by Rossi and Schwabe in 1995. The goal of OOHDM is to provide a suitable abstraction for defining hypermedia applications. Therefore, OOHDM supplies a convenient and expressive modeling language which includes (1) a conceptual, (2) a navigational and (3) an interface model. To map these platform-independent models to a concrete implementation, developers can leverage a tailored IDE called the OOHDM-Web environment [100].

Creating web applications using OOHDM starts with the design of the conceptual model. The conceptual model captures domain-specific entities and their relations using a graphical syntax that is similar to the UML class diagram notation. Afterwards, the navigational model is constructed as a *view* on top of the conceptual model. This includes defining navigation nodes, establishing links between navigation nodes and specifying access structures for data retrieval from concept classes. As depicted in Figure 3.11 navigation nodes may aggregate attributes from numerous concept classes. Moreover, views represent the central means to establish the application's role concept by designing specific views for the diverse user profiles. After defining the navigational model, the interface model has to be created. That includes establishing the mapping of navigation nodes to abstract interfaces, specifying actions in response to external or user-generated events and describing interface transformations. As a result, interface classes are created that in turn are constructed from primitive elements (e.g. input fields, button, etc.) and existing interface classes.

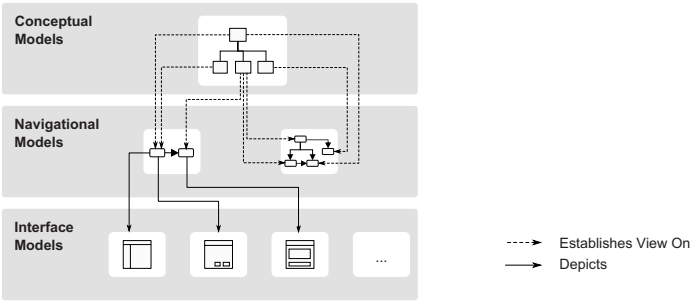


Figure 3.11: OOHDM models and their relations [23]

Besides creating various OOHDM models, web developers have to transform the high level application specification into executable code. Just like WebML and UWE, OOHDM also provides a dedicated IDE, the OOHDM-Web environment, that supports this transformation. OOHDM-Web allows mapping conceptual models to database tables and navigational as well as

interface models to HTML pages. To also support interactive web applications, OOHDM-Web translates navigation and interface specifications into HTML templates while dynamic content is expressed using the Lua script language [101].

In general, the strengths and weaknesses of OOHDM are similar to the introduced model-based approaches WebML and UWE. The specific models for data, navigation and interface aspects promote the separation of concerns paradigm which is beneficial in terms of software maintenance costs. Reuse is also emphasized by OOHDM since artifacts from the analysis and design phase can partially be used for the implementation through automatic transformations. Moreover, Rossi and Schwabe state that OOHDM is a viable and proven development method for arbitrary hypermedia applications (e.g. web-based as well as desktop-based hypermedia applications). However, the transformation of existing web applications suffers again from the non-existence of OOHDM models for arbitrary web applications which results in excessive modeling effort to reproduce the existing application. Learning OOHDM also entails substantial effort due to the variety of model types and notation languages. But the major OOHDM deficiency is the missing concurrency control and workspace awareness support that are indispensable for groupware applications.

3.4.4 WebComposition

In contrast to the presented model-driven web engineering approaches, Gaedke et al. introduced the WebComposition approach [24, 25, 26] adopting object-oriented design principles [102] such as abstraction, encapsulation and inheritance. The aim of WebComposition is to foster reuse and application maintainability by providing a fine-grained object-oriented web application model throughout the entire application lifecycle instead of adapting coarse-grained file-based resources (e.g. HTML files).

Conventional web development approaches do not differentiate between design time artifacts and runtime artifacts such as HTML, JavaScript or CSS files, which impairs adopting established object-oriented paradigms like abstraction, encapsulation or inheritance. For example, the lack of fine-grained encapsulation support for web resources leads to the copy-and-paste phenomena, e.g. copying styles to numerous CSS files or including HTML headers or footers in various HTML resources. Leveraging copy-and-paste breaks the application's consistency if one copied source code snippet changes while others are not modified correspondingly. Another example is the missing abstraction for hyperlinks that are directly inserted in HTML. This again breaks the application logic if only one hyperlink is modified in one HTML file while other HTML documents do not include required hyperlink changes. Therefore, the WebComposition system introduces fine-

grained components promoting the benefits of object-oriented systems such as reuse and maintainability.

WebComposition components vary in granularity and cover items ranging from coarse-grained artifacts like web sites or pages to fine-grained objects like tables, anchors or even table entries. To construct web applications, components are combined to composite components. For instance, a simple web site may be created aggregating a header, content and a footer component. All components are defined using the XML-based WebComposition Markup Language (WCML) [103]. To map the WCML design time objects to HTML, JavaScript and CSS documents, each WCML component has to call the `generateCode` method and the entirety of generated source code embodies the web application. To leverage the reuse paradigm to its full extent, the WebComposition system establishes the sharing and the prototyping concept. While sharing allows incorporating the very same component multiple times in one application (e.g. a hyperlink component in numerous pages), prototyping enables deriving components from other components.

The WebComposition architecture is depicted in Figure 3.12 consisting of WebComposition-specific artifacts such as the component store, component server or resource generator and general web infrastructure artifacts like the web server. Implementing web applications starts by specifying WCML components adopting a set of dedicated authoring tools. While the component server manages access, revision control, etc., the persistent storage is materialized by a RDBMS. Component developers also trigger the resource generation which maps WCML components to file-based resources that can be accessed from the Internet using a regular web server.

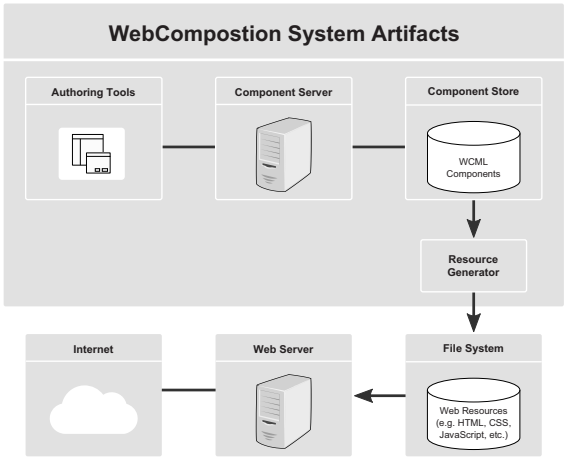


Figure 3.12: The WebComposition architecture [24]

Due to the extensive adoption of object-oriented principles, the WebComposition approach is a capable solution to promote component reuse and component encapsulation which both drive development productivity. From a learnability point of view, the WebComposition system requires only modest familiarization effort. It therefore outperforms other web engineering approaches that confront developers with a vast set of differing modeling languages. However, just like WebML, UWE or OOHDM, evolving an existing web application necessitates developers to adopt a specific model dialect (e.g. WCML) which essentially requires rebuilding the existing web application and thus, the universality of the approach is limited. Even though concurrency control and workspace awareness features are not natively anchored in the WebComposition system, the component-based architecture allows effectively adding special-purpose components for concurrency control and workspace awareness support. Thus, the implementation of synchronous web applications is feasible leveraging the WebComposition approach.

3.5 Assessment

After having introduced a number of libraries and frameworks as well as transformation and web engineering approaches for the efficient development of collaborative web applications, this section summarizes findings for each category with respect to the defined functional and efficiency requirements (cf. Section 2.3). Furthermore, a detailed results table aggregating assessments for all approaches is presented and discussed.

Groupware development libraries like SAP Gravity and ShareJS only partially address the functional requirements concurrency control and workspace awareness. On the one hand, concurrency control is properly supported by appropriate API functions. On the other hand, workspace awareness widgets such as telepointers or telecarets are not provided at all. From the efficiency point of view, low-level libraries are handy in terms of learnability since developers may easily adopt collaboration libraries. Also reuse and encapsulation are to some extent promoted because the library functions allow reusing and encapsulating logic. Nevertheless, substantial invasive changes disqualify groupware development libraries for single-user to multi-user transformations. In summary, modern groupware libraries are feature-wise incomplete and are not able to efficiently support transformation approaches. However, the low entry hurdle that developers face is unmatched by other approaches and key for new approaches to gain adoption.

Groupware development frameworks such as Apache Wave, MAUI and GroupKit demonstrated that their groupware capabilities surpass the other explored development approaches. While most approaches (e.g. collaboration libraries) recognize the need for concurrency control in collaborative applications, workspace awareness features are commonly neglected. In

contrast, Greenberg and Gutwin pioneered the introduction of workspace awareness widgets in frameworks like GroupKit or MAUI. However, the rich feature set offered by groupware frameworks comes at a cost, namely, learnability and universality. First, the high-level abstraction of frameworks leads to substantial effort in terms of familiarization with development tools and methodologies. Second, the framework abstraction with its rigid development process also entails inflexibility. This inflexibility prevents leveraging groupware frameworks for single-user to multi-user application conversions that have to consider the existing single-user code base. Hence, our objective of leveraging the variety of single-user web applications for shared editing cannot be reached adopting existing groupware frameworks.

Transformation approaches being tailored for single-user to multi-user application conversions obviously support transformation scenarios decently. All of the presented approaches (Transparent Adaptation, Flexible JAMM, JEIS) are non-invasive. Thus, getting acquainted and changing the original source code is not necessary. Moreover, transformation approaches are appropriate in terms of feature encapsulation and reuse. In contrast to specialized groupware development libraries or frameworks that are dedicated for from-scratch implementations, transformation approaches prove to be more flexible since they can be leveraged for transformation and from-scratch scenarios. For from-scratch scenarios, single-user features are first implemented and afterwards, the regular transformation capabilities are employed to introduce multi-user features. Even though the transformation class does not expose a general deficiency, the individual approaches exhibit shortcomings. While Transparent Adaptation incurs significant development effort requiring an extra collaboration adapter for each application, Flexible JAMM entails a complex programming model relying on runtime replacement mechanics and JEIS misses to offer sufficient concurrency control and workspace awareness support. Hence, the approaches cannot be used in their current form but they reveal essential principles for a hybrid design supporting from-scratch as well as transformation scenarios.

Web engineering approaches like WebML, UWE, OOHDM or WebComposition represent the last class of analyzed approaches. All approaches separate design time and runtime artifacts which, in particular, propels encapsulation and reuse. The specific design time representation is a major impediment for transforming web applications into collaborative ones since the original application has to be re-created in form of the dedicated design time model. In terms of learnability, the WebComposition approach accommodating solely WCML models is more lightweight than the model-driven approaches that confront developers with a variety of differing models. However, the deficiency that all web engineering approaches suffer from is the lack of synchronous collaboration support. While the WebML and the WebComposition approach provide extension mechanisms to incorporate concurrency control and workspace awareness capabilities, UWE and OOHDM

currently do not address synchronous collaboration support. Consequently, the existing web engineering approaches cannot be adopted directly to efficiently develop collaborative web applications. Nevertheless, their focus on reuse and encapsulation as well as WebComposition's component-based design are vital characteristics for development productivity.

Table 3.1 summarizes our review and assessment of the individual approaches in the light of the specified requirements. This summary shows again that all approach categories (libraries, frameworks, transformation as well as web engineering techniques) have their unique strengths. However, a balanced approach supporting from-scratch as well as transformation scenarios, providing capable collaboration facilities and promoting proven efficiency elements such as encapsulation or reuse cannot be identified.

	Concurrency Control	Workspace Awareness	Minimal Invasiveness	Encapsulation	Learnability	Reuse	Universality
SAP Gravity	++	○	-	○	+	○	+
ShareJS	+	-	-	○	++	○	○
Apache Wave	+	○	-	+	-	+	-
MAUI Toolkit	○	++	-	+	○	+	○
GroupKit	+	+	-	○	○	+	○
Transparent Adaptation	++	○	++	+	-	○	+
Flexible JAMM	+	+	++	+	-	+	○
JEIS	○	-	++	○	+	+	+
WebML	○	○	-	++	○	++	○
UWE	-	-	-	++	○	+	○
OOHDM	-	-	-	++	-	+	-
WebComposition	○	○	-	++	+	++	○

Support Levels: - Poor ○ Sufficient + Good ++ Excellent

Table 3.1: Overall assessment of the individual approaches with respect to the specified requirements (cf. Section 2.3)

3.6 Summary

In the state of the art chapter, we identified major approaches to efficiently develop web-based real-time groupware considering industry best practices (e.g. groupware libraries) as well as established scientific approaches (e.g. web engineering approaches). We divided existing techniques in the categories libraries, frameworks, transformation techniques as well as web engineering approaches and carved out individual as well as class-wide strengths and shortcomings. This thorough analysis revealed that a balanced approach delivering necessary collaboration features (concurrency control, workspace awareness) as well as supporting essential productivity drivers (e.g. reuse, encapsulation, etc.) is currently not at available. Nevertheless, the in-depth exploration of the state of the art also demonstrated how certain characteristics like learnability, minimal invasiveness, etc., can be designed in an effective manner. These findings form the foundation for the design of a groupware development approach for the web that addresses all of the defined functional and efficiency requirements.

Chapter 4

Generic Collaboration Infrastructure

After presenting use cases for web-based real-time groupware, corresponding development problems and state of the art solutions, we introduce the Generic Collaboration Infrastructure (GCI) [104, 105]. The GCI is a capable means to inject concurrency control and workspace awareness features into web applications while promoting development efficiency. Thereby, we revisit development issues and requirements for collaborative web applications, discuss emerging challenges devising the GCI, present the GCI architecture and illustrate the accompanying development methodology. While this chapter gives an overview of the GCI and its core components (the DOM adapter, the framework adapter and the workspace awareness adapter), Chapters 5, 6 and 7 present the GCI core components in detail.

4.1 Design Considerations

To thoroughly devise a generic collaboration infrastructure for web-based real-time groupware, we first revisit a number of decisive aspects. These aspects include (1) development requirements discussed in Section 2.3 that were derived from two concrete use cases, (2) general inefficiency problems exposed in Section 1.3 that arise when implementing collaborative web applications and (3) state of the art solutions analyzed in Chapter 3. To illustrate and justify GCI design decisions, we briefly elaborate on these aspects.

Development requirements (cf. Section 2.3) form the foundation for the GCI architecture. Section 2.3 classified concurrency control and workspace awareness as *functional* requirements and minimal invasiveness, encapsulation, learnability, reuse as well as universality as *efficiency* requirements. While functional requirements strongly influence the API capabilities offered by the GCI, efficiency requirements predominantly impact the GCI architecture. However, the identified functional and efficiency requirements

should not be the sole source for GCI requirements since they were mainly derived from two specific use cases that may not be representative for all GCI usage scenarios. Therefore, we also take into account identified inefficiency problems (cf. Section 1.3) as well as state of the art solutions (cf. Chapter 3).

Inefficiency problems (cf. Section 1.3) encapsulate valuable development experience that should not be neglected in the GCI design phase. In contrast to development requirements, which originated from concrete use cases, inefficiency problems are of a more general nature. First, the lack of a reusable awareness widget library is an issue that has to be addressed by the GCI. Second, the limited applicability of concurrency control libraries with respect to data model support and development approach support represents a major restriction. By contrast, the GCI should support a wide range of web applications. And third, single-user to multi-user transformation scenarios should not only be non-invasive but the GCI should also emphasize the learnability of application conversions.

State of the art solutions (cf. Chapter 3) are a third influencing dimension that we consider when designing the GCI. Each of the four state of the art categories demonstrated specific strengths that we take into account to devise the GCI architecture. First, groupware development libraries readily satisfy the learnability requirement since programmers are usually familiar with their imperative or object-oriented APIs. Consequently, the GCI should offer the ease of use that typical programming libraries nowadays deliver. Second, groupware development frameworks provide a comprehensive toolbox including pre-built workspace awareness widgets. The idea of an advanced toolbox should conceptually be adopted when designing the GCI. Third, transformation approaches illustrate how non-invasive interfaces can be devised in an effective manner. This characteristic should also be realized by the GCI architecture. And fourth, web engineering approaches strongly promote encapsulation and reuse. Therefore, we adopt reuse and encapsulation concepts for the design of the GCI.

4.2 Architecture

After revisiting decisive aspects regarding the GCI design, we outline a naive architecture for collaborative web applications and entailed challenges. Moreover, we show how to overcome these challenges and illustrate the overall GCI architecture. Additionally, we introduce the GCI core components: (1) the DOM adapter, (2) the framework adapter and (3) the workspace awareness adapter.

4.2.1 Challenges

Concurrency control as specified in Definition 2.1 is necessary for effective real-time collaboration. However, providing concurrency control *generically* to satisfy the reuse and universality requirements is challenging.

To illustrate the challenges emerging from the need for generic concurrency control, we introduce two exemplary collaborative editors: a simple text editor and a minimal graphics editor. To properly synchronize document changes and resolve editing conflicts, both editors should benefit from the same application-agnostic concurrency control component. This shared concurrency control component should furthermore rely on the established Operational Transformation (OT) algorithm (cf. Section 2.2.1) since OT is the de facto industry standard (e.g. used by Google Docs, Etherpad, SAP Process Flow, etc.) and OT offers the most advanced concurrency control support in terms of comprehensiveness and sophistication [58].

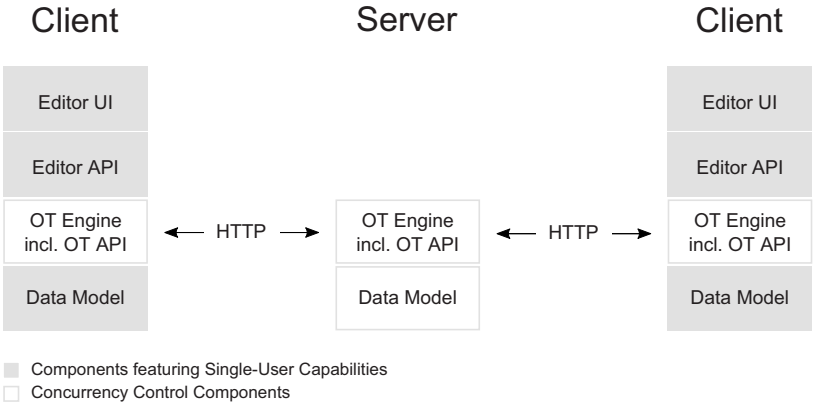


Figure 4.1: Naive architecture for groupware applications

Implementing a collaborative text editor, a multi-user graphics editor or any other shared editing application is commonly carried out by means of the naive coarse-grained architecture depicted in Figure 4.1[1]. To allow multiple users to edit the very same document in parallel, various editor instances are linked to a central server to exchange sync messages in form of OT operations. While grey boxes in Figure 4.1 represent components providing single-user capabilities, white boxes depict the concurrency control component. Client instances consist of a UI supporting the end-user interaction, an editor API accommodating domain-specific operations (e.g. a text editor may provide `insertText` or `deleteText` operations), an OT API

[1] Note that this architecture paradigm is widespread and adopted by most collaborative applications (e.g. SAP Process Flow).

transforming local operations against remote concurrent operations and a data model to store the application's state. If the naive architecture should allow for reuse and support a variety of editors, the following major challenges emerge:

- Editor-Specific Implementation: The heterogeneity of editor operations exposed by different editors induces the need for editor-specific operation support.
- OT Complexity: The OT conflict resolution scheme for numerous sets of domain-specific editor operations requires a vast number of OT transformation functions.
- Invasive Coupling: The invasiveness of introducing concurrency control support entails major learning and development effort.

The heterogeneity of editor operations is demonstrated in Figure 4.2 where the operation sets for the exemplary text and graphics editor are depicted. While the text editor offers an insert character `ins(character c, index i)` and a delete character operation `del(index i)` to author plain text documents, the graphics editor exposes the methods `insShape(polygon p)` and `delShape(reference r)` to create SVG shapes. Thereby, the key challenge is the support for disjoint sets of editor operations. In our example, the text editor operation set O_{TE} encompasses $\{\text{ins}(c,\ i),\ \text{del}(i)\}$ and the graphics editor operation set O_{GE} accommodates $\{\text{insShape}(p),\ \text{delShape}(r)\}$ which results in transformation functions that are also disjoint for the text and the graphics editor (cf. Figure 4.2). Consequently, the client-side concurrency control component is editor-specific with respect to the transformation functions and thus, not suitable for an application-agnostic GCI.

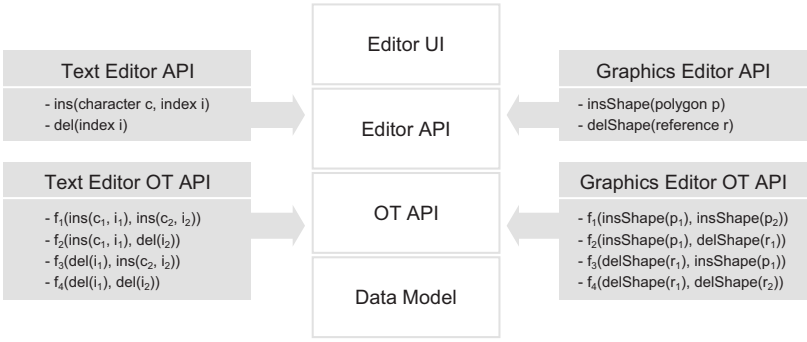


Figure 4.2: Coarse-grained client architecture and corresponding text and graphics editor operations as well as associated transformation functions

Besides requiring specific transformation functions for each supported editor, the naive architecture is also not suited for generic concurrency control because of the large number of required transformation functions. Adopting an OT concurrency control algorithm means that the number of transformation functions grows quadratically with the number of supported editor operations. For the simple text editor offering two operations `ins(c,i)` and `del(i)`, four transformation functions have to be implemented. These four transformation functions cover all combinations of editor operations that can occur in parallel. In Figure 4.3 all combinations of text editor operations are summarized in the transformation matrix.

	$\text{ins}(c_2, i_2)$	$\text{del}(i_2)$
$\text{ins}(c_1, i_1)$	$f_1(\text{ins}(c_1, i_1), \text{ins}(c_2, i_2))$	$f_3(\text{ins}(c_1, i_1), \text{del}(i_2))$
$\text{del}(i_1)$	$f_2(\text{del}(i_1), \text{ins}(c_2, i_2))$	$f_4(\text{del}(i_1), \text{del}(i_2))$

Figure 4.3: OT conflict resolution matrix for a simple text editor with the operation set $O_{TE} = \{\text{ins}(c, i), \text{del}(i)\}$

Figure 4.4 defines the introduced transformation functions that carry out necessary index adaptations to successfully resolve editing conflicts.

$$f_1 = \begin{cases} \text{if } i_1 > i_2 & \{\text{ins}(c_1, i_1 + 1), \text{ins}(c_2, i_2)\} \\ \text{if } i_1 < i_2 & \{\text{ins}(c_1, i_1), \text{ins}(c_2, i_2 + 1)\} \\ \text{if } i_1 = i_2 & \{\text{ins}(c_1, i_1), \text{ins}(c_2, i_2 + 1)\} \end{cases} \qquad f_3 = \begin{cases} \text{if } i_1 > i_2 & \{\text{ins}(c_1, i_1 - 1), \text{del}(i_2)\} \\ \text{if } i_1 < i_2 & \{\text{ins}(c_1, i_1), \text{del}(i_2 + 1)\} \\ \text{if } i_1 = i_2 & \{\text{ins}(c_1, i_1), \text{del}(i_2 + 1)\} \end{cases}$$

$$f_2 = \begin{cases} \text{if } i_1 > i_2 & \{\text{del}(i_1 + 1), \text{ins}(c_2, i_2)\} \\ \text{if } i_1 < i_2 & \{\text{del}(i_1), \text{ins}(c_2, i_2 - 1)\} \\ \text{if } i_1 = i_2 & \{\text{del}(i_1 + 1), \text{ins}(c_2, i_2)\} \end{cases} \qquad f_4 = \begin{cases} \text{if } i_1 > i_2 & \{\text{del}(i_1 - 1), \text{del}(i_2)\} \\ \text{if } i_1 < i_2 & \{\text{del}(i_1), \text{del}(i_2 - 1)\} \\ \text{if } i_1 = i_2 & \{\emptyset, \emptyset\} \end{cases}$$

Figure 4.4: Definition of transformation functions for the OT matrix in Figure 4.3

In contrast to this minimal example necessitating little implementation effort, real-life graphics editors may expose 15 editor operations (e.g. create, delete, move or resize shape, group and ungroup, etc.) leading to a 15 x 15 transformation matrix which translates into 225 transformation functions. Moreover, if numerous editors should be supported by the same OT system, up to several thousands of transformation functions may have to be provided due to the quadratic relation between editor operations and transformation functions. Developing such a complex concurrency control component is neither feasible nor cost-effective.

The last challenge imposed by the naive architecture is the invasiveness of the approach that developers are confronted with. Linking the concurrency control components to the components encapsulating single-user capabilities (e.g. editor API, data model, etc.) requires a plethora of invasive source code changes. These changes essentially affect all functions that modify the data model. Hence, each object creation, manipulation and deletion is subject to source code changes. The development effort does not only arise from carrying out source code manipulations but also from learning the application internals such as the editor API and the data model structure. In essence, the strong coupling of the concurrency control component with interacting components substantially reduces development efficiency which contradicts the defined development requirements (cf. Section 2.3).

4.2.2 Application-Agnostic APIs

The objective of the GCI is to promote development efficiency when implementing collaborative web applications. However, Section 4.2.1 described challenges adopting a naive approach when incorporating generic concurrency control. The identified challenges predominantly originate from linking the concurrency control component directly to the editor-specific API layer (cf. Figure 4.1). Thus, the key for a generic solution is the availability of an application-agnostic API that allows connecting the concurrency control component in an application-independent manner. Therefore, we analyzed a variety of mature and widely adopted web applications to identify common application-agnostic APIs. In the following, we present two widespread approaches that allow accessing and modifying data structures in web applications in an application-agnostic way.

DOM-based Data Models: The first class of applications includes examples like CKEditor [35], SVG-edit [36] or TinyMCE [42] and leverages the Document Object Model (DOM) as well as associated APIs to create and access data models (cf. Section 2.2.3). This class of applications is reflected in the application stack (1) in Figure 4.5 where Editor UI and Editor API are specific for each application but the DOM and the DOM API are independent from the concrete implementation. The DOM Core Specification [40] defines how to construct data models (e.g. `createElement`, `createTextNode`, `removeAttribute`, etc.) and the associated DOM APIs specify, among others, the event system (DOM Events Specification [106]), model traversal and content access (DOM Traversal and Range Specification [107]), model presentation (CSS Specification [108]), etc. DOM specifications are issued by the World Wide Web Consortium (W3C) and they are widely adopted and implemented by all modern browsers (e.g. Google Chrome, Internet Explorer or Mozilla Firefox). The broad acceptance makes the DOM and its API a unique platform-neutral interface that allows linking the concurrency control component in a generic fashion.

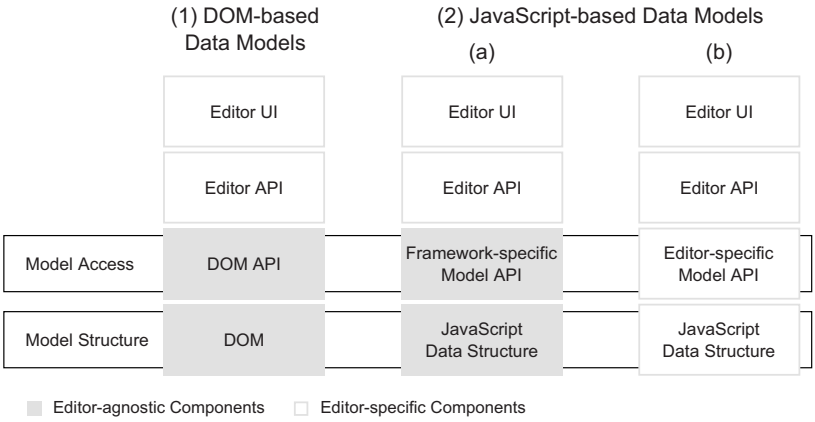


Figure 4.5: Classification of web applications with respect to data model access and data model structure

JavaScript-based Data Models: In contrast to data models that are materialized by the DOM, the second class of web applications (e.g. Eclipse Orion [43] or the Ace editor [109]) uses JavaScript data structures to represent data model instances. This class of web applications can be further divided into (2a) framework-based applications and (2b) pure JavaScript applications (cf. Figure 4.5). On the one hand, framework-based applications are built upon frameworks like Knockout [30], Backbone [37], SproutCore [110], etc. Web applications that rely on the same framework also leverage the same uniform interface for data access and manipulation. Thus, the application-agnostic API exposed by a framework represents a second effective means to link a generic concurrency control component. On the other hand, pure JavaScript applications can construct data models and its interfaces in an arbitrary manner that is solely constrained by the JavaScript language vocabulary. JavaScript-based web applications exposing individual model APIs require tailored concurrency control support and are therefore not suitable for devising the GCI.

After identifying application-agnostic APIs, we illustrate how the described challenges can be overcome adopting generic APIs. First, the editor-specific implementation to incorporate concurrency control becomes obsolete since the plethora of applications can leverage a single concurrency control adapter that links to the application-agnostic API (e.g. the DOM API or a framework API). Hence, the time-consuming task of mapping a specific editor API to the concurrency control API is no longer required. Second, the OT complexity induced by the vast set of necessary transformation functions is also not an issue anymore because the number of transformation functions does not increase with the number of supported editors. For ex-

ample, to support several applications with DOM-based data models only the transformation functions for the DOM operations are required which is feasible with reasonable effort. And third, the invasiveness can also be overcome since application-agnostic APIs such as the DOM API offer means to record and replay model changes. Consequently, synchronizing model instances, which necessitates record and replay mechanics, does not require leveraging the editor-specific API and invasively adapting the editor's source code. This logic can be encapsulated in a general-purpose record and replay adapter (e.g. a DOM adapter) serving multiple applications.

4.2.3 Derived Architecture

The described design considerations and the need for an application-agnostic API layer led to the derived GCI architecture shown in Figure 4.6.

The first design principle that is realized by the high-level architecture is the clear separation between single-user editor components represented as white boxes and multi-user components shown as grey boxes. This conscious choice was inspired by single-user to multi-user transformation systems (e.g. transparent adaptation [39]) where non-invasiveness is a key requirement. Hence, the white components can be regarded as fully functional single-user editors exposing a UI for end-user interaction, an editor API encapsulating domain-specific operations (e.g. `insertShape`, `deleteShape`), an application-agnostic API accommodating low-level methods and the actual data model. Furthermore, the grey components accompany single-user editor components in a non-invasive manner.

Besides the clear separation of single-user and multi-user components, a second core principle is that all communication between editor and GCI components is channeled through an application-agnostic interface. If the Concurrency Control Adapter (CCA) and the Workspace Awareness Adapter (WAA) were linked to the editor API, the aforementioned challenges would emerge again and the implementation of a generic collaboration infrastructure would not be feasible.

A third design principle that led to the GCI architecture in Figure 4.6 is the focus on the web as runtime environment. The distributed nature of the web is a natural fit for collaborative applications. Therefore, the communication protocols for client-server interactions are deliberately limited to HTTP and the WebSocket protocol. The bi-directional communication capabilities and the minimal overhead qualify the WebSocket protocol as the preferred solution (cf. Section 2.2.3) and HTTP as fallback variant.

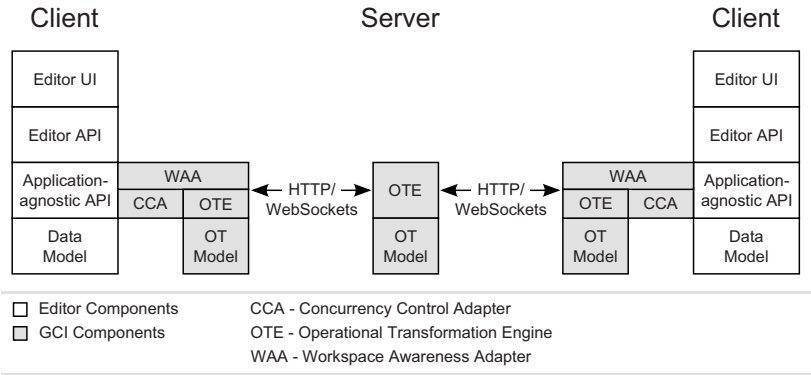


Figure 4.6: Overall GCI architecture

The overall GCI architecture shown in Figure 4.6 embraces the application-agnostic API design and thereby relies on three API services: (1) change tracking, (2) operation replay and (3) widget integration. Change tracking, on the one hand, is required by the CCA to propagate model modifications to the Operational Transformation Engine (OTE) that initiates the document synchronization process. On the other hand, it is a prerequisite for the WAA where model or UI changes trigger awareness widget updates (e.g. adapting the creation coloring range or the telepointer position). The operation replay service is necessary to allow the CCA carrying out the last step of the synchronization process. Initially, local data changes are recorded as well as propagated to all remote clients and eventually the operation replay takes place. The last service enables the widget integration where, in particular, workspace awareness widgets such as participant lists, radar views, etc., have to be incorporated in the UI of the shared workspace.

Besides identifying application-agnostic APIs (DOM APIs, framework APIs) and deriving an abstract GCI, a major research contribution of this dissertation is the architecture of the CCA and WAA components. These adapter components form the GCI foundation that provides concurrency control and workspace awareness functionality. Moreover, these adapter components are essential for the development efficiency because the degree of required adapter integration effort determines whether the approach can substantially increase efficiency. For example, the effort to link a web application to a GCI adapter may range from completing a simple configuration file to implementing a specific integration module. In subsequent sections, we explore ways that require minimal effort to integrate GCI adapters with web applications.

4.3 Core Components

While the last section introduced the high-level GCI architecture and illustrated the component interplay, we now briefly discuss the GCI core components: (1) the DOM adapter, (2) the framework adapter and (3) the workspace awareness adapter. The DOM adapter and the framework adapter are both CCA embodiments offering concurrency control functionality. Both adapters rely on different APIs (the DOM API or a framework API). The workspace awareness adapter provides workspace awareness capabilities in a generic manner interoperating with both concurrency control adapters.

4.3.1 DOM Adapter

The standardized DOM interface and associated APIs are a prerequisite for application-agnostic document synchronization and conflict resolution. To implement concurrency control an adapter bridging the gap between the OTE and the application's data model is necessary (cf. Figure 4.6). The DOM adapter acts as the DOM API - OTE bridge capturing DOM changes that are of interest, propagating those changes to the OTE and replaying recorded changes remotely.

To illustrate the role of the DOM adapter, we employ an exemplary graphics editor where a user creates a rectangle shape. Figure 4.7 depicts the established editor layers (Editor UI, Editor API, DOM API, DOM) as well as the top-down command translation. The creation of the rectangle starts with a user interaction where an end-user constructs a rectangular shape that becomes immediately visible on the editor UI. This user interaction is mapped to the editor API call `drawRectangle` that gets split into numerous DOM API calls. The `createElementNS` and `setAttribute` calls finally provoke DOM modifications, i.e. the highlighted `<rect>` element is added to the `<svg>` node.

In this example, the DOM capture process has to record three DOM API calls that construct and adapt the `<rect>` node. Therefore, the DOM adapter utilizes the DOM notification system allowing registering event listeners on DOM nodes. The specific class of DOM events that announce DOM manipulations are the so called DOM Mutation Events which comprise notifications about node insertions, node removals, attribute changes and character modifications [106]. Once DOM changes are recorded, DOM operations are translated into OT operations. The OTE serializes and distributes the OT operations to all remote clients but before replaying the OT operations, an essential step is the transformation against concurrent OT operations which allows resolving potential editing conflicts (cf. Section 2.2.1). Transformed OT operations are mapped to the corresponding DOM operations which in our example are the same DOM calls that are shown

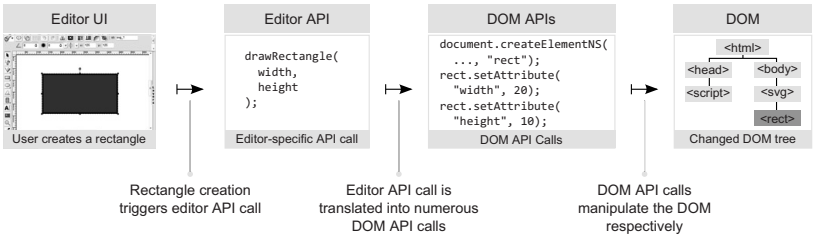


Figure 4.7: Exemplary translation of an editor-specific function call into standardized DOM API calls

in Figure 4.7[2]. Carrying out these DOM operations leads to a synchronized and consistent application state among all participating clients.

This introduction to the DOM adapter establishes a basic understanding for the responsibilities of the specific concurrency control adapter. Chapter 5 goes beyond this brief discussion and elaborates thoroughly on DOM adapter concepts, implementation details, the evaluation procedure as well as evaluation results.

4.3.2 Framework Adapter

In addition to the DOM adapter, another CCA embodiment is the framework adapter. While the DOM adapter supports web applications with DOM-based data models, the framework adapter provides concurrency control capabilities for web applications that are built on top of a framework and that facilitate a JavaScript-based data model.

Figure 4.8 illustrates the different approaches for data model implementations. The graphics editor in Figure 4.8a uses a DOM-based data model that is represented by the subtree spanned by the `<svg>` element. By contrast, Figure 4.8b represents a text editor where multiple paragraphs can be visualized in the `Text` node that is linked to the `<textarea>` element. Even though the DOM is exploited as a view component in form of the `<textarea>` element, the data model is materialized as an isolated JavaScript data structure adhering to the UML class diagram in Figure 4.8b. Isolating the view from the data model is a common pattern known as the Model-View-Controller (MVC) pattern [111] where the controller updates the model upon view changes and vice versa.

A significant share of web applications adopts the MVC pattern. To increase the applicability of the GCI, we aim to support MVC web applications using a dedicated framework adapter. The framework adapter works in a similar way as the DOM adapter, i.e. results of application-specific op-

[2]Note that parameter values may vary in the replay function calls since they might have been adapted through OT transformations.

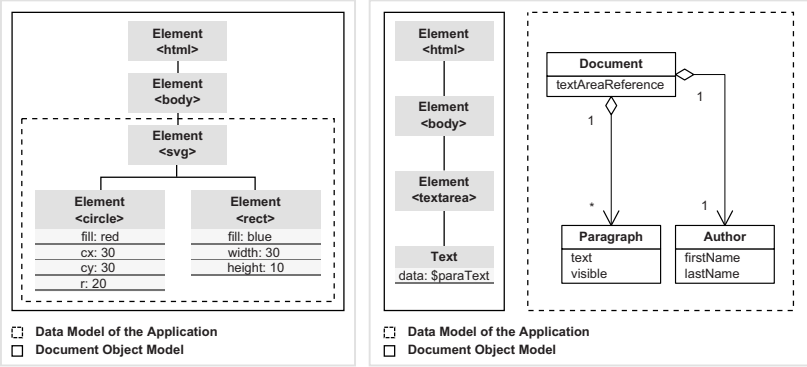


Figure 4.8: Example of a DOM-based data model and an isolated JavaScript-based data model

erations are applied to the data model and thereby, model changes trigger the synchronization that exploit a generic OTE. However, to leverage the GCI, framework-based MVC applications have to fulfill the following set of requirements:

- Data Model Isolation: To synchronize numerous data model instances in a device- and browser-independent manner, the data model should be isolated.
- Notification Mechanism: To record model changes, a notification mechanism should be provided by the framework API.
- Traversable Data Model: To access the data model in its entirety, the data model should be traversable from a single entry point.

A detailed discussion about frameworks that satisfy these requirements is presented in Chapter 6. Moreover, Chapter 6 also describes the framework adapter architecture, the implementation, the evaluation as well as the applicability.

4.3.3 Workspace Awareness Adapter

In contrast to the CCAs that allow synchronizing application states and resolving editing conflicts, the workspace awareness adapter is crucial for participants to understand how others interact with the shared space. The prevalent means to establish an understanding about other participants are workspace awareness widgets (e.g. participant list, radar view, etc.) and therefore, the WAA is primarily a container for reusable awareness widgets.

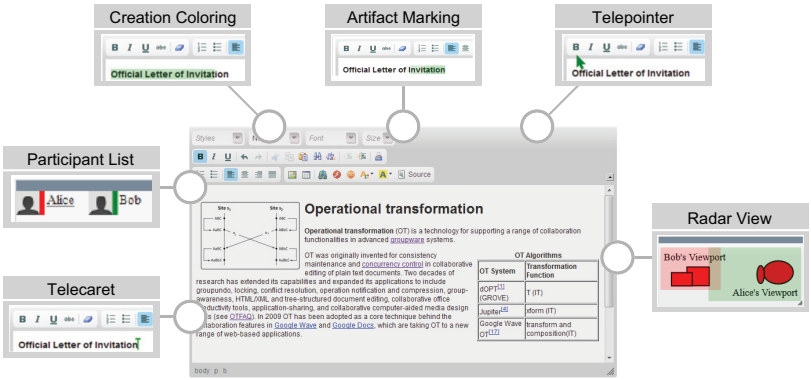


Figure 4.9: Awareness widget examples enhancing a text editor

Figure 4.9 shows for example a text editor and a selection of awareness widgets that are useful in shared editing scenarios. The participant list, the telecaret, the telepointer, the radar view as well as the widgets for creation coloring and artifact marking all convey information about other participants in the shared workspace. The conveyed information considers aspects like location, presence, authorship, etc. (cf. Section 2.2.2).

To facilitate and operate a reusable awareness widget library for web applications, the workspace awareness adapter mainly accommodates two types of modules: event modules and widget modules.

- Event Modules: Capturing relevant awareness information is the task of event modules. Therefore, application-agnostic APIs (e.g. DOM APIs) are leveraged providing low-level events (e.g. `keydown`, `mouseover`, or `scroll` events) that are translated into a set of predefined awareness events such as `CaretMove`, `ViewportChange`, etc. Awareness events abstract from the actual underlying event source and can thus be handled in a uniform way.

- Widget Modules: Awareness events originating from event modules are consumed by widget modules that process and visualize incoming awareness information. To receive notifications, widget modules have to subscribe to awareness events. Each widget module materializes a specific awareness widget (e.g. telepointer, telecaret, artifact marking, etc.) implementing the `IWidget` interface. Exposing a fixed `IWidget` interface also allows extending the library of awareness widgets.

Those core modules are accompanied by the initialization, the identification, the participant, the network and the operation origin module that are explained in Chapter 7. Furthermore, Chapter 7 discusses the detailed WAA architecture, the WAA evaluation as well as the WAA applicability.

4.4 Development Methodology

In addition to the proposed GCI, we also establish a methodology to support developers adopting the GCI and its core components. Primarily, the methodology guides programmers through transformation as well as from-scratch development scenarios.

In Section 2.1, we introduced two use cases for the development of collaborative web applications. On the one hand, we outlined the capabilities of the single-user graphics editor SVG-edit and defined functional requirements that allow leveraging SVG-edit for collaborative work. On the other hand, we specified the envisioned CoBAT application representing a tool to collaboratively carry out cost-benefit analyses that has to be implemented from-scratch.

Even though both use cases aim to develop collaborative web applications, the requirements for the two development approaches differ. While an efficient approach for transformation scenarios requires non-invasiveness since it cannot be assumed that this evolution step is processed by the original developers; developing shared editing applications from scratch can assume source code familiarity and therefore, the non-invasiveness requirement is not of utmost importance. For the from-scratch implementation it is important that the integration of multi-user features is streamlined with the overall development process.

Hence, the overall methodology[3] depicted in Figure 4.10 varies the strategy to incorporate collaboration functionality. Developers have to first decide whether they want to transform an existing web application or build a new collaborative web application from scratch. Once the scenario type is fixed, the subsequent methodology steps as depicted in Figure 4.10 differ to a large extent.

For transformation scenarios, where non-invasiveness is crucial, not all kinds of web applications are supported. Therefore, the necessary criteria check (cf. Section 5.4.1) determines whether the existing web application is eligible to leverage the GCI. In particular, web applications have to fulfill the requirements induced by the DOM adapter, i.e. the W3C standards compliance and the existence of a DOM-based data model. W3C standards compliance means that web applications rely on open W3C standards (e.g. the DOM Core [40] or the DOM Events Specification [106]) rather than on plugin technologies (e.g. Adobe Flash or Microsoft Silverlight) that do not expose application-agnostic APIs and thus, are not suited for the adoption of the DOM adapter. The DOM-based data model represents a second necessary criterion since the DOM adapter is solely capable of tracking and replaying changes targeting the DOM.

[3]Note that even though we introduce the overall methodology in this section, we refine the corresponding methodology part when discussing the respective GCI core component in the Section 5.4.1, 6.4.1 and 7.2.5.

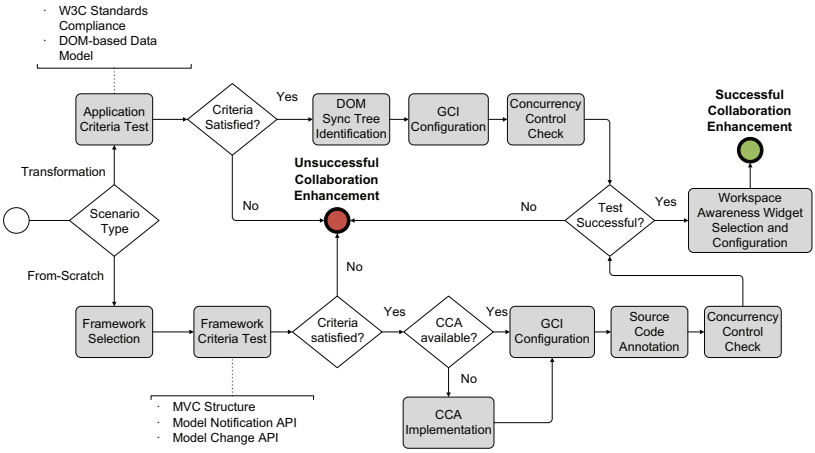


Figure 4.10: Overall development methodology

If the necessary criteria check is successful, the actual integration of the web application and the DOM adapter takes place (cf. Section 5.2.1). First, the DOM subtree that represents the data model has to be identified. While the data model accommodates content artifacts (e.g. text or graphic objects), other DOM elements embodying UI elements such as toolbar icons, scrollbars, resize handles, etc., should not be synchronized among all participants and therefore, have to be excluded from the data model. Second, developers need to configure the GCI which includes specifying various properties, e.g. the sync server URL or the identifier of the DOM sync tree representing the data model. Moreover, the GCI has to be anchored in the original web application. A final test of the DOM synchronization functionality concludes the transformation-specific methodology. This check validates the proper sync of multiple DOM instances and may reveal issues associated to the DOM sync (e.g. performance degradation for vast sets of DOM changes, improper synchronization behavior, etc.).

After the DOM synchronization is setup, the process of workspace awareness integration (cf. Section 7.2.5) follows. Note that this subprocess is equal for transformation and from-scratch development projects. It comprises selecting and configuring workspace awareness widgets that are encapsulated by the workspace awareness adapter. The selection from the awareness widget library[4] allows adapting the shared workspace to the end-users' needs. For example, a telepointer widget might support efficiency in a shared draw-

[4]The workspace awareness widget library encompasses a participant list, a telepointer, a radar view, a telecaret as well as widgets for creation coloring and artifact marking.

ing session with few participants but can also be confusing if a large number of participants collaborate in the same shared workspace.

In addition to the transformation branch in Figure 4.10, there is also the from-scratch branch illustrating the methodology for new groupware development projects (cf. Section 6.4.1). In contrast to transformation projects where non-invasiveness is essential, the prevalent characteristic for from-scratch projects is the seamless integration with existing development approaches. Nowadays, developers commonly adopt web development frameworks[5] to exploit higher-level functionality and to abstract from browser-specific implementations (e.g. Internet Explorer, Mozilla Firefox, etc.) or concrete form factors (e.g. PCs, tablets or smartphones). To support established web development processes and to exploit the familiarity of developers with widely adopted frameworks, the methodology in Figure 4.10 allows developers to adopt their web framework of choice. That is a key aspect for the from-scratch methodology to support development efficiency.

Thereafter, in line with the transformation approach, necessary criteria addressing web development frameworks have to be verified (cf. Section 6.4.1). Eligible frameworks have to structure web applications in a MVC fashion and have to expose a model notification API and a model manipulation API. First, the encapsulated data model exposed by MVC applications is crucial to synchronize numerous application instances in a device- and browser-independent manner since application models are a means to store data without including specifics about their presentation. Second, web frameworks should offer a notification API to inform about model changes and a manipulation API to change the data model. Notification and manipulation API are required to record and replay model changes.

The next methodology activity asks developers to check whether a framework-specific CCA implementation is already available. If that is not the case, the CCA for the selected framework has to be implemented. Note that this CCA is specific for a dedicated framework but not for the actual application, i.e. collaborative applications built on top of the same framework leverage the very same CCA.

To link the GCI and the web application, a GCI configuration is again required to define basic properties such as the sync server URL. Depending on the framework adapter flavor (cf. Section 6.2), the configuration may also include the step of annotating the web applications source code. A specific annotation language (encompassing expressions like @Sync, @Class, etc.) allows enriching the source code. Annotations are a means to mark partial data models that shall be synchronized, to specify constructor functions for replay operations, etc. After the configuration activity, a check validates the proper GCI operations, in particular, regarding the synchronization and the

[5]For example, according to Web Technology Surveys the jQuery framework [27] is used by 54.8 percent of all web pages [112].

conflict resolution capabilities. As depicted in Figure 4.10, the remaining methodology steps representing the workspace awareness configuration are identical for transformation and from-scratch projects.

The presented methodology introduced the basic building blocks for both supported development approaches and highlighted the differing objectives. While the transformation methodology emphasizes that the source code does not require changes and that the GCI adoption is non-invasive; the from-scratch methodology aims to streamline the development process exploiting the benefits of web frameworks and leveraging the familiarity of developers with respect to widespread development frameworks.

4.5 Summary

In this chapter, we proposed an approach to support development efficiency for web-based real-time groupware that is based on the generic collaboration infrastructure. Thereby, we first revisited decisive aspects like development requirements for concrete use cases, common inefficiency challenges in groupware implementation projects and related state of the art approaches that influenced the GCI design. Second, we discussed challenges offering reusable concurrency control and showed how these challenges can be overcome exploiting application-agnostic APIs. Third, the overall GCI architecture and the GCI core components (the DOM, the framework and the workspace awareness adapter) were described. Finally, the associated methodology for transformation as well as from-scratch scenarios was specified.

Chapter 5

DOM Adapter

In this chapter, we will elaborate on the DOM adapter [104, 105, 113] that allows transforming single-user web applications into collaborative ones. First, we present research questions regarding the DOM adapter approach that target the technical feasibility, the end-user acceptance and the applicability to web applications. Second, we revisit the first research question and demonstrate the technical feasibility of the DOM adapter approach. Therefore, we introduce the DOM adapter architecture as well as processes that are carried out by the DOM adapter. Third, we present our evaluation approach and report on a usability study that aims to assess end-user acceptance. Fourth, we investigate the applicability by showing web applications that can leverage the DOM adapter and by discussing DOM adapter limitations.

5.1 Research Questions

To define the research focus for the DOM adapter discussion, we specify research questions that we will examine throughout the chapter. The set of research questions comprises the following three:

- **Research Question 1:** Is the design of a DOM adapter in a generic and non-invasive fashion technically feasible?
- **Research Question 2:** Does a generic, non-invasive DOM adapter appropriately support real-life collaborative editing sessions?
- **Research Question 3:** Can a generic, non-invasive DOM adapter be adopted by a large set of single-user web applications?

Research Question 1 analyzes, on the one hand, if the end-to-end capture, propagate and replay process can be implemented at all on top of standardized DOM APIs and, on the other hand, if that is achievable in a non-invasive fashion. Consequently, we study whether relevant DOM modifications (e.g. insert or remove DOM nodes, change DOM attributes, etc.)

can be recorded leveraging the DOM Events API [106]. Moreover, we inspect whether the means provided by the DOM Core API [40] are sufficient to carry out replay operations. To satisfy the concurrency control requirement (cf. Section 2.3), we investigate how DOM changes can be mapped to OT operations and vice versa. And again, we explore non-invasive means to realize the DOM adapter functionality, i.e. changes to the source code of the original web application are not necessary.

Research Question 2 aims to verify whether a DOM adapter as part of the GCI (cf. Section 4.2.3) suits the needs of end-users in real-life collaboration sessions such as shared text editing or shared drawing. Note that end-user acceptance, even though it requires the technical feasibility, goes beyond the discussion of research question 1. For example, even though a DOM adapter can be constructed, the high load of DOM changes may impair overall system performance and thus, the resulting end-user experience might be poor. Analyzing research question 2 requires a proper evaluation design including the selection of editors, study participants, collaborative tasks as well as evaluation characteristics (e.g. reliability, usability, learnability, etc.). Additionally, evaluation results have to be interpreted to allow for meaningful conclusions.

Research Question 3 targets the viability of the approach, i.e. what fraction of single-user web applications can effectively be supported by the DOM adapter. Therefore, an analysis of the web application landscape is necessary, in particular, in terms of collaboration domains (e.g. collaborative writing, shared graphics editing, etc.) and potential transformation candidates. Concrete single-user web applications have to be identified and it has to be investigated to what extent these applications may adopt the DOM adapter. Moreover, the discussion of this research question also includes determining DOM adapter limitations.

5.2 DOM Adapter Architecture and Integration

The objective of the DOM adapter is to promote reuse by supplying collaboration services (real-time model synchronization as well as automatic conflict resolution) to standards-based web applications. The DOM adapter design targets the reuse challenges (cf. Section 4.2.1) that disallow providing generic concurrency control support. Heterogeneous editor-specific operations, the OT complexity induced by vast sets of editor operations as well as the invasiveness of a naive approach can only be overcome if an application-agnostic API instead of an editor-specific API is adopted. Thus, the DOM adapter is exclusively built on top of standardized DOM APIs (e.g. DOM Core [40], DOM Events [106], etc.) satisfying the application-agnostic API requirement (cf. Section 4.2.1).

To validate our assumption that an application-agnostic collaboration infrastructure is technically feasible, we devised the DOM adapter architecture shown in Figure 5.1, which is derived from the high-level GCI architecture (cf. Figure 4.6). The architecture exposes three kinds of components: the single-user editor components, DOM adapter components as well as SAP Gravity components. Like in Figure 4.6, the original single-user editor consists of four layers whereas the application-agnostic API layer is represented by DOM APIs and the data model is embodied by the DOM. Moreover, the concurrency control adapter depicted in the GCI architecture in Figure 4.6 is represented by numerous DOM adapter components. They are in charge of initializing the DOM adapter (DOM Adapter Initializer), recording DOM manipulations (DOM Change Recorder), mapping DOM operations to OT operations and vice versa (Operation Transformer) as well as replaying local DOM manipulations remotely (DOM Manipulator). The third class of components is represented by SAP Gravity components materializing the OTE. We made the design decision to exploit SAP Gravity [29] as the operational transformation engine of choice since it has a proven track record of serving real applications, e.g. being adopted by the industrial-strength product SAP Process Flow [29]. However, instead of using SAP Gravity, other OTEs can also be adopted.

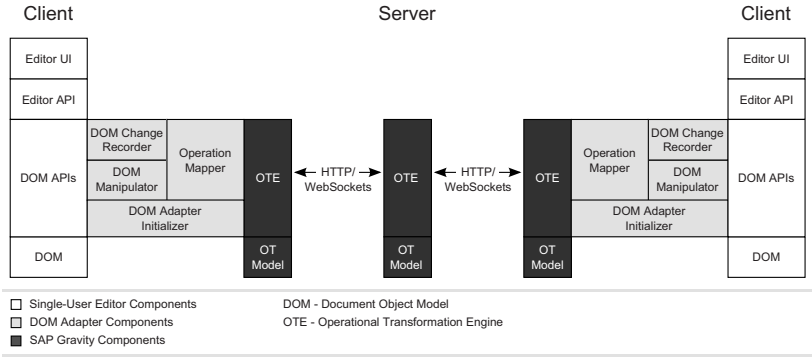


Figure 5.1: DOM adapter architecture

In the following, we will describe the individual DOM adapter components, the component interplay as well as the underlying processes that these components implement (e.g. capture or replay DOM changes). Therefore, we introduce a tangible example of a minimal graphics editor that will be revisited throughout the component description sections.

In this example, the DOM adapter should be adopted to convert the single-user graphics editor depicted in Figure 5.2 into a multi-user version. Thus, numerous participants can freely edit the very same graphic simultaneously. Figure 5.2 shows three representations of the original single-user

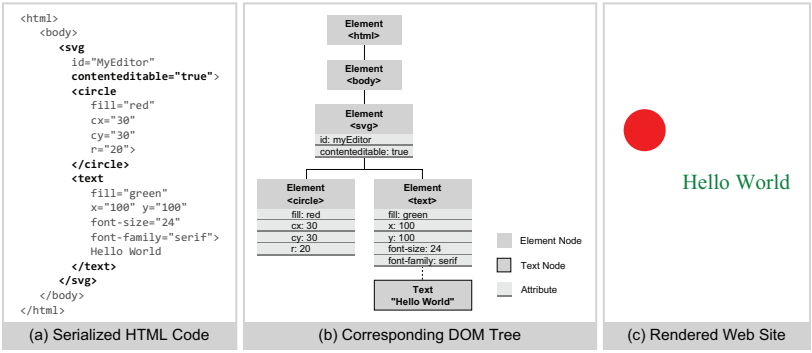


Figure 5.2: Minimal graphics editor example

graphics editor: the HTML source code, the DOM as well as the rendered web site. The current content of the graphics editor comprises a circle shape and a text node that are accommodated in a `<svg>` element node. All content encapsulated by the `<svg>` element can be modified since the `contenteditable` attribute is set to `true`. Note that the `contenteditable` attribute is interpreted by the browser and therefore, the browser engine determines how the editor palette looks like and what options are offered to change existing content[1]. In the specified form, the minimal graphics editor is limited to single-user scenarios. However, collaboration capabilities could broaden the graphics editor adoption and thus, we will revisit this example and show how the graphics editor can be transformed into a multi-user tool supporting shared drawing.

5.2.1 DOM Adapter Incorporation

Single-user to multi-user editor conversions require incorporating the DOM adapter and client-side SAP Gravity components into the original single-user editor. To ease the incorporation of these components, multi-user capabilities are bundled in one dedicated JavaScript dependency file: the `gci.js`. This `gci.js` has to be embedded in the editor's HTML code. For instance, for the exemplary graphics editor, Figure 5.3a depicts the inclusion of the required JavaScript dependency. Once a browser loads the updated HTML page, the `gci.js` is executed by the browser's JavaScript engine.

Besides facilitating the required multi-user capability logic, the `gci.js` script also references a default configuration file `gci-config.js` that has to be completed to capture configuration properties. Currently, the configura-

[1]Note that most browsers nowadays only support the `contenteditable` attribute for text elements (e.g. `<p>` or `<h1>` elements). However, browser implementations will evolve and also support the `contenteditable` attribute in the context of the `<svg>` element.

```
<html>
  <head>
    <script type="text/javascript" src="gci.js">
    </script>
  </head>
  <body>
    <svg id="MyEditor" contenteditable="true">
      ...
    </svg>
  </body>
</html>
```

(a) Editor HTML Skeleton

```
// gci-config.js Configuration File //
//////////////////////////////////////

// Sync Server URL Configuration
sync-url =
 "http://www.tu-chemnitz.de/mysyncserver";

// DOM Sub-Tree Selector Configuration
dom-node-id = "MyEditor";
// more DOM nodes could be added here
```

(b) Configuration File

Figure 5.3: DOM adapter incorporation and configuration for the exemplary graphics editor (cf. Figure 5.2)

tion requires setting the sync server URL as well as defining DOM subtrees that should be included in the DOM synchronization.

Figure 5.3b shows the definition of a custom sync server URL. Additionally, the configuration file accommodates DOM subtree selectors that reference DOM nodes using identifiers. While the specified DOM nodes and the spanned subtrees are synchronized, not encompassed DOM nodes are excluded from the synchronization. Application developers have to distinguish between application parts that are included in the sync and those that are not included, i.e. changes only affect the local workspace. For example, a collaborative text editor should propagate all manipulations targeting the actual text document. However, selecting the bold or italic formatting option in the toolbar should not affect the workspace of all participants. In the configuration file in Figure 5.3b the `<svg>` root node `MyEditor` is marked as the DOM subtree selector, i.e. changes to the root node and its descendants are synchronized.

In essence, the DOM adapter incorporation only requires to embed a specific JavaScript file and to adapt an additional configuration file.

5.2.2 DOM Adapter Initialization

After embedding and configuring the DOM adapter, the Initializer Component (cf. Figure 5.1) is in charge of setting up all routines that allow the DOM adapter to continuously capture, propagate and replay DOM changes. Initializing the DOM adapter consists of the following mandatory steps:

1. Model Identification Setup: Assign unique identifiers to DOM model nodes to establish a system-wide referencing mechanism.

2. Notification Mechanism Initialization: Install DOM listeners on relevant nodes to be informed about DOM changes.

3. OT Model Initialization: Construct an initial OT data model that is in sync with the corresponding DOM.

Model Identification Setup

The initializer component starts with the routine to establish a Global Identification Scheme (GIS). The GIS is a prerequisite for replaying DOM manipulations because it represents a means to refer to a DOM node unambiguously at all sites. For example, if a user resizes a circle using a collaborative graphics editor, this change has to be replayed among all sites. Assuming that the SVG document already contains numerous circles, an unambiguous reference is necessary to assure that the replay operation *resize circle* addresses the correct circle.

Therefore, the initializer component exposes GIS functionality capable of identifying the relevant DOM artifacts that are *element nodes*, *text nodes* and *attributes* (cf. Figure 5.2b). To support element nodes, the initializer component generates Universally Unique Identifiers (UUIDs) and assigns them to the `id` attribute of the element node. However, for text nodes and attributes there is no `id` attribute that can be set. Thus, the parent is exploited since the parent of a text node or attribute is always an element node with a unique ID. For text nodes, the combination of the parent node ID and the index position within the list of attached text nodes constitutes an unambiguous reference. For attributes, combining the parent node ID with the attribute `name` also represents a unique key to identify a DOM attribute.

Notification Mechanism Initialization

Once the GIS setup is completed, the initialization of the notification mechanism follows. To get notified about DOM changes (e.g. the insertion or removal of a DOM node), listeners and event handlers have to be registered on relevant DOM nodes. Relevant nodes are the ones which are included in the DOM synchronization process. For example, in Figure 5.2b changes regarding the `MyEditor` element and the respective descendant nodes are relevant and therefore, they have to be observed. To execute the listener registration and to handle fired events, the Initializer Component leverages the DOM Change Recorder component. Details about the setup and the operations mode of the notification mechanism are exposed in Section 5.2.3 discussing the DOM Change Recorder.

OT Model Initialization

The last process step executed by the initializer component is the creation of the OT model that adheres to the existing DOM. This step is required since we reuse the Gravity OTE[2]. The Gravity OTE allows synchronizing

[2]The GCI architecture is by no means bound to a specific OTE implementation. For example, the first GCI prototype was built on top of the Apache Wave OTE [32].

graph-structured documents. Thereby, SAP Gravity maintains document consistency and resolves editing conflicts automatically. Since the Gravity OTE is tightly coupled to a specific OT model, using the concurrency control services necessitates mapping the DOM tree structure to a Gravity graph structure. Because a tree represents a subset of a graph, this mapping is technically feasible. The initializer component initiates the OT model construction and then hands over the actual Gravity model creation to the Operation Mapper component. Thus, details about the mapping of DOM nodes to Gravity nodes are exhibited in Section 5.2.4 introducing the Operation Mapper.

5.2.3 DOM Change Recorder

As mentioned in the previous section, the DOM Change Recorder's first responsibility is the setup of the notification mechanism. In addition to the setup, the continuous propagation of relevant DOM changes to the operation mapper represents a second responsibility.

As depicted in Figure 5.1, the DOM change recorder is built upon DOM APIs, in particular, on top of the DOM Events API [106]. The DOM Events API, which is supported by all modern browsers, allows monitoring a variety of events such as UI events (resize, scroll), mouse events (click, mouseover), keyboard events (keydown, keyup) or mutation events (DOMNodeInserted, DOMNodeRemoved). Nevertheless, for the synchronization of multiple DOM copies, only mutation events are of interest since they report changes regarding the DOM instead of informing about user interactions (e.g. that the mouse pointer was moved). Currently, the change recorder implementation supports all DOM mutation events (i.e. DOMNodeInserted, DOMNodeRemoved, DOMAttrModified and DOMCharacterDataModified) that are defined in the DOM events specification [106].

To illustrate the task of setting up the notification mechanism, Figure 5.4 shows a skeleton of the implementation. This code snippet demonstrates how DOM mutation event listeners are attached to the DOM element `MyEditor` (cf. Figure 5.2). The `addEventListener` method takes two arguments. While the first argument represents the event type, the second

```
// select the node that should be observed
var rootNode = document.getElementById("MyEditor");

// register the DOM mutation event handler functions
rootNode.addEventListener("DOMNodeInserted", function () { ... });
rootNode.addEventListener("DOMNodeRemoved", function () { ... });
rootNode.addEventListener("DOMAttrModified", function () { ... });
rootNode.addEventListener("DOMCharacterDataModified", function () { ... });
```

Figure 5.4: Registration of DOM mutation event handlers

argument is the actual handler function which is executed once the event is fired. Note that the registration of DOM mutation event handlers is not required for each DOM node since registered handlers fire upon changes regarding the node where the handler is registered but also upon changes regarding respective descendant nodes. Hence, the event handler registration in Figure 5.4 is capable of monitoring all DOM changes affecting the `MyEditor` node or the associated subtree.

The second task of propagating DOM changes is shown in form of a coarse-grained process model in Figure 5.5. Besides the process steps itself, the responsible DOM adapter components are also depicted. A DOM manipulation triggered by a user interaction fires a DOM mutation event which notifies the corresponding listener. The listener logic has to extract information about the DOM manipulation (e.g. the ID of a affected DOM node) and forwards this extracted information to the operation mapper. In the next step, the operation mapper has to convert the DOM change to a corresponding Gravity operation.

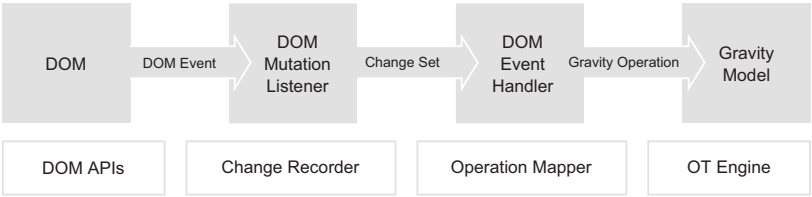


Figure 5.5: DOM change recording process

5.2.4 Operation Mapper

The Operation Mapper component has two functions: on the one hand, during the initialization of the DOM adapter the current DOM needs to be mapped to a Gravity model representation; on the other hand, in the operations mode of the DOM adapter, DOM changes have to continuously be transformed to Gravity model changes and vice versa.

In order to execute on both of these tasks, the operation mapper has to implement a mapping that transforms DOM elements into Gravity elements and vice versa. The essential DOM elements that have to be converted into a Gravity model representation are *element nodes*, *element attributes*, *text nodes* and *parent-child relations* linking nodes. The Gravity Model (GM) provides *nodes* bound to a unique ID, *attributes*, *atomic references* and *ordered references*. While atomic references point to exactly one node, ordered references are capable to establish links to a set of nodes. The devised mapping adheres to the following rules:

1. A DOM element node is represented by a GM node. If the element node does not yet have an ID, a UUID is generated and assigned to the DOM element node as well as to the GM node. Moreover, atomic element values referencing a single object such as `nodeType` or `nodeName` are expressed using GM attributes.

2. A DOM element attribute is represented by an atomic reference and a GM node whereas the GM reference name is the attribute name and GM node attributes hold the atomic attribute values such as `nodeType`, `nodeValue`, etc.

3. A DOM text node is represented by a GM node and atomic values such as `nodeType` or `data` are expressed as GM node attributes.

4. A DOM parent-child relationship associating DOM nodes (e.g. element and text nodes) is represented by a GM ordered reference.

Assigning the same identifier to a DOM node and to the corresponding GM node establishes a bijective mapping, i.e. a DOM node can unambiguously be associated to a GM node and vice versa. Therefore, DOM manipulations can be captured and represented in the GM. Furthermore, changes to the GM can also be replayed in the DOM.

Even though the mapping rules are specified on DOM and GM nodes, the actual mapping produces a set of operations including calculated parameter values (cf. Figure 5.5). Thus, the implementation of the operation mapper component determines Gravity operations that correspond to DOM changes or DOM operations that correspond to GM changes.

Figure 5.6 shows an example of a DOM-to-GM mapping. The SVG element introduced in Figure 5.2 and its child nodes are transformed into

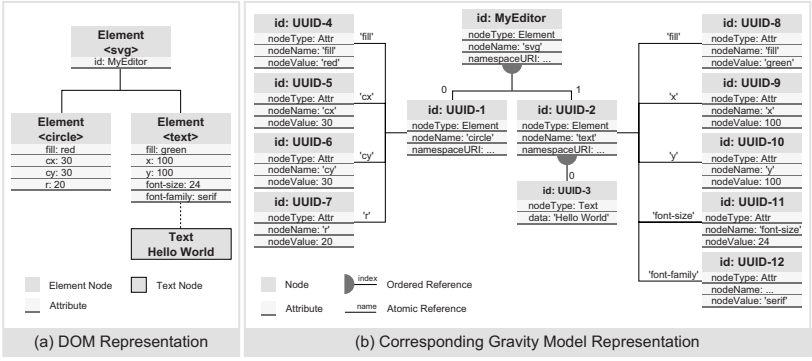


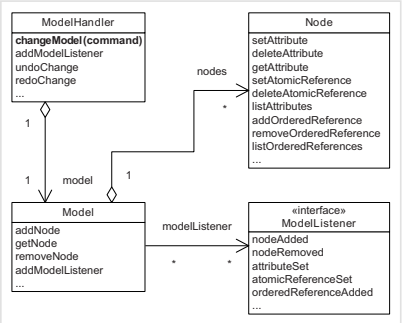
Figure 5.6: Mapping of the graphics editor DOM tree (cf. Figure 5.2b) to a Gravity model graph

the GM representation according to the established rules. In the center of Figure 5.6b are the counterparts of the four DOM nodes. The additional nodes (UUID-4 to UUID-12) had to be created to reflect DOM attributes.

5.2.5 Operational Transformation Engine

The introduced Operation Mapper component passes Gravity operations to the Operational Transformation Engine (OTE) that is in charge of (1) applying incoming operations to the Gravity model and (2) synchronizing all Gravity model instances. Note that instead of reimplementing an OTE, we reuse the existing SAP Gravity OTE (cf. Section 3.1.1). To grasp the complete capture and replay process, we include a concise Gravity OTE description in this section.

The first task of applying Gravity operations is crucial to finalize the sync of the DOM and the Gravity model. Figure 5.7a presents an excerpt of the Gravity API to modify Gravity models. The `ModelHandler` method `changeModel` is the exclusive entry point for all GM changes and the parameter `command` represents a function embodying GM changes. Figure 5.7b shows an example of a GM change. Thereby, all manipulating operations accommodated in the body of the anonymous function are considered as one complex operation that is either applied entirely or not at all. The latter case occurs if two operations are not compatible and the only way to resolve the conflict is a complete rollback of one of the operations (e.g. two users set the fill color of a rectangle to differing values).



(a) Gravity Model Change API

```
var change = modelHandler.changeModel(
   // manipulate the Gravity model
   function (model) {
     var rootNode = model.getNode("MyRootNode");
     var gravityNode = model.addNode(...);
     gravityNode.setAttribute(...);
   }
);

change.onAcknowledge(
   function () {
      alert("Changes accepted.");
   });

change.onCollision(
   function () {
      alert("Changes rejected due to a collision.");
   });
```

(b) Example of a Gravity Model Manipulation

Figure 5.7: Gravity API specification and usage example

Once the local GM is in sync with the local DOM, the Gravity operations are serialized into a JSON format [114] and propagated to the central Gravity server. For the client-server communication, Gravity leverages the CometD open source library [115] which allows bi-directional communication while hiding protocol specifics from developers. CometD autonomously

adapts to the supported bi-directional means at runtime and abstracts from the concrete implementation that uses the WebSocket protocol or HTTP polling or streaming techniques (cf. Section 2.2.3). After the operation transmission, the Gravity server deserializes the JSON message and incorporates the Gravity operation into its own internal GM adopting the OT algorithm (cf. Section 2.2.1). Concurrent operations received from other clients have to be transformed against the incoming operation to maintain GM consistency. Transformed Gravity operations are again converted into a JSON string and propagated to all clients except the sender client. Just like the Gravity server component, receiving clients deserialize Gravity operations and transform them against concurrent local operations to resolve editing conflicts and to maintain a consistent GM.

5.2.6 DOM Manipulator

To carry out the last step of the capture, propagate and replay process, the DOM Manipulator component plays a pivotal role. The DOM manipulator realizes the sync of the remote Gravity model and the remote DOM, which means that local DOM changes are finally incorporated in all associated remote DOMs.

Figure 5.8 shows the overall change replay process starting with an event fired by the Gravity notification mechanism that is caught by the listener belonging to the operation mapper component. Registering listeners on the GM is processed similarly to registering DOM listeners. The corresponding API is exhibited in Figure 5.7a and, in particular, represented by the `ModelListener` interface. According to the bijective mapping rules specified in Section 5.2.4, the operation mapper determines the proper DOM operation to represent the Gravity model change.

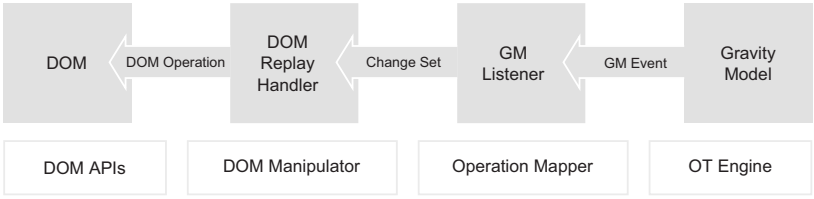


Figure 5.8: DOM change replay process

Applying the DOM operation is accomplished by the DOM manipulator leveraging the regular DOM API [40]. Functions like `createElement`, `createTextNode`, `setAttribute`, `removeAttibute`, etc., are contained in the DOM API and allow for arbitrary DOM modifications. Using the global identification scheme (cf. Section 5.2.2) enables the DOM manipulator to retrieve the proper DOM node and to apply changes at the correct location

in the DOM tree. Synchronizing the DOM with the Gravity model requires temporarily disabling registered DOM listeners since they would propagate DOM changes to the OTE even though this manipulation is already reflected in the Gravity model.

After the local DOM change is incorporated in the remote DOM, the end-to-end capture and replay workflow has been processed successfully, i.e. the process from recording DOM changes to distributing and transforming corresponding Gravity operations as well as replaying original DOM changes has been carried out completely.

5.3 Evaluation

In this section, we evaluate the collaboration capabilities provided by the introduced DOM adapter. Therefore, we devise an evaluation methodology capable of assessing software and collaboration qualities of transformed editors. Furthermore, we report on a conducted usability study encompassing two converted editors.

5.3.1 Evaluation Methodology

The specification of the DOM adapter architecture and the DOM adapter implementation in Section 5.2 demonstrated the technical feasibility of the proposed transformation approach. However, to validate that end-users are appropriately supported by the DOM adapter when working collaboratively requires a thorough evaluation.

An evaluation of a software system is "a significant check of a system's capacity to deliver what is required of it" [116]. For the proposed transformation approach, we claim that successfully transformed applications allow for shared editing and provide reasonable collaboration support.

Designing the evaluation methodology, we selected numerous quality aspects that should be assessed. A collaborative application, on the one hand, should decently support general software characteristics like functionality, usability or reliability. On the other hand, our evaluation should specifically target collaboration qualities like communication and coordination which are not sufficiently covered by general software metrics. Therefore, we compiled the quality criteria catalog from two established software metrics: the ISO/IEC 9126 standard for product quality [117] and the groupware-specific Mechanics of Collaboration (MoC) catalog [118].

ISO/IEC 9126 Software Quality Characteristics

The ISO/IEC 9126 [117] defines a quality model comprising six quality characteristics that are further subdivided into more detailed aspects in Figure 5.9. Depending on the type of usability study, some quality characteristics

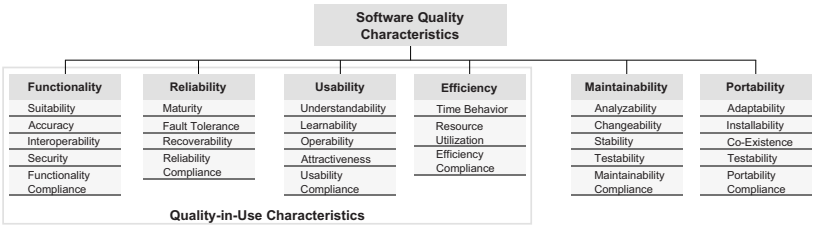


Figure 5.9: The ISO/IEC 9126 quality model [117]

might be out of the scope. While functionality, reliability, usability and efficiency are quality-in-use aspects that are of interest in end-users studies [119], maintainability and portability represent aspects that are relevant for developer studies. Since our evaluation targets end-users exclusively we only consider quality-in-use characteristics.

Mechanics of Collaboration Quality Characteristics

Besides the ISO/IEC 9126 quality characteristics, we also included the MoC catalog which defines "small-scale actions and interactions that group members must carry out in order to get a task done in a collaborative fashion" [118]. Hence, they are useful in groupware evaluations since "the mechanics are observable, collaboration can be analyzed and broken down into specific actions that evaluators can assess one at a time" [118]. The MoC characteristics are grouped into the communication categories *explicit communication* and *information gathering* as well as into the coordination categories *shared access* and *transfer*. While explicit communication defines to what extent collaborative applications are capable to supply sufficient communication support, information gathering specifies to what extent these applications are able to support their users retrieving sufficient information about other participants and their actions. In contrast, shared access represents a measure for the capability to support unconstrained access to shared document artifacts and tools. The transfer characteristic targets the support to exchange document artifacts and synchronize shared documents correctly.

- Explicit Communication: The extent to which collaborative applications are capable to supply sufficient communication support (e.g. spoken, written or gestural messages).

- Information Gathering: The extent to which collaborative applications are capable to support their users retrieving sufficient information about other participants and their actions (e.g. basic awareness

information answering questions like who is in the workspace or where are other participants working).

- Shared Access: The extent to which collaborative applications are capable to support unconstrained access to shared document artifacts and tools (e.g. obtain or reserve a resource).

- Transfer: The extent to which collaborative applications are capable to exchange document artifacts and synchronize shared documents correctly (e.g. handoff an object).

Since the DOM adapter does not provide any support for explicit communication, we excluded this aspect in the evaluation. However, explicit communication support is typically provided by external tools such as instant messengers, teleconference systems, etc. We also excluded the transfer characteristic from the evaluation since this aspect is already covered by the ISO/IEC 9126 functionality aspect. Consequently, we only considered the information gathering and the shared access categories in the evaluation.

5.3.2 Usability Study

After selecting the appropriate evaluation characteristics, we conducted a laboratory experiment with various converted editors. To cover numerous domains, we didn't solely focus on one editor but selected two representative single-user tools that were eligible to adopt the presented DOM adapter. First, we selected the graphics editor SVG-edit [36] that was introduced in the use case description in Section 2.1.1. In addition to the graphics editing domain, we also aimed to analyze the prevalent shared editing domain, namely, collaborative writing and thus, selected a text editor named CKEditor [35]. Both single-user editors were transformed into multi-user versions as specified in Section 5.2.1. Our demonstration page `http://vsr.informatik.tu-chemnitz.de/demo/GCI/` exposes several videos showcasing the shared editing capabilities of the collaborative versions of SVG-edit and CKEditor.

Study Procedure

In summary, thirty participants studying computer science or a related subject worked in teams of two to complete joint tasks. All fifteen teams had to finish a shared text editing assignment using the multi-user version of the CKEditor and a shared drawing assignment using the collaborative SVG-edit. For both assignments, the teams had to follow a fixed evaluation procedure consisting of (1) a 5 min editor tutorial, (2) a 15 min shared editing session and (3) a 10 min questionnaire completion phase. In the first phase of the evaluation procedure, participants had to get familiar with the

offered editor features. Therefore, each student had to complete ten small exercises on its own (cf. Appendix A.1). For example during the SVG-edit tutorial, participants had to create, fill, move and delete rectangle shapes. Detailed written instructions were given to all participants, so they could easily identify the suitable editor tool to accomplish the task.

Subsequently, participants were placed in one room equipped with two separated desks and two standard PCs running the collaborative applications (the converted CKEditor and the transformed SVG-edit). While participants could talk to each other, they were not able to see each other nor could they look at the other participant's screen since the office was partitioned by a room divider (cf. Figure 5.10). This setting tried to mimic a conventional setting where teams work geographically dispersed supported by an audio conferencing system. We applied this simplification to minimize the evaluation costs. Moreover, an evaluator was located behind the participants' desks to observe the entire evaluation procedure and to resolve potential setup issues. However, the evaluator did not actively interfere and influence the evaluation session.

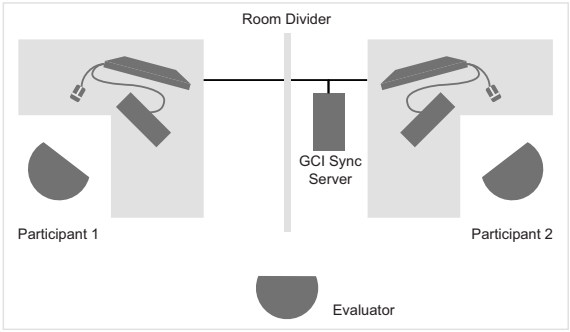


Figure 5.10: Physical usability study setup

Once the evaluation setup was prepared, the two team members had a time slot of 15 minutes at their disposal to complete one shared editing task. While authoring the shared editing tasks, we adhered to the following list of requirements:

1. Tasks should urge participants to work continuously and interactively.
2. Tasks should be independent of the subjects' knowledge, culture and skill set.
3. Tasks should reflect realistic assignments.
4. Tasks should not restrict participants in the way they use the collaborative application.

Eventually, we prepared two joint tasks (cf. Appendix A.2). The first exercise embraced the collaborative SVG-edit and participants had to jointly draw the floor plan of the evaluation office. This floor plan should be as detailed as possible including furniture, IT equipment and other inventory items. Moreover, the teams should enrich the floor plan with desirable items which could improve the work atmosphere (e.g. plants). The second exercise targeted the collaborative CKEditor which was used by team members to author a document containing an inventory list of the evaluation office. Besides the name of the inventory item, the document should also expose estimations about inventory characteristics like purchase price, maintenance costs, lifespan, etc. Furthermore, participants were asked to write a letter to the office manager listing inventory items that they suggest for purchase and a justification why these inventory items are beneficial for the work atmosphere.

In the last step of the evaluation procedure, participants had to complete a questionnaire in a 10 minutes time slot. The questionnaire comprised 22 questions (cf. Appendix A.3) that aimed to assess the software quality characteristics *functionality*, *reliability*, *usability* and *efficiency* as well as the collaboration characteristics *information gathering* and *shared access*. Respondents to the questionnaire could choose from a balanced seven-level Likert scale [120] exposing the options: (1) strongly disagree, (2) disagree, (3) disagree somewhat, (4) neither agree nor disagree, (5) agree somewhat, (6) agree and (7) strongly agree. In total, our laboratory experiment resulted in 60 completed questionnaires. While 30 questionnaires addressed the SVG-edit evaluation, the other 30 questionnaires targeted the evaluation of the collaborative CKEditor.

Study Results

The results reflecting the aggregated answers of all 30 participants are summarized in Figure 5.11 and 5.12 (cf. Appendix A.4). Both figures show the questions associated to the targeted quality characteristics. Moreover, the results of the SVG-edit and CKEditor evaluation are specified in form of the calculated mean μ and standard deviation σ. While grey bars represent the mean μ, black error bars symbolize the standard deviation σ.

In general, the results regarding the ISO/IEC 9126 characteristics (cf. Figure 5.9) allow for the following conclusions with respect to the transformed SVG-edit and CKEditor:

1. Overall, the transformed editors decently support collaborative work.
2. The specific collaboration features are easy to use.
3. Users are satisfied with the editors' performance.
4. Users struggle with reliability issues.

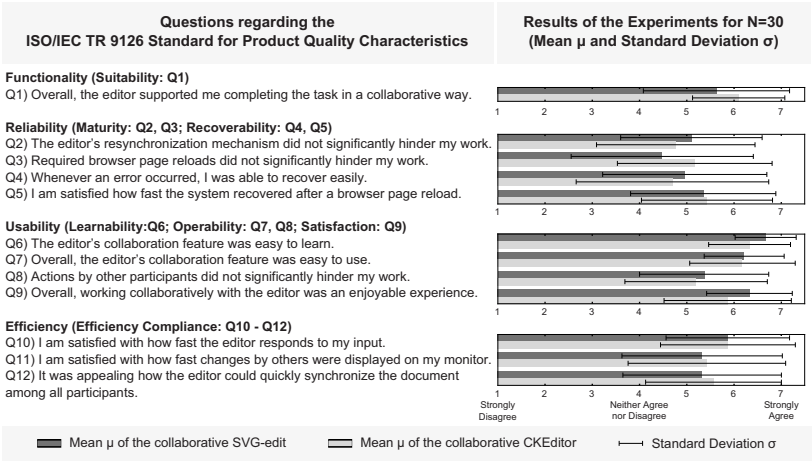


Figure 5.11: Questions and results regarding the ISO/IEC 9126 quality characteristics

Interpreting the individual ISO/IEC 9126 categories, we carved out the following findings:

Functionality ratings $\mu_{svg} = 5.63$, $\mu_{ck} = 6.10$ showed that users were comfortable with the provided collaboration support.

Reliability was assessed with ratings only slightly above the neutral rating *neither agree nor disagree* ($4.48 \leq \mu \leq 5.43$) representing the poorest assessment within the ISO/IEC 9126 categories. Those ratings are mainly due to rare browser crashes or situations where the browser remained unresponsive for some time. Both phenomena occurred only when high processor loads were observed[3].

Usability scores of $\mu \geq 5.20$ and $\sigma \leq 1.52$ show that the majority of participants was able to easily learn and adopt collaboration features. Furthermore, users felt that the collaborative work was an enjoyable experience (cf. Q9: $\mu_{svg} = 6.33$, $\mu_{ck} = 5.87$). However, answers addressing question Q8 delivered modest results ($\mu_{svg} = 5.37$, $\mu_{ck} = 5.20$) demonstrating that the work of remote participants occasionally impaired the local editing process. This is partially due to the single-threaded JavaScript programming model, where background tasks (e.g. transforming OT operations) can decrease the UI responsiveness.

Efficiency questions were constantly assessed with high ratings ($\mu \geq 5.33$). However, there is small gap between ratings for the local user feedback (cf. Q10) and the feedback resulting from remote operations (cf. Q11, Q12). We assume that users who communicated during the collaborative

[3]In Section 5.4.2, we explain the root cause for high processor loads.

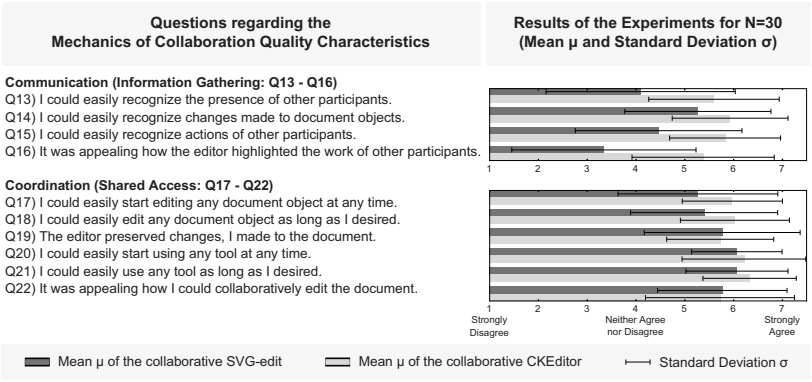


Figure 5.12: Questions and results regarding the mechanics of collaboration characteristics

assignment sometimes noted a small delay until the remote manipulation was incorporated and therefore expressed that they were not fully satisfied.

Besides the ISO/IEC 9126 quality aspects, the MoC criteria were also assessed for both collaborative editors. The results allow for the following general conclusions:

1. Primitive awareness support can crucially improve the perceived awareness of other users and their actions.
2. Both converted editors offer appropriate support for shared access.

A detailed analysis of the ratings in the communication and coordination categories reveals the following findings:

Information gathering ratings are ambivalent. On the one hand, the scores for the SVG-edit are poor ($3.35 \leq \mu \leq 5.27$) and, on the other hand, the assessment for the CKEditor is modest ($5.38 \leq \mu \leq 5.93$). One apparent reason for the divergence of the results is the differing workspace awareness support. While for the textual editor a participant list and creation coloring widget was provided, the graphics editor could solely offer a participant list. Note that the workspace awareness adapter (cf. Section 4.3.3) including prebuilt widgets (e.g. the telepointer, the telecaret, the radar view, etc.) could not yet be adopted since the implementation was still ongoing. In Chapter 7 we will present the workspace awareness adapter offering a variety of widgets and investigate the impact of workspace awareness in collaborative editing sessions in detail.

Shared access received consistently high ratings ($5.27 \leq \mu \leq 6.33$) which demonstrates that users appreciate the free and unconstrained access to document objects as well as to editor tools. Shared tool access rankings (cf. Q20 - Q22) are slightly better than the assessment of the shared artifact

access (cf. Q17 - Q19). One explanation is that certain remote operations occasionally impaired the ability to locally select an object. For example, if a remote user moves a graphical shape, the local selection in the SVG-edit did not always work properly.

5.4 Applicability

After elaborating on the technical feasibility of the DOM adapter and discussing the end-user satisfaction regarding converted editors, we will now examine the applicability of the approach. Thereby, we discuss the two main facets: applicability to web applications as well as limitations.

5.4.1 Applicability to Web Applications

To illustrate the applicability of the transformation approach, we analyzed the web application ecosystem and established a systematic eligibility check that is part of the overall transformation methodology. This eligibility check encompasses a criteria catalog containing properties that single-user web applications have to fulfill to adopt the DOM adapter. On the basis of the eligibility check, we assessed a variety of existing web applications to demonstrate the relevance of the devised transformation approach.

Eligibility Check Specification

Figure 5.13 embeds the two-step eligibility check into the overall transformation methodology that was introduced in Section 4.4. The eligibility check is split into a *necessary* criteria and a *critical* criteria check and both eligibility checks encompass a catalog of application properties. On the one hand, necessary application properties have to be fulfilled in order to allow for a single-user to multi-user transformation; on the other hand, the non-fulfillment of critical properties does not exclude applications from the set

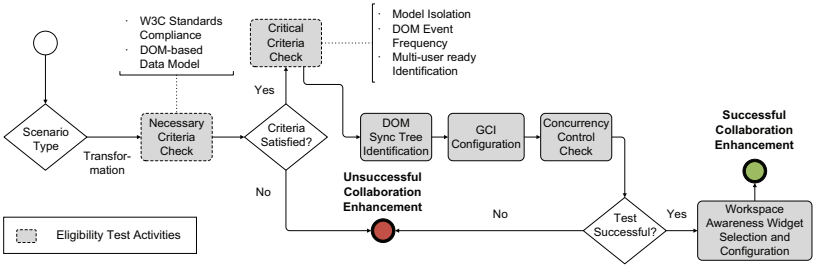


Figure 5.13: Web application eligibility check embedded in the overall transformation methodology

of convertible applications. However, these applications may suffer from an impaired GCI performance or functionality.

The list of necessary application properties comprises (1) the W3C standards compliance and (2) the DOM-based data model.

W3C standards compliance is required since the document change capturing and the document change replay is built on top of W3C standards (cf. Section 5.2). Conventional plugin technologies such as Adobe Flash [121] or Microsoft Silverlight [122] do neither comply with the DOM Events specification nor with the DOM Core standard. Hence, the record manipulations process (based on DOM Events) and the replay manipulations process (based on the DOM Core) is bypassed which disqualifies web applications leveraging plugin technologies.

A **DOM-based data model** represents the second necessary characteristic. Since the DOM is the only standardized representation all modern browsers can process in a uniform way, it is essential that the data model is accommodated in the DOM. If the data model is not encapsulated in the DOM, the DOM adapter synchronization mechanism breaks since standardized DOM APIs can no longer be exploited. Web applications structured according to the established Model-View-Controller (MVC) pattern are one example where the data model is not included in the DOM. In this case, only the view is represented in the DOM and the model is represented by a separate JavaScript data structure.

Besides the necessary application properties, there are also the critical application properties: (1) model isolation, (2) DOM event frequency and (3) multi-user ready identification.

Model isolation is a critical application property enabling the seamless sync of the data model. In contrast to the data model, view-related aspects like toolbar selections, window sizes, scrolling positions, etc., should not be synchronized since these characteristics are individual to each virtual workspace. However, web applications do not always strictly separate the data model and view-related aspects in distinct DOM subtrees which impairs the DOM adapter sync that operates on selected DOM subtrees (cf. Section 5.2.1). Hence, if the data model and view aspects are intermingled in the same DOM subtree, the sync incorrectly includes view elements.

The **DOM event frequency** is critical with respect to the DOM adapter performance. Currently, the DOM mutation event rate should not surpass multiple hundreds of events a second since this is the upper limit the latest GCI implementation is able to process. For example, these situations may arise carrying out drag-and-drop operations in graphics editors whereas the dragged object (e.g. a circle shape) changes its x and y coordinates hundreds of times a second. Another example are group operations where dozens or hundreds of DOM nodes are affected, e.g. a cut-and-paste operation involving numerous pages in a text document.

Multi-user ready identification is the last critical application property we identified. Web applications that were not meant to be used in multi-user scenarios may adopt a simple naming scheme for referencing document artifacts that breaks when linking various application instances. For example, the graphics editor SVG-edit [36] references created shapes using an incremented integer. If two users simultaneously construct a new shape in different workspaces, both shapes receive the very same integer ID leading to an incorrect application behavior.

Application Eligibility

Taking into account the necessary and critical application properties, we analyzed a set of 12 single-user web editors that are listed in Table 5.1. Thereby, we selected solely open-source applications that are widespread and also adopted by a large community. The selection process ensured that the applications cover a multitude of domains (e.g. text editing, source code editing, etc.). Conducting the analysis, we tested web applications with respect to the compiled criteria catalog. We excluded the DOM event frequency property since an adequate test would require excessive and time-consuming editor usage in a variety of scenarios. For example, peaks of DOM events can occur carrying out a specific operation such as a copy-and-paste operation affecting numerous DOM nodes. However, it is complex to find and test all of these scenarios. Furthermore, if one of the two necessary properties was not met, we omitted the test addressing critical application properties.

Ultimately, 6 editors (marked bold in Table 5.1) from the set of 12 satisfy the necessary application properties and hence are eligible for a GCI

Editor Name	Editing Domain	Standards Compliance	DOM-based Data Model	Model Isolation	Multi-User Readiness
ACE Editor	Source Code	Yes	No	-	-
Canvas Painter	Pixel Graphics	Yes	Yes	Yes	No
CKEditor	Rich Text	Yes	Yes	Yes	Yes
Eclipse Orion	Source Code	Yes	No	-	-
GelSheet	Spreadsheets	Yes	No	-	-
ImageBot	Pixel Graphics	Yes	Yes	No	Yes
jQuerySheet	Spreadsheets	Yes	Yes	Yes	No
Ketcher	Chemical Structures	Yes	No	-	-
Popcorn Maker	Video	Yes	No	-	-
SVG-edit	SVG Graphics	Yes	Yes	Yes	No
TinyMCE	Rich Text	Yes	Yes	Yes	Yes
Zwibbler	Pixel Graphics	Yes	No	-	-

Table 5.1: Results of the application eligibility test

transformation. Note that 50 percent of the analyzed applications expose an external data model which shows the large adoption of the MVC pattern in the web application ecosystem. The notion of structuring applications according to the MVC principle is promoted by numerous web application frameworks (e.g. Knockout [30] or Backbone [37]) that enforce applications to be divided into model, view and controller components. Nevertheless, the analysis shows that the DOM adapter is a viable option for numerous applications from multiple domains.

From the set of 6 eligible editors, we converted 4 applications (CKEditor [35], jQuerySheet [123], SVG-edit [36] and TinyMCE [42]) and adopted the resulting collaborative counterparts in real-life collaboration scenarios. For example, the multi-user versions of the CKEditor and the SVG-edit were exploited for the presented user study (cf. Section 5.3.2). In [113], we leveraged the collaborative TinyMCE editor and the converted jQuerySheet was adopted in an SAP-internal project.

5.4.2 Limitations

After demonstrating that the DOM adapter is a capable means to convert a variety of existing single-user editors, we will discuss the current limitations as well as approaches to overcome these limitations.

Plugin Technologies: As stated in Section 5.4.1, plugin technologies such as Adobe Flash [121] or Microsoft Silverlight [122] are not qualified to adopt the DOM adapter since they do not adhere to the four-layer web editor architecture depicted in Figure 5.1. Instead of leveraging the DOM as a representation of the application model, plugin-based web applications use the DOM only as a container to embed the plugin frame. User interactions within the plugin frame cannot be monitored by the DOM adapter because they bypass DOM APIs. Nevertheless, the importance of plugin-based solutions in terms of developer adoption decreases rapidly. That is due to (1) the rise of mobile platforms that do not support plugin technologies (e.g. Apple's mobile operating system iOS does neither support Flash nor Silverlight) and (2) due to emerging web standards (encompassing for example native audio and video playback [49] or device access to microphones or cameras [124]) that are supported by modern browser engines and render plugin technologies superfluous. Thus, future web applications will likely not be affected by this limitation.

JavaScript-based Data Model: Analyzing numerous web-based applications, we observed that editors targeting large documents (e.g. multi-page office documents) often separate the application model from the view model. This decision is based on an established design principle known as the Model-View-Controller pattern [111] that, in this particular case, can reduce the memory footprint significantly. Having a multi-page document comprising thousands of lines represented in a DOM can easily consume

more than 100 MB of RAM. In contrast, creating a specific application model using a tailored JavaScript implementation can efficiently compress large size documents. Thereby, the DOM is only used as a view model reflecting an excerpt of the application model. In this case, the DOM adapter is not able to access the model through the standardized DOM API and cannot be adopted. However, the framework adapter, which is discussed in Chapter 6, is complementary to the DOM adapter and specifically targets MVC-based web applications. Hence, even though web applications incorporating a JavaScript-based data model cannot leverage the DOM adapter, they may be eligible to exploit the framework adapter for the GCI (cf. Chapter 6).

High DOM Event Load: Certain editor operations affect numerous DOM nodes and produce vast sets of DOM events (cf. Section 5.4.1). Currently, all DOM mutation events are processed by the DOM adapter and are reflected in form of DOM replay operations. In rare cases, high operation loads can impair the performance and responsiveness of collaborative web applications. Examples for high load scenarios are (1) fade animations or drag shape operations (producing up to 150 `DOMAttrModified` events per second), (2) copy and paste operations involving numerous objects (triggering multiple `DOMNodeRemoved` and `DOMNodeInserted` events) and (3) formating operations affecting various DOM nodes (e.g. change the fill color of 100 table cells). In [125], Hauer addressed this issue incorporating an *operation composer* into the existing DOM adapter. The operation composer is in charge of reducing the number of operations that have to be synchronized. For instance, drag shape operations might change x and y attributes of a shape 150 times a second. These translation operations can be aggregated for short time intervals (e.g. 100 ms) without notably impairing the collaborative session. Hauer showed that the DOM operation load for high-load scenarios can be significantly reduced. For instance, the number of synchronized operations for a collaborative session using the SVG-edit application was reduced by more than 90 percent compared to leveraging the original DOM adapter. Thus, adopting an operation composer can eliminate high-load peaks and allows operating the DOM adapter within manageable load ranges.

Intention Violation: Besides maintaining document consistency, another objective of groupware systems is the *intention preservation* property [126]. Preserving the intention of all users' means that triggered operations are actually executed (e.g. a character is actually inserted into a text document after a user triggered the insert operation). However, this cannot always be guaranteed for any pair of editor operations. For example, if two users simultaneously resize the same rectangular shape the question is how can both intentions be preserved? Or what should be the combined effect if two users move a rectangular shape to different canvas positions? Those conflicts are typically resolved adopting the *single-operation effect* policy [127]

that preserves one operation entirely and dismisses the other operation completely. Even though SAP Gravity implements the single-operation effect policy, the current DOM adapter implementation may violate this policy. For example, if one user moves a shape from position $(0, 0)$ to $(10, 10)$ and another user concurrently moves the very same shape to position $(20, 20)$, the shape might be placed at position $(10, 10)$, $(20, 20)$, $(10, 20)$ or $(20, 10)$. Only if the shape is located at $(10, 10)$ or $(20, 20)$, the single-operation effect is properly implemented. However, the DOM adapter propagates two individual operations (x and y attribute changed) to the Gravity OTE instead of combining these two operations to one atomic operation which would ensure the single-effect policy. As described in Section 5.2.5, the Gravity `changeModel` function can construct complex operations from various primitive operations. In [125], Hauer presents a generic grouping mechanism leveraging the DOM adapter to properly combine semantically associated operations and thus, the single-operation effect can also be enforced using the generic DOM-based sync scheme.

5.5 Summary

In this chapter, we discussed the DOM adapter representing a generic means to transform existing single-user web applications into their collaborative counterparts. Thereby, we showed the technical feasibility of the DOM adapter presenting the high-level architecture as well as crucial implementation details. Furthermore, we conducted a thorough usability study exposing two converted editors to 30 end-users and demonstrated the suitability of the DOM adapter for collaborative work. Besides showcasing the feasibility and the end-user satisfaction, we also illustrated the viability of the transformation approach introducing 12 single-user web applications whereas 6 were eligible for the single-user to multi-user conversion. We concluded this chapter discussing existing limitations accompanied by proposals how these shortcomings can be overcome.

Chapter 6

Framework Adapter

Besides incorporating concurrency control capabilities into existing single-user web applications as an evolutionary step, a second approach is the from-scratch development. Thereby, shared editing requirements are already identified at the beginning of the development life cycle. The from-scratch development is supported by the framework adapter [128, 129] that is presented in this chapter. In line with Chapter 5, we divided this chapter into research questions, an architecture part, an evaluation as well as an applicability section. While the research questions highlight the scientific challenges, the framework adapter architecture sheds light on the individual components and their implementation. In the evaluation section, we expose information about a developer study comparing the framework adapter approach with a conventional programming library for collaborative web applications. We conclude this chapter discussing the applicability of the framework adapter as well as the framework adapter limitations.

6.1 Research Questions

In the framework adapter discussion, we will explore ways to integrate collaboration features into the from-scratch development of multi-user web applications. Due to the familiarity web developers gained with widespread web development frameworks such as Knockout [30], Backbone [37], etc., we will focus on enhancement approaches enriching existing web frameworks. Thereby, the following three research questions are of interest:

- **Research Question 1:** Can a generic collaboration infrastructure effectively enhance general-purpose web frameworks?
- **Research Question 2:** Do developers benefit from a framework adapter for a general-purpose web framework or do they prefer special-purpose programming libraries to introduce collaboration features?

- **Research Question 3:** To what extent can a framework adapter be adopted by existing web development frameworks?

Research Question 1 explores ways to anchor concurrency control in general-purpose web development frameworks to prove the feasibility of the framework adapter approach. Thereby, the investigated approaches should conceptually not be bound to a specific framework to allow reusing the framework adapter design for other web development frameworks as well.

Research Question 2 elaborates on the benefits and deficiencies of a framework adapter approach in contrast to a traditional programming library. The advantages and disadvantages are derived from a developer study encompassing eight programmers. Thereby, programmers build collaborative web applications using the framework adapter approach as well as using a conventional programming library. In order to mine meaningful results, we exploit completed questionnaires as well as produced source code.

Research Question 3 analyzes the web ecosystem in terms of popular web development frameworks and assesses to what extent these web development frameworks could profit from a framework adapter. Additionally, we also list and describe existing limitations that are induced by the framework adapter approach.

6.2 Framework Adapter Architecture

The architecture section presents design decisions for the framework adapter, elaborates on framework prerequisites to adopt the framework adapter and provides a conceptual view as well as implementation details regarding the framework adapter component.

6.2.1 Design Decisions

Framework and DOM adapter share the common goal to increase programmer productivity developing collaborative web applications while supporting differing development paths (cf. Section 4.4). In addition to the overarching goal of supporting programmer productivity, Section 2.3 defined functional requirements[1] (concurrency control, workspace awareness) and efficiency requirements (minimal invasiveness, encapsulation, learnability, reuse, universality). These requirements led to numerous design decisions that we will illustrate subsequently.

Concurrency Control Implications: The offered collaboration functionality should be mature and flexible in terms of concurrency control. While maturity is key to satisfy end-user needs (e.g. reliability), flexibility

[1] From the set of functional requirements, we only considered concurrency control since the workspace awareness support is promoted by the workspace awareness adapter (cf. Chapter 7).

is necessary to serve the plethora of web applications that can potentially be built on top of the enhanced general-purpose development framework. Due to its proven track record (e.g. powering SAP Process Flow [29]), we selected SAP Gravity as the operational transformation engine of choice. The decision to leverage SAP Gravity was also justified because of the flexible graph model that is capable of supporting arbitrary application models (e.g. array, tree or graph data structures).

Minimal Invasiveness and Learnability Implications: To support the developer adoption of a framework adapter, the approach should be easy to learn and require as few source code changes as possible[2]. If source code changes are inevitable, we decided to exploit *source code annotations* since they can easily be distinguished from the rest of the source code. Moreover, source code annotations represent a widely adopted, lightweight means to introduce additional application features. For example, Java frameworks like Hibernate or Spring provide annotations (e.g. @Table, @Column, etc.) to enhance applications with features like dependency injection, object persistence or container configuration. In essence, a minimal annotation vocabulary can ensure that developers easily grasp the mechanics to incorporate collaboration functionality.

Reuse and Encapsulation Implications: To promote reuse and encapsulation, component-based design represents an established software principle that we took into account and that led to a strict component-based architecture. Adhering to a component-based layout not only supports reuse and maintainability but also eases the bundling of multiple product variants. For example, software vendors might want to offer single-user as well as multi-user product variants. Thus, we clearly separated collaboration components from other functional components.

Universality Implications: The universality objective is two-fold. On the one hand, a universal framework adapter should not depend on a specific device or a specific browser platform which would strongly limit the reach of the approach. This is particularly true since today's web landscape is largely fragmented in terms of devices (e.g. smartphones, tablets, PCs) as well as in terms of browsers (e.g. Google Chrome, Internet Explorer, Mozilla Firefox). Hence, the architecture should leverage an existing web application framework providing a robust abstraction and hiding browser inconsistencies as well as device specifics. Besides browser and device independence, the approach should also be universal in terms of framework support, i.e. the conceptual architecture of the framework adapter should be framework-agnostic and adoptable by numerous frameworks.

[2]Note that in contrast to the transformation approach (cf. Chapter 5), the minimal invasiveness objective is not of such an importance for the framework adapter approach since it can be assumed that developers are familiar with the source code.

6.2.2 Framework Prerequisites

In contrast to the DOM adapter supporting the DOM API exclusively, the framework adapter approach aims to support various framework APIs. To implement the capture, propagate and replay processes, a minimal set of framework prerequisites has to be fulfilled. These framework prerequisites target the data model format and structure as well as the availability of a notification mechanism.

Data Model Isolation: A first prerequisite that web development frameworks have to fulfill is the separation of the presentation and the data layer. The existence of an isolated data model is crucial to synchronize numerous application instances in a device- and browser-independent manner since application models are a means to store data without including specifics regarding their presentation. Various web development frameworks enforce applications to be built according to the MVC pattern [111]. Since MVC applications natively expose a separated data model, web development frameworks enforcing the MVC paradigm comply with the data model isolation requirement. Examples for MVC-compliant web development frameworks are AngularJS [130], Backbone [37], Knockout [30], etc.

Notification Mechanism: A second framework prerequisite stems from the fact that model changes have to be captured and propagated to the collaboration engine. Implementing the change tracking behavior requires a so called notification mechanism. This notification mechanism offers publish-subscribe means to register listeners on data models and thus, the collaboration engine can be fed with model changes and ultimately maintain data model consistency. Notification mechanisms are widespread and most modern web frameworks, in particular MVC frameworks like AngularJS, Backbone or Knockout, provide means for notification.

Traversable Data Structure: Minimal invasiveness is a requirement for the framework adapter and thus, the number of source code changes should be kept to a minimum. Analyzing various frameworks, we identified two approaches to structure data models that allow minimizing source code changes. On the one hand, data models are *scattered* and divided into multiple partial models; on the other hand, there exist *coherent* data models exposing exactly one root node. Figure 6.1 depicts the structure of a scattered and a coherent data structure. To synchronize data model instances, the entire data structure has to be monitored, i.e. the collaboration engine has to be aware of all model changes. Thus, a model discovery mechanism has to be able to identify each model object which can be easily achieved for coherent and scattered data structures since they can be traversed from the root nodes. While coherent data models require exactly one entry point for traversal, scattered data structures necessitate one entry point for each (partial) model. However, both data model types allow discovering the data structure and thus are suited for injecting collaboration capabilities.

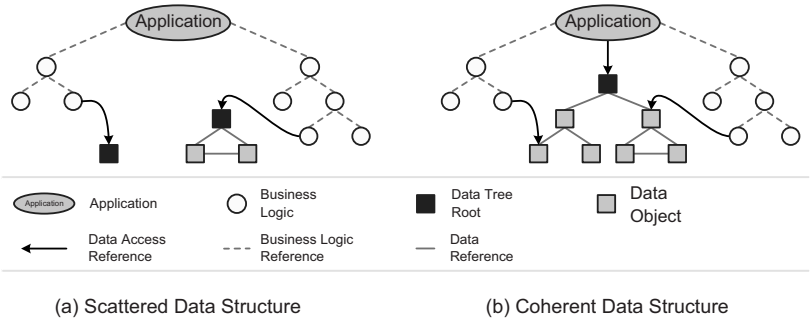


Figure 6.1: Classification of typical data model structures

6.2.3 Architecture Overview

The presented design decisions, framework prerequisites as well as the high-level GCI architecture (cf. Section 4.2.3) influenced the final architecture of the framework adapter that is depicted in Figure 6.2.

The distributed system derived from the GCI overall architecture (cf. Figure 4.6) accommodates numerous clients and a central server that communicate using the WebSocket protocol or HTTP. The architecture building blocks are materialized by single-user editor components, framework adapter components as well as concurrency control components.

The components belonging to the single-user editor include the application-specific components Editor UI and Editor API as well as the application-agnostic components Framework APIs and Data Model. Due to the universality requirement (cf. Section 2.3), the framework adapter components are linked to the application-agnostic framework API layer.

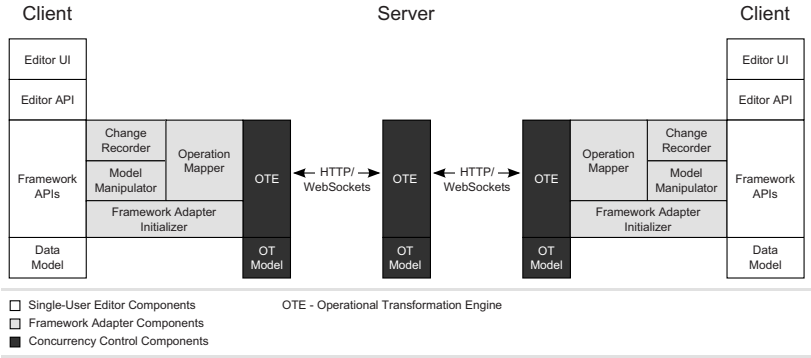


Figure 6.2: Framework adapter architecture

In line with the DOM adapter architecture, the framework adapter is divided into sub-components such as the Change Recorder, Model Manipulator and Operation Mapper. While framework adapter components are commonly bound to one specific web development framework, the Operation Mapper also encapsulates reusable functionality. For example, the JSON-to-OT Model mapping can be leveraged by various frameworks since it is common to structure data models according to the JSON standard [114].

The concurrency control components OT Engine and OT Model are in charge of synchronizing OT model instances and resolving editing conflicts. As stated in the design decisions section (cf. Section 6.2.1), we leverage the SAP Gravity OTE to implement concurrency control.

Due to the differing data structure types (scattered and coherent models) that should be supported by the abstract framework adapter architecture, we implemented two varying framework adapter flavors. On the one hand, for coherent data models we created a *wrapper approach* that enriches the original API with collaboration support. Instead of calling the framework API directly, applications call a wrapper API that invokes the original framework API and also triggers the model sync. On the other hand, we devised an *annotation-based approach* for scattered data models where annotating the data model allows injecting program logic to track and replay model changes. Both approaches are presented in the following sections.

6.2.4 API Wrapper Approach for Coherent Data Models

Taking into account the abstract framework adapter architecture in Figure 6.2, we derived the API wrapper approach to incorporate concurrency control functionality. Thereby, it is assumed that there exists exactly one fixed framework API to manipulate the application's data model. For instance, the SAPUI5 framework [38] or the SproutCore framework [110] expose such a central API that allows manipulating various web application data models in a uniform fashion. Thus, it is sufficient to replace this uniform data model API with a collaboration-enhanced version. Since web development frameworks offering exactly one fixed data manipulation API commonly also promote structuring data models in a coherent way, the API wrapper approach is specifically suited for applications that accommodate coherent data structures (cf. Figure 6.1b).

API Enhancement

A common way to adapt an API to the needs of a novel architecture is the wrapper pattern which is also known as the proxy design pattern [131]. Leveraging a wrapper allows preserving the existing API declaration as well as enriching the original functionality (e.g. interfering model manipulations or propagating change notifications). An example of a wrapped API is de-

picted in Figure 6.3. The interface `IDataModel` defines the API. Originally, the `DataModel` class exclusively implemented the defined methods which suited the use case of developing single-user web applications. To also support multi-user web applications, the class `SharedDataModel` was added. The `SharedDataModel` class does not only implement the `IDataModel` interface but also leverages the functionality of the `DataModel` class by calling its methods. The methods of the `SharedDataModel` wrap and enhance the `DataModel` functions. For example, the `addObject` method belonging to the `SharedDataModel` class calls the `addObject` method from the `DataModel` class and additionally triggers the sync process involving the collaboration engine (cf. Figure 6.3).

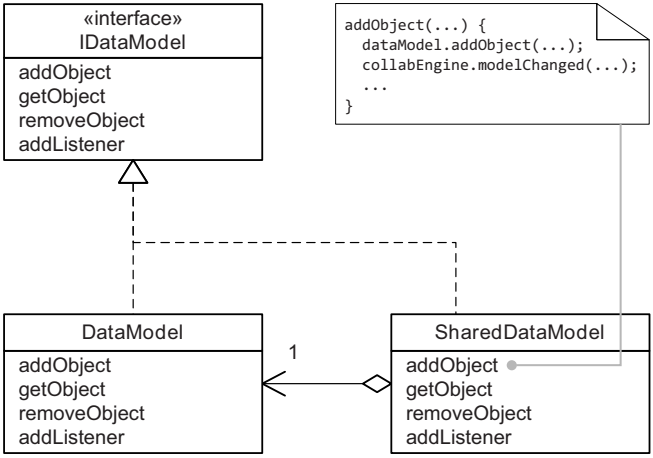


Figure 6.3: Example of wrapping an existing API

Applying the wrapper pattern is the key idea to convert APIs in a lightweight fashion while (1) promoting ease of use through a stable API and (2) allowing API reuse in novel contexts. To validate the wrapper concept, we have chosen to enhance the SAPUI5 framework [38]. SAPUI5 is a web development framework primarily used to build rich client-side applications or sophisticated UIs for client-server applications. Thereby, SAPUI5 offers a rich set of UI controls (e.g. `DataTable`, `DatePicker`, `MenuBar`, etc.) and a lightweight programming model embracing web standards such as HTML, JavaScript or CSS.

SAPUI5 provides a data model API that allows managing coherent data models in a uniform way and thus, it is qualified to adopt the wrapper approach. When building MVC applications, developers can choose from the following model representations: JSON [114], OData [132] or XML [133]. To bind the data model to the user interface, UI element constructors accept

a binding path expression pointing to a specific model element. Figure 6.4a shows the exemplary construction of the `jsonModel` that is bound to the text fields `tf1` and `tf2`, i.e. participant names are displayed in the corresponding text fields.

To support the development of multi-user web applications with SAPUI5, the main task was to wrap the `JSONModel` API in the `SharedJSONModel` class as depicted in Figure 6.4b[3]. Instead of instantiating the `JSONModel` class, programmers leverage the `SharedJSONModel` that provides the same interface but enhanced functionality. In the exemplary API transformation, we focused on the JSON data model but the scheme could also be applied to the OData or the XML data model provided by SAPUI5. In the next sections, we will discuss the adoption of a collaboration-enhanced API as well as the underlying processes such as the model mapping, change tracking and change replaying. To illustrate the concepts in a tangible manner, we will again adopt the SAPUI5 framework example.

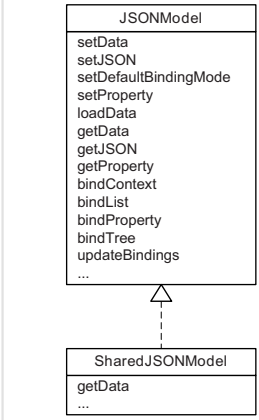
```
// setup the data model and initialize the JSON data
var jsonModel = new sap.ui.model.json.JSONModel();

jsonModel.setData(
{ "session": {
     "id": 100,
     "public": true,
     "participants": [
         { "name": "Alice" },
         { "name": "Bob" } ]
     }
});

// bind the JSON model to the UI
sap.ui.getCore().setModel(jsonModel);

var tf1 = new sap.ui.commons.TextField({
   value: "{/session/participants/0/name}"
});
var tf2 = new sap.ui.commons.TextField({
   value: "{/session/participants/1/name}"
});
```

(a)



(b)

Figure 6.4: a) SAPUI5 model creation and UI binding example b) API wrapper for the JSONModel class

API Wrapper Adoption

From the developer's point of view, the adoption of the wrapped framework API is straightforward since the API remains stable. Therefore, developers can leverage their existing framework API knowledge since the invocation of

[3]Note that the classes responsible for the data binding also had to be reimplemented since they would otherwise not inform UI elements about model changes entailed by replay operations.

object fields or object methods is preserved by the proxy pattern approach. In comparison to the single-user programming model, the only difference is the model instantiation. Instead of creating an instance of the single-user model class, programmers have to adopt the multi-user model constructor.

For the example in Figure 6.4a, the adoption of the wrapped SAPUI5 API would result in exactly one change. The constructor instantiation in line 2 needs to call the `SharedJSONModel` instead of calling the `JSONModel`. The remaining code does not require any changes which demonstrates the benefits of the proposed proxy pattern.

Data Model Mapping

An implementation of the wrapped framework API has to ensure that the application's data model and the generic OT model are in sync (cf. Figure 6.2). This requires (1) that all data model elements can be addressed in an unambiguous fashion and (2) that all elements of the application's data model can be expressed as OT model elements.

The unambiguous addressing scheme is necessary for selecting the corresponding model nodes to replay captured changes. Two established approaches are widely adopted to implement unique identifiers: (1) Universally Unique Identifiers (UUIDs) and (2) hierarchical identifiers. While UUIDs offer a compact string representation and are supported by various programming libraries (e.g. the Java Class Library or the .Net Framework Class Library), composite hierarchical identifiers are common in the web ecosystem (e.g. URIs [66], XPath expressions [134]).

In contrast to the DOM adapter implementation leveraging UUIDs (cf. Section 5.2.2), we adopted the hierarchical addressing scheme JSONPath. Like XPath allows querying XML documents, JSONPath similarly allows querying JSON documents. Figure 6.5a formalizes JSONPath in the Extended Backus-Naur Form (EBNF) [135] and examples of JSONPath expressions are depicted in Figure 6.5b. We opted for JSONPath because it can directly be derived from the JSON resource without having to construct and store an extra identifier.

```
jsonPath = "/" | {pathFragment};
pathFragment = "/", (index | string);
index = "0" | "1" | "2" | ...
string = concatenation of arbitrary
         characters except "/"
```

(a)

```
"/"                              -> Entire JSON document
"/session/id"                    -> ID of the session
"/session/participants"          -> Participant array
"/session/participants/0"        -> First participant object
"/session/participants/0/name"   -> Name of the first partipant
```

(b)

Figure 6.5: a) EBNF grammar defining JSONPath and b) JSONPath examples querying the JSON document depicted in Figure 6.4a

Besides establishing identifiers, being able to express application model elements in the OT model representation is the second requirement for a

successful model mapping. To illustrate the process, we consider again the SAPUI5 framework example and show how the JSON data model can be expressed as a Gravity data model. As described in Section 5.2.4, the Gravity model provides nodes with a unique identifier, attributes, atomic references and ordered references.

The JSON language [114] comprises *primitive data types* (string, number, boolean, null), *objects* as well as *arrays*. The following mapping rules are sufficient to map an arbitrary JSON model to a Gravity Model (GM) representation.

1. The JSONPath expression of a JSON element represents the ID of a GM node.

2. A primitive JSON value is represented by GM node with a `type` and a `value` attribute. While the `type` attribute holds the JSON data type, the `value` attribute stores the corresponding JSON value.

3. A JSON object is represented by a GM node and a `type` attribute set to "object". Key-value pairs belonging to the JSON object are mapped to atomic references where the key corresponds to the reference name.

4. A JSON array is represented by a GM node and a `type` attribute set to "array". Array values are mapped to an ordered reference where the array index corresponds with the index of the ordered reference.

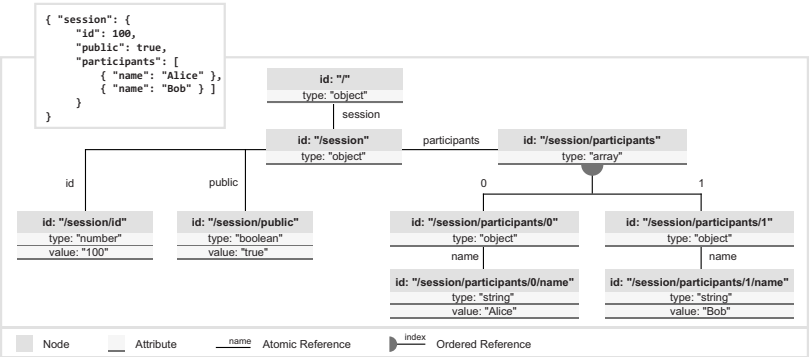


Figure 6.6: Exemplary mapping of a JSON model to a Gravity graph model

Figure 6.6 depicts exemplary mappings for primitive types (e.g. the node with the ID `/session/id`), for arrays (e.g. the node with the ID `/session/participants`) and for objects (e.g. the node with the ID `/session`).

Change Tracking

A second workflow, the framework adapter is in charge of, is the change tracking workflow that captures local model manipulations. To monitor local model changes, the following steps have to be executed.

1. Local Data Model Sync: Once a user interaction triggered a UI change, the data binding logic has to ensure that this change is reflected in the local data model. Since the binding of the UI and the data model is standard functionality provided by most web development frameworks, the API wrapper delegates this process step.

2. Local Shared Data Model Sync: After the local data model is in sync with the UI, the API wrapper has to map the recorded manipulation to the local shared data model. This encompasses to re-create local changes in the shared data model representation and to sync identifiers between the local and the shared model instance.

3. Remote Shared Data Model Sync: The local OTE has to propagate the change to all remote instances. Thus, remote OTE instances can incorporate the change, resolve editing conflicts and ultimately maintain document consistency.

For the concrete SAPUI5 example, we implemented the change tracking process using the SAP Gravity OTE. The process steps carried out by the SAP Gravity OTE are similar to the ones that are executed in the DOM adapter implementation described in Section 5.2.5. The major difference is the defined model mapping taking into account the JSON representation and converting it to the Gravity model format.

Change Replaying

The third workflow realized by the framework adapter is the change replay workflow. Change replaying includes the following process steps.

1. Local Data Model Sync: Once a client-side OTE has incorporated a received change operation into the shared data model, the application's native data model has to be synchronized. This comprises re-creating changes in the local data model and synchronizing model identifiers.

2. User Interface Sync: Finally, the replayed change has to be rendered on the UI of the application. The API wrapper implementation commonly delegates the sync to the underlying data binding API exposed by the web development framework.

For the interplay of the SAPUI5 framework wrapper and the SAP Gravity OTE, the only specific implementation aspect varying from other implementations is the model mapping. Inverting the previously established JSON to Gravity mapping rules produces the required set of mappings.

6.2.5 ColADA Approach for Scattered Data Models

While the presented wrapper approach is adoptable by frameworks exposing a uniform data model API and a coherent data model structure, we also aim to support scattered data models with application-specific model access. Thus, we propose the Collaborative Applications via Data Annotations (ColADA) approach [128] representing a second framework adapter materialization. In the ColADA approach, source code annotations are leveraged to mark partial data models for document synchronization. An annotation processor replaces these source code annotations with executable code to convert the local data structure into a shared data model instance that allows for collaborative editing.

The ColADA adapter, which materializes a second framework adapter, also allows bridging the gap between single-user application components and the OTE (cf. Figure 6.2). In the following sections, we specify the annotation language, the annotation incorporation process as well as the annotation replacement process. Moreover, we describe the synchronization workflows that are carried out by the runtime components of the framework adapter (e.g. the change recorder, the model manipulator, etc.).

To be able to validate the ColADA approach, we selected a web development framework that fulfills the framework prerequisites (cf. Section 6.2.2) and that has a substantial distribution in the web development ecosystem. Due to the massive adoption with more than 1 million downloads in 2012 alone [136], we chose the Knockout web development framework [30]. The Knockout framework allows building rich MVC web applications offering a notification mechanism, declarative bindings as well as template mechanics. The concrete framework adapter implementation for the Knockout framework will be used in the entire ColADA discussion to illustrate the concept in a tangible manner.

Annotation Language

Annotation vocabularies are popular for enhancing applications with additional runtime features. For instance, object persistence or container configurations are commonly implemented through annotation approaches (e.g. in the Hibernate or Spring framework). In our use case, the ColADA annotation language represents a means to configure synchronization processes. For the Knockout-specific implementation, the following annotations are required.

- @Sync: The `@Sync(modelName)` annotation marks the Knockout data model that should be synchronized among all application instances sharing the same session. The parameter `modelName` identifies the name of the JavaScript variable pointing to the data model.

- @Class: The `@Class(className)` annotation acts as a selector for the object constructor. In order to allow for a proper replay of a local object creation at all remote sites, an object constructor has to be leveraged since the object creation might incur side effects. For example, creating a new object might entail to increment a global counter. This side effect of incrementing a counter cannot be replayed in a generic fashion and thus, the collaboration engine requires a handle to the actual object constructor.

This compact set of annotations is sufficient for the annotation processor to inject the record and replay logic.

Annotation Incorporation

Besides getting familiar with the annotation vocabulary, developers have to accomplish a number of tasks to enrich web applications with ColADA collaboration support. For the Knockout framework, these development tasks include:

- Annotation Insertion: Insert source code annotations in all files encapsulating data model definitions.

- Annotation Processor Configuration: Complete a dedicated configuration by listing all files which contain annotations.

- Knockout Collaboration Adapter (KCA) Import: Adapt the view definition in order to replace the original model import with the `kca.js` import.

To show how source code annotations can be adopted to implement collaborative Knockout applications, we introduce the minimal example of a todo list application. This collaborative application should allow multiple users to concurrently add, remove or edit tasks that are organized in a list. Knockout applications commonly comprise two distinct parts: a view definition as well as a model definition which are automatically associated at runtime.

```
...

  <!-- <script type="text/javascript" src="model.js"/> -->
  <script type="text/javascript" src="kca.js"/>
...

  <input data-bind="value: input"/>
  <button data-bind="click: addTask">Add Task</button>
  <ul data-bind="foreach: tasks">
     <li>
         <span data-bind="text: name"></span>
         <a href='#' data-bind="click: delete">
            Delete Task</a>
     </li>
  </ul>
...
```

(a)

```
// @Class("Task")
var Task = function (data) {
          this.name = ko.observable(data.name) }

Task.prototype.delete = function() {
                        model.tasks.remove(this) }

// @Sync("model")
var model = { input: ko.observable(),
              tasks: ko.observableArray() }

model.addTask = function() {
       model.tasks.push(
         new Task({'name': model.input()}));
       model.input("") }

ko.applyBindings(model);
...
```

(b)

Figure 6.7: a) Exemplary Knockout view definition and b) the corresponding annotated Knockout data model

Figure 6.7 depicts the collaboration-enabled view as well as the model for the exemplary todo list application. The view definition (cf. Figure 6.7a) mainly comprises regular HTML tags intermingled with a Knockout-specific `data-bind` attribute. While HTML tags define the UI to enter new tasks and to enumerate them in a dedicated list, the `data-bind` attribute establishes the link to the data model. The only difference between the original and the collaboration-enabled view definition is the script import section. Instead of embedding the original Knockout model (encapsulated in the `<!--` / `-->` tags), the collaboration adapter `kca.js` has to be included. The `kca.js` script exploits a dedicated configuration file to retrieve associated model definitions. Finally, the parser encapsulated in the `kca.js` locates all annotations and replaces them with the synchronization logic.

In Figure 6.7b, an annotated model definition associated to the view definition in Figure 6.7a is depicted. The `@Class` annotation marks the object constructor to allow for the creation of new task objects and the `@Sync` annotation points to the `model` variable to get a handle to the actual data model. Note that all annotations are encapsulated in JavaScript comments since JavaScript does not offer a native annotation concept.

Annotation Processing

To grasp the mechanics of the Knockout-specific implementation, it is crucial to understand the annotation processor that replaces annotations with JavaScript source code at runtime. The annotation processing starts by parsing all model definition files specified in the configuration and thereby inserted annotations are identified. Those annotations are expanded to blocks of JavaScript source code which for the @Class annotation is straightforward. The logic replacing the @Class annotation expands to a function call storing a reference to the constructor method in a global map. In contrast to this simple substitution, the replacement of the @Sync annotation is chal-

```
function traverseModel(knockoutModel) {
    ... // list includes all nodes of the Knockout model
    return koNodeList;
}

foreach(koNode in traverseModel(knockoutModel)) {

    koNode.setUUID();

    if(koNode.isType(Primitive)) {

        // create Gravity node and set inital value
        gravityNode = gravityModel.addNode(...);

        // propagate local changes
        koNode.subscribe(function(newValue) {...});

        // subscribe to Gravity model changes
        gravityModel.addModelListener(...);
    }

    if(koNode.isType(Array)) {...}
}
```

I II III IV V

Figure 6.8: Skeleton of the JavaScript function replacing the @Sync annotation

lenging since the injected source code has to bridge the gap between the Knockout model and the OTE which enables to propagate and to replay local manipulations. Figure 6.8 depicts a skeleton of the inlined function replacing the @Sync annotation. Note that this pseudocode is bound to the specific SAP Gravity OTE.

To sync an application, the Knockout model has to be mapped to the Gravity data structure and vice versa. This bi-directional mapping is materialized by the functions depicted in Figure 6.8. Establishing the mapping is subdivided in (I) traversing the Knockout model, (II) assigning a unique ID to Knockout nodes, (III) creating Gravity counterparts for Knockout nodes, (IV) registering listeners on Knockout nodes to inform about local changes and (V) attaching listeners to Gravity nodes to replay remote changes. In comparison to inserting a one-line annotation, the complex inlined function supporting arbitrary Knockout models adds up to more than a thousand lines of JavaScript code. This complexity originates from the generic applicability of the function that supports: the traversal of all graph-structured Knockout models, the mapping of various Knockout node types, the callback registration for different model change operations, etc. Note that inserting the source code for a specific Knockout model would drastically reduce the code complexity but the five major code blocks (cf. Figure 6.8) are still required.

Synchronization Workflows

After all annotations were replaced with corresponding JavaScript functions, the synchronization workflows as depicted in Figure 6.9 are executed by the browser's JavaScript engine. The synchronization is divided into two pro-

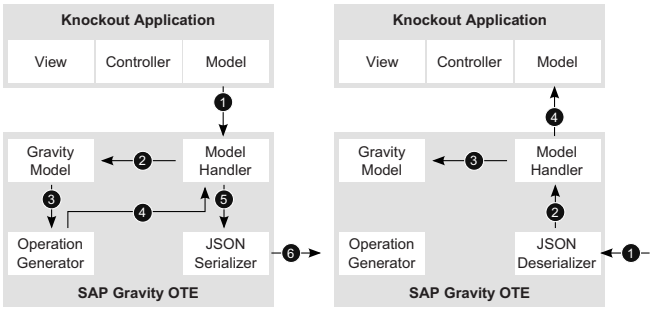


Figure 6.9: ColADA synchronization workflows for the Knockout prototype

cesses: the local change propagation and the remote change incorporation (cf. Figure 6.9).

The local change propagation workflow encompasses numerous steps. First, listeners registered on the Knockout model translate all kinds of model manipulations (e.g. change, create or delete operations) into Gravity API calls (cf. Section 3.1.1) and inform the model handler. As soon as the model handler is notified, the Gravity API calls are applied to the Gravity model. These changes to the Gravity model are observed by the operation generator which is in charge of extracting and grouping the resulting OT operations. Grouping OT operations is a specific Gravity OTE concept allowing to encapsulate multiple primitive OT operations in one complex OT operation that is executed in a transactional manner, i.e. complex operations are either completely executed or completely rolled back. For example, inserting a table in a word processor might comprise creating various table cells. The compound create-table operation can be easily translated to Gravity's complex operation concept. Aggregated OT operations are forwarded by the model handler to the JSON serializer. Ultimately, the JSON serializer converts OT operation objects into a JSON representation that is transmitted to the server.

The server distributes the JSON messages to all clients except the sender client. Clients receiving JSON change sets trigger the remote change incorporation process (cf. Figure 6.9b). Initially, the JSON deserializer transforms JSON messages into JavaScript objects accommodating OT operations. The model handler then transforms these operations against concurrent local operations and the resulting transformed operations are applied to the Gravity model. Thereby, the operation generator is detached from the Gravity model to avoid propagating the change back to remote clients. Finally, the model handler leverages the inlined code to reflect the changes in the Knockout model.

6.3 Evaluation

To assess development productivity, we conducted a developer study comparing a traditional concurrency control library with the framework adapter approach. Due to resource constraints, we selected one adapter implementation from the set of two (SAPUI5 collaboration adapter and Knockout collaboration adapter). We opted for the Knockout Collaboration Adapter (KCA) since the Knockout web development framework is available as open-source and much more proliferated. Hence, the results of the study are more valuable for the web development community.

Discussing the evaluation, we report on evaluation characteristics, the evaluation procedure, development tasks and evaluation results.

6.3.1 Evaluation Characteristics

In order to yield meaningful results conducting a developer study, we again adopted standardized software quality metrics. In particular, we took into account the product quality model and the quality in use model which are both defined in the ISO/IEC 25010 standard[4] [48]. While the product quality model determines the static quality of a software system, the quality in use model emphasizes characteristics that are relevant in concrete usage scenarios. Both models encompass quality aspects that are not appropriate for the assessment of development productivity. Figure 6.10 shows relevant characteristics that we embraced in the developer study as well as irrelevant characteristics that we chose to neglect.

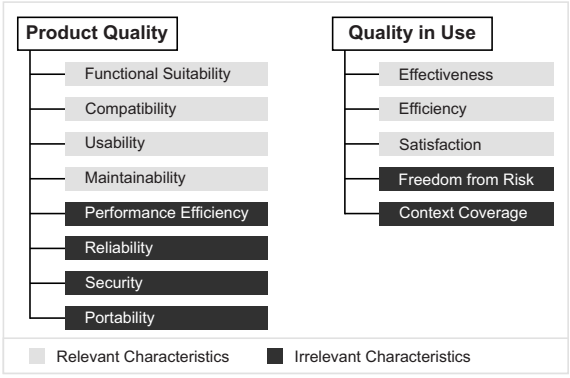


Figure 6.10: Product quality and quality in use model defined in the ISO/IEC 25010 [48]

[4]The ISO/IEC 25010 standard represents the successor of the ISO/IEC 9126 [117] that was adopted for the usability study in Section 5.3.2.

6.3.2 Evaluation Procedure

Before conducting the actual developer study, several aspects had to be planned in advance. For example, developers were recruited, introductory lectures about the ColADA approach and the selected concurrency control library were prepared, a suitable task description for the development of a collaborative application was devised and a questionnaire to assess the selected quality characteristics was authored.

To limit the evaluation costs, we offered a three months course at the Dresden University of Technology instead of recruiting professional developers. Students were only eligible for the course if they were enrolled at the faculty of computer science. Eventually, eight students participated. All students completed a questionnaire to assess their programming expertise. The completed questionnaires showed that all students were familiar with numerous programming languages (e.g. Java, C, etc.). However, no student was acquainted with the development of shared editing applications.

The offered course was divided into three development sprints. First, students were asked to develop a single-user web application using the Knockout framework. Afterwards, students had to enhance the single-user web application with collaboration capabilities. On the one hand, they had to use the devised ColADA solution and, on the other hand, they had to adopt the concurrency control library SAP Gravity. Each development sprint started with a specific lecture targeting (1) the Knockout framework, (2) the annotation-based programming model and (3) the SAP Gravity API.

To compare the traditional concurrency library SAP Gravity with the ColADA solution, a development task had to be designed. In this development study, we leveraged the introduced CoBAT use case (cf. Section 2.1.2). CoBAT should represent an efficient means to support collaborative cost-benefit analyses. A mockup of the envisioned CoBAT tool as well as the functional requirements are discussed in Section 2.1.2.

After receiving an introductory session about programming Knockout applications, the task specification (cf. Section 2.1.2) was distributed among all participants. In the first development sprint, students had to program the single-user application which served as the base application for the development of the collaborative CoBAT applications. In the second sprint, students programmed the collaborative CoBAT application adopting the Knockout collaboration adapter and in the third sprint, they were asked to introduce shared editing capabilities using the SAP Gravity API. Adopting the SAP Gravity API means students had to manually write the source code to synchronize the Knockout and the Gravity model which includes traversing the models, registering callback functions, etc. During the entire development period, students had to work and implement their prototypes autonomously. However, a weekly meeting was setup to discuss issues with other participants or with a dedicated supervisor.

To properly analyze the students' work, various qualitative and quantitative data sources were captured. In particular, the following data sources were used to compare the two approaches for collaborative application development:

- Development Documentation: In every development sprint, students were asked to complete a form. This form was divided into two parts: the time recording and the issues section. In the time recordings' part, students had to enter a subtask description associated to the time spent for the completion. In the issues section, students explained encountered problems.

- Source Code: The source code handed in at the end of each sprint was analyzed to assess the fulfillment of the functional requirements and to measure the resulting lines of code. The lines-of-code analysis divided the code contributions into the individual programming languages (e.g. JavaScript, HTML, etc.).

- Questionnaire: After completing the three development sprints, all students had to fill out a questionnaire comprising 34 questions. While 17 questions addressed the Gravity-based development, the other 17 questions aimed to assess the annotation-based development approach. The questionnaire (cf. Appendix B.1) depicted in Figure 6.13 was designed to evaluate product quality characteristics (cf. Q1 - Q12) and quality in use characteristics (cf. Q13 - Q17).

6.3.3 Evaluation Results

To compare the effectiveness and efficiency of the ColADA approach with a conventional concurrency control library, we first employed two quantitative measures: (1) the development time and (2) the lines of code measure. We used data collected in form of development documentation and in form of handed-in source code. Only if all functional requirements were fulfilled, the collected data was included in the effectiveness and efficiency assessment. From eight students seven were able to completely finish the three development tasks (the one single-user application and the two collaborative applications).

Consequently, when calculating the mean μ of the total development time, we only considered the timesheets from seven students. On average, students spent 54 hours to get familiar with the Gravity API and to program the Gravity-based collaborative application. In contrast, 42 hours were on average used to create a collaborative CoBAT application adopting source code annotations (cf. Figure 6.11). Hence, employing the annotation-based approach could reduce the development effort by 22 percent. The spent development times for the collaborative applications in Figure 6.11 include 25 hours that were dedicated for the implementation of the single-user application. Thus, the actual development effort for introducing shared editing capabilities adds up to 29 hours for the Gravity approach and 17 hours for the KCA approach. This represents a 41 percent reduction when adopting the annotation-based KCA approach.

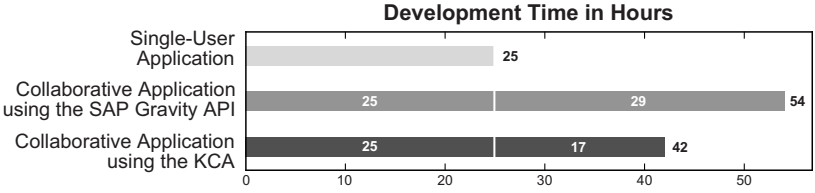


Figure 6.11: Calculated mean μ in the development time analysis

Even though the evaluation only encompassed eight developers and included solely one concurrency control library and one annotation-based approach, the trend is apparent that configuring a collaboration engine using source code annotations is beneficial in terms of efficiency and can significantly outperform conventional collaboration libraries. In terms of effectiveness, both development approaches are suitable to develop collaborative applications since the resulting implementations were able to fulfill all functional requirements.

The second quantitative measure analyzed the Lines of Code (LoC) metric. In the LoC analysis, seven valid source code contributions were included. The code contributions were divided into the individual categories (1) HTML code, (2) JavaScript code, (3) annotation code and (4) configuration code. Figure 6.12 shows the LoC measurements whereas in each category the mean μ is depicted. One distinguishing factor between the use of the Gravity API and the use of annotations is the JavaScript LoC measure. On average, developers needed 97 lines of JavaScript code accompanied by 4 annotations and 7 configuration lines to inject collaboration capabilities in contrast to 515 lines of JavaScript code for adopting the Gravity API. This represents a considerable reduction of 81 percent in terms of JavaScript code when leveraging the proposed annotation-based approach. Even though the HTML LoC exposes only minor differences, the overall LoC measure resulting in

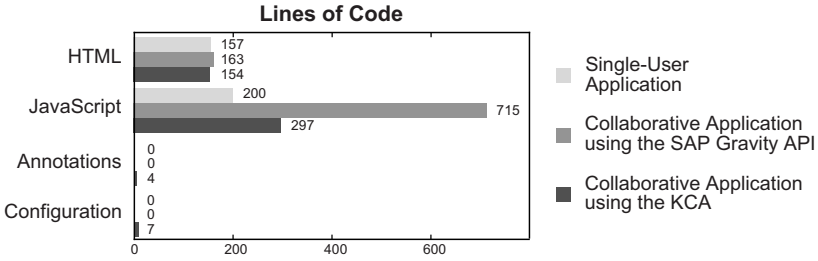


Figure 6.12: Calculated mean μ in the lines of code analysis

878 LoC versus 462 LoC once again shows a 47 percent source code reduction adopting the introduced ColADA solution. The substantial LoC reduction is another demonstration of the efficiency an annotation-based solution can deliver.

Besides employing quantitative measures, we also exploited qualitative evaluation techniques. Therefore, we created a questionnaire (cf. Figure 6.13) that targeted the selected evaluation characteristics (cf. Figure 6.10). For both development approaches, eight completed questionnaires were used to calculate the mean μ as well as the confidence interval [137]. While the mean μ is depicted in Figure 6.13 by grey bars, the confidence interval is visualized using black error bars. Confidence intervals are associated to a confidence level of 90 percent and the significance level of $\alpha = 0.1$. The confidence level expresses the likelihood that further equally conducted developer studies would also expose a mean within the limits of the confidence interval. Moreover, confidence intervals are a viable means to detect whether the difference of various mean constants (e.g. the means calculated for the two development approaches) are significant or not. If confidence intervals do not overlap, the differences of the means are significant [137]. If, on the contrary, confidence intervals do overlap, deriving an assured conclusion is not possible.

In general, the results of the experiment (cf. Appendix B.2) demonstrate that the ColADA approach constantly received superior ratings compared to the conventional approach leveraging the Gravity API. The functional suitability ratings could confirm that both approaches provide the necessary functionality to develop collaborative applications (Q1: $\mu_{KCA} = 4.50$, $\mu_{Gra} = 3.88$). However, the ColADA approach could significantly outperform the Gravity approach with respect to the ease of development (Q2: $\mu_{KCA} = 4.50$, $\mu_{Gra} = 3.25$). Regarding the compatibility characteristics (Q3 - Q4) developers stated that both programming methodologies did not restrict their choice of technology and that the technology interplay worked well. Even though the KCA ratings for compatibility characteristics

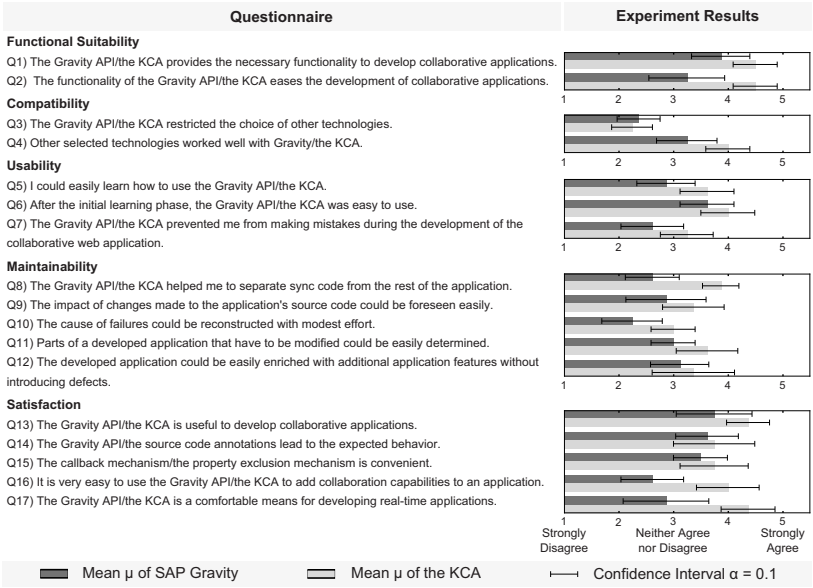


Figure 6.13: Developer study questionnaire and the corresponding evaluation results

are slightly better (Q3[5]: $\mu_{KCA} = 2.25$, $\mu_{Gra} = 2.38$, Q4: $\mu_{KCA} = 4.00$, $\mu_{Gra} = 3.25$), the difference is not substantial. The same trend with slightly better ratings for the KCA approach continued in the usability category where students were asked to assess the learnability, the ease of use and the error prevention. We would, however, have expected a larger difference, in particular in terms of learnability (Q5: $\mu_{KCA} = 3.63$, $\mu_{Gra} = 2.88$). Maintainability ratings once again exhibited considerable advantages for the KCA approach. In particular the separation of synchronization code from the rest of the application code is appropriately supported by the annotation-based approach (Q8: $\mu_{KCA} = 3.88$, $\mu_{Gra} = 2.63$). Nevertheless, the ability to detect and reconstruct failures leaves room for improvement. For the KCA approach error detection and debugging is especially cumbersome since annotations are replaced at runtime and consequently, the design time and the runtime definitions differ. The satisfaction category once again assured that the KCA is easily adoptable (Q16: $\mu_{KCA} = 4.00$, $\mu_{Gra} = 2.63$) and that the KCA represents a comfortable means to develop collaborative applications (Q17: $\mu_{KCA} = 4.38$, $\mu_{Gra} = 2.88$). The non-overlapping confidence intervals in Q16 and Q17 exhibit that this difference is significant.

[5]Note that in Q3 the principle the lower the mean μ the better the rating applies.

6.4 Applicability

In addition to describing the feasibility of the framework adapter approach and demonstrating the induced development efficiency, we also discuss the framework adapter applicability with respect to existing web development frameworks. Moreover, we exhibit identified framework adapter limitations.

6.4.1 Applicability to Web Development Frameworks

When designing the framework adapter, one objective was the universality of the approach aiming to support a large variety of existing web development frameworks. To analyze the universality requirement, we studied numerous popular web development frameworks.

Adopting the framework adapter approach requires applying the presented development methodology (cf. Section 4.4) that is refined for the from-scratch branch in Figure 6.14. First, a web development framework has to be selected and the criteria test has to be executed. If all criteria are fulfilled, it depends on the data structure (coherent or scattered) and the data model API (uniform or not uniform for all models) whether the wrapper approach or the ColADA approach can be leveraged. While Section 6.2 focused on describing the wrapper and the ColADA approach, we now present common web development frameworks that are eligible for the framework adapter approach.

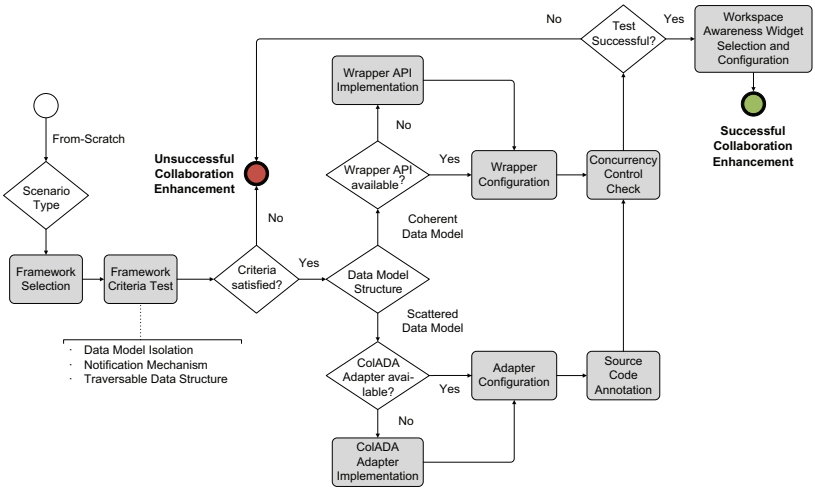


Figure 6.14: Refined from-scratch branch of the overall development methodology (cf. Section 4.4)

In this analysis of existing web development frameworks, we did not only focus on the defined framework perquisites (cf. Section 6.2.2) but also investigated additional framework properties providing valuable development insights. Having these insights at hand eases the task for developers to decide what framework is appropriate for their envisioned collaborative web application.

Hence, in addition to the identified framework prerequisites *data model isolation*, *notification mechanism* and *traversable data structure* (cf. Section 6.2.2), we included the following framework properties in our analysis.

- Software License: The source code license is of interest since it manages terms and conditions to use, to modify or to share the framework software artifact.

- Integration Process: The way the framework is integrated in the application development process impacts development efficiency. On the one hand, most frameworks only require including a JavaScript file; on the other hand, some frameworks require a preprocessing step.

- UI Implementation: For implementing user interfaces, frameworks commonly offer either widget-based libraries or templating mechanisms. Widgets represent sophisticated UI components that are instantiated with minimal effort and templating provides flexibility in a more verbose fashion.

- Programming Language: The programming language is a valuable characteristic indicating whether developers can leverage the native web scripting language JavaScript. In rare cases, frameworks require developers to adopt specific languages (e.g. CoffeeScript) necessitating additional tooling (e.g. an extra compiler).

- Popularity: To illustrate the popularity of a framework, we monitored forks and watchers on the predominant hosting platform GitHub [138]. While *forks* represent the number of projects copying the original framework's source code and starting an independent development, *watchers* are people that want to be notified about project changes.

A summary of the framework analysis is presented in Table 6.1. Even though we screened numerous web development frameworks, Table 6.1 only includes frameworks that are eligible to adopt the framework adapter approach. Thus, all listed frameworks fulfill the framework prerequisites data model isolation, the existence of a notification mechanism as well as the availability of a traversable data structure.

	Data Model Isolation	Notifcation Mechanism	Data Structure (C - Coherent, S - Scattered)	Software Licence	Integration Process (JI - JavaScript Include, PP - Preprocessor)	UI Implementation (T - Templating, W - Widgets)	Programming Language (JS - JavaScript, CS - CoffeeScript)	Popularity (GitHub Forks / GitHub Watchers)
AngularJS	✓	✓	S	MIT	JI	T	JS	126/919
Backbone	✓	✓	S	MIT	JI	T	JS	1070/7998
Batman.js	✓	✓	S	MIT	PP	T	CS	80/868
Cappuccino	✓	✓	S	LGPL	PP	W	-[1]	239/1655
Ember.js	✓	✓	S	MIT	JI	T	JS	304/2807
JavaScript MVC	✓	✓	S	MIT	JI	T	JS	72/405
Knockout	✓	✓	S	MIT	JI	T	JS	221/1873
SAPUI5	✓	✓	C	-[2]	JI	W	JS	-[3]
SproutCore	✓	✓	C	MIT	PP	W	JS	222/1739

[1]Cappuccino uses the Objective-J programming language.
[2]The SAPUI5 license is proprietary.
[3]SAPUI5 is not hosted on GitHub.

Table 6.1: Eligible web development frameworks for the framework adapter approach

In first place, the framework analysis shows that the framework adapter approach is relevant in terms of applicability. This conclusion is supported by the number of eligible frameworks but also by the GitHub statistics. Except the proprietary SAPUI5 framework, all frameworks have more than 70 forks and 400 watchers meaning the web community strongly embraces these widespread libraries.

A second derived conclusion is that the ColADA approach is more relevant in practice than the wrapper approach since supported data structures are mostly scattered ones. Solely SAPUI5 and SproutCore developers can leverage the presented wrapper approach.

Additionally, the selected frameworks are mainly distributed under the MIT license. Since the MIT license is a permissive free software license, developers are allowed to also use these frameworks in proprietary projects. Moreover, the majority of frameworks adheres to established web development best-practices, i.e. dependencies are commonly included via JavaScript imports and the preferred programming language is JavaScript.

6.4.2 Limitations

The framework adapter approach represents a viable solution for a variety of web development frameworks and a plethora of collaborative use cases. To conclude the applicability discussion, we present identified framework adapter limitations.

Impaired Debugging: Stepping through source code by means of a debugger is one effective way to identify the cause of an application issue (e.g. a thrown exception, peculiar input or output values, etc.). This approach to detect the cause of an error assumes that the design-time representation of the source code is equal to the runtime source code representation. However, for the ColADA approach, the annotation processor replaces annotations with blocks of JavaScript source code. When debugging the annotated application, developers are confronted with generated code and not with the familiar annotation representation. This representation switch[6] may impair the debugging efficiency since developers have to adapt to the injected source code.

Notification Bypassing: The notification mechanism is essential to capture data model changes and to trigger the replay process. However, skilled application developers may for whatever reason adapt or bypass the notification mechanism. Consequently, updates regarding model changes may get lost, i.e. the collaboration engine is not notified and the replay process breaks. One example is the Knockout notification mechanism. Knockout notifications are established through *observable* functions that allow monitoring model elements (cf. Figure 6.7b). Model properties are declared as observable leveraging the `observable()` function for simple properties, the `computed()` function for aggregated properties or the `observableArray()` function for arrays. If model elements are not declared as observable, the synchronization process will fail.

Disregarded Model Enhancements: The prototype-based JavaScript language allows enhancing model objects with arbitrary fields and functions at runtime. However, dynamically added model fields are not always automatically included in the corresponding record and replay processes. For example, the annotation processor belonging to the Knockout adapter locates and replaces annotations once the Knockout adapter script is loaded. If the model definition changes at runtime, the Knockout adapter will not take notice and will fail synchronizing this novel model property. Periodically examining all runtime objects for property enhancements could eliminate this limitation but at the cost of performance degradation.

[6]Note that the wrapper approach (cf. Section 6.2.4) is not affected by this limitation.

6.5 Summary

In this chapter, we discussed how established general-purpose web frameworks can be leveraged for the implementation of collaborative web applications. Thus, we introduced the framework adapter concept that is capable to inject collaboration functionality into existing frameworks and presented two exemplary implementations (the SAPUI5 and the Knockout framework enhancements). To show the benefits of the framework adapter approach, we conducted an extensive developer study comparing a collaboration-specific library with an enhanced general-purpose framework. The evaluation results demonstrated that the framework adapter is superior in terms of effort reduction and developer satisfaction. We concluded this chapter by identifying nine popular web development frameworks that are eligible to adopt the framework adapter approach which illustrates the broad applicability.

Chapter 7

Workspace Awareness Adapter

In addition to concurrency control implemented by the DOM and framework adapter, workspace awareness is the second functional requirement for collaborative web applications. This chapter therefore introduces the workspace awareness adapter [139, 140] that is in charge of non-invasively incorporating workspace awareness widgets (e.g. telepointer, participant list, etc.) into standards-based web applications. First, we discuss the emerging research questions regarding the feasibility, end-user satisfaction as well as the applicability of a workspace awareness adapter. Second, we expose the awareness adapter architecture and highlight relevant implementation details. Third, we present an end-user study evaluating software quality characteristics of the materialized workspace awareness adapter. And fourth, we assess the applicability of the awareness adapter in terms of awareness element coverage and web application coverage.

7.1 Research Questions

In line with previous chapters (cf. Chapter 5 and 6), we define research questions that we will discuss in the subsequent sections. In the context of the workspace awareness adapter, we identified the following research questions:

- **Research Question 1:** To what extent is a generic workspace awareness adapter technically feasible?
- **Research Question 2:** Do reusable workspace awareness widgets meet the end-user expectations in collaborative scenarios?

- **Research Question 3:** Is the coverage of the workspace awareness adapter in terms of awareness element support and applicability to web applications substantial?

Research Question 1 embraces the feasibility of the workspace awareness adapter approach which is challenging due to the universality requirement. Nowadays, the web application ecosystem is heterogeneous and differs especially in three dimensions: the application-, the browser- and the device-dimension. First, web applications vary in terms of their application domain (e.g. text editing, graphics editing, etc.). Second, there are numerous browser implementations for differing operating systems (e.g. Google Chrome, Internet Explorer, Mozilla Firefox, etc.). And third, web-enabled devices differ with respect to form factors (e.g. PCs, tablets or smartphones) and hardware (CPU, RAM, etc.). Thus, devising a generic workspace awareness adapter necessitates finding a suitable abstraction to serve applications in a manner that does not strongly depend on the application domain, the executing browser engine or the underlying hardware platform.

Research Question 2 targets the end-user acceptance of workspace awareness widgets that enhance collaborative web applications. Since the technical feasibility does not necessarily entail a compelling usability, we carry out another end-user study tailored for assessing workspace awareness qualities. On the one hand, this includes evaluating established metrics for software systems (e.g. usability, efficiency, reliability, etc.). On the other hand, characteristics like coordination and communication that are of utmost importance for shared editing applications should also be represented in the evaluation.

Research Question 3 analyzes the applicability of the workspace awareness adapter. Consequently, two facets have to be considered: (1) the workspace awareness coverage and (2) the limitations impairing the awareness adapter adoption. Workspace awareness coverage regards the distinct elements of workspace awareness (e.g. location, intention, reach, etc.) that were introduced in Section 2.2.2. Each awareness widget (e.g. participant list, radar view, etc.) covers the various awareness elements to a certain extent. Hence, a comprehensive and balanced awareness widget library has to ensure that the awareness elements are equally addressed. Secondly, this research question explores the adoption prerequisites and the limitations that may impair the usage of the awareness adapter.

7.2 Awareness Adapter Architecture

To address the feasibility research question, we set out to devise an awareness adapter architecture. In contrast to the presented DOM and framework adapter providing application-agnostic concurrency control, the awareness adapter analyzes the second functional GCI requirement which is the need

for workspace awareness (cf. Section 2.3). Besides offering comprehensive workspace awareness support, the awareness adapter should also target the efficiency requirements: minimal invasiveness, encapsulation, learnability, reuse and universality.

Subsequently, we will discuss the coarse-grained awareness adapter architecture, the individual modules accommodated in the awareness adapter (e.g. widget or event modules) as well as the integration with existing web applications.

7.2.1 Overview of the Awareness Adapter

Taking into account the requirements from Section 2.3, we constructed the awareness adapter architecture depicted in Figure 7.1, which is derived from the abstract GCI architecture shown in Figure 4.6. The distributed system consisting of a central server and an arbitrary number of clients comprises three types of components: (1) the single-user editor components, (2) the concurrency control components and (3) the workspace awareness components.

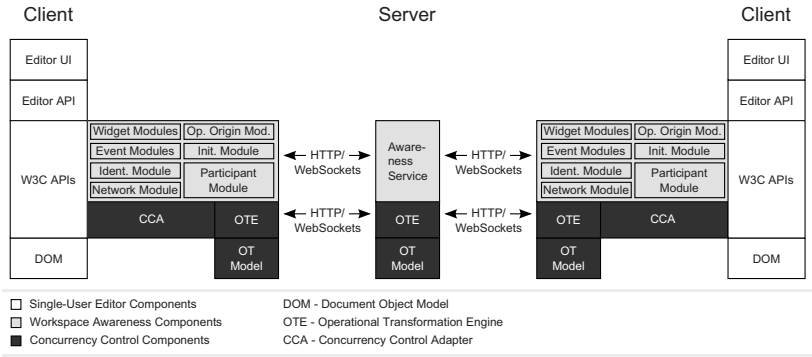


Figure 7.1: Architecture of the workspace awareness adapter

The well-known stack of single-user components encompasses the application-specific layers (Editor UI and Editor API) as well as the application-agnostic layers (W3C APIs and DOM). Again, the important architectural aspect is that the workspace awareness and concurrency control components exclusively leverage standardized W3C APIs instead of application-specific ones[1]. Linking web applications and collaboration infrastructure by means of a standardized layer is the essential aspect to allow for a reusable and application-agnostic GCI.

[1]Note that the concurrency control adapter can also leverage framework APIs (cf. Chapter 6 discussing the framework adapter) which is neglected in Figure 7.1.

For the sake of completeness and due to the fact that workspace awareness features only can unleash their potential in combination with real-time synchronization, Figure 7.1 includes concurrency control components. The concurrency control adapter can for instance be materialized by the DOM adapter (cf. Chapter 5) or by the framework adapter (cf. Chapter 6). However, since the awareness adapter is well encapsulated and external dependencies are linked via predefined interfaces, other concurrency control providers can also profit from the provided awareness support.

In this architecture discussion, we will focus on the individual workspace awareness components. Awareness widgets have to constantly receive updates from all clients in order to instantly inform participants about actions and activities of others in the shared space. Therefore, relevant changes in the shared workspace have to be captured, propagated and processed by the corresponding awareness widget. For example, the mouse cursor position has to be tracked to update the telepointer widget or newly entered text has to be recorded to feed the creation coloring widget.

To implement the capture, propagate and process awareness information workflow, the awareness adapter includes seven distinct modules. While Figure 7.1 integrates these modules into the global architecture, Figure 7.2 shows the modules and exposed interfaces as well as the grouping into *main* and *support* modules. The central initialization module is in charge of configuring and initializing the workspace awareness adapter. While event modules listen and notify about workspace modifications, widget modules leverage this input to update the awareness widget UI. Like the initialization module, the remaining ones (identification, participant, network and operation origin module) represent support modules. The identification module ensures workspace elements are addressable in an unambiguous way; the participant module handles leaving or joining participants; the network module

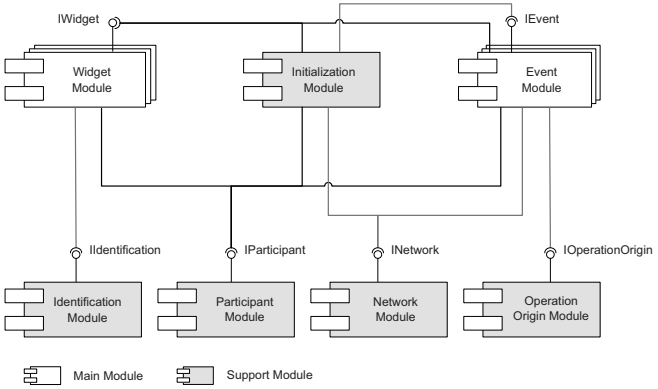


Figure 7.2: UML component diagram representing awareness modules

abstracts from the actual transport layer and the operation origin module distinguishes user-created operations from operations that are generated by the concurrency control provider.

To grasp the mechanics of the awareness adapter, we continue discussing the technical details of the introduced awareness modules.

7.2.2 Event Modules

To keep the view of an awareness widget up-to-date, widgets continuously require input (e.g. updates of mouse cursor coordinates for the telepointer). Offering information about changes in the shared workspace is the task of the various event modules. In essence, each event module takes care of (1) registering to DOM event sources, (2) deriving higher-level awareness events and (3) distributing awareness events.

Registration to DOM Event Sources: Standards-based web applications that do not leverage plugin-technologies (e.g. Adobe Flash [121], Microsoft Silverlight [122], etc.) materialize their user interface in form of the document object model. Thus, manipulations and interactions regarding the DOM can be monitored exploiting a vast set of standardized events that are defined in the DOM events specification [106]. Figure 7.3 lists event groups and selected event examples that are defined in the DOM events specification. Various events may be used to trigger awareness widget updates. For instance, there are UI events like `resize` or `scroll` affecting the radar view; mouse events like `mousemove` triggering telepointer updates; keyboard events such as `keydown` changing the telecaret position or mutation events like `DOMNodeInserted` inducing updates for the creation coloring widget. All these events can be captured adding event handler functions via the `addEventListener(DOMString type, EventListener listener)` method (cf. Figure 5.4).

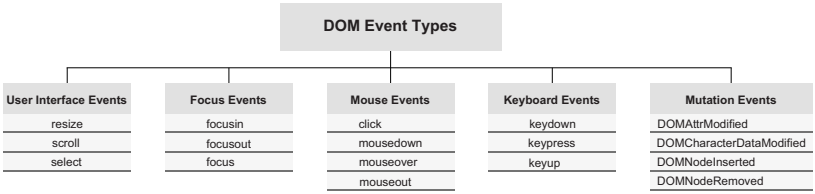


Figure 7.3: Examples of standardized DOM event types [106]

Deriving Awareness Events: Handling the multitude of low-level DOM events creates a lot of overhead in terms of code duplication since numerous DOM events may provoke the same awareness widget change. For instance, selecting a word in a text editor can be accomplished using the mouse device or the keyboard device. Even though the result in the artifact

marking widget is the very same, the thrown events differ depending on the used device. Therefore, we established a higher-level event abstraction called awareness events. Currently, awareness events encompass the events depicted in Figure 7.4 whereas these event types are generated from DOM events according to Table 7.1a. Besides considering DOM events, other event providers can also be included in the set of awareness events. For example, the collaboration engine (e.g. SAP Gravity [29]) can act as the participant event provider notifying about joining or leaving participants.

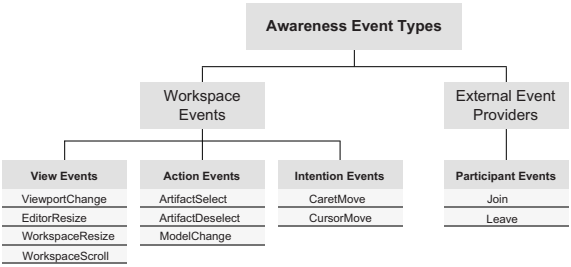


Figure 7.4: Defined awareness event types

Distributing Awareness Objects: Once awareness event objects are assembled, all web clients have to have access to awareness updates. The task of distributing awareness objects is executed by the network module that is discussed in Section 7.2.4.

To illustrate the event generation process, we utilize the example shown in Figure 7.5. A user of a collaborative editor pastes the word *World* into the shared document which results in the new value *Hello World* of the corresponding text node. Thereby, a `DOMCharacterDataModified` event is thrown encapsulating values like the identifier of the changed text node, the old value, the new value, etc. This mutation event is caught by the `ModelChange` event module which converts the mutation event to an awareness event. Besides copying values (e.g. the text node identifier), the event handler also computes additional values. For instance, the difference of the old and new text node value is calculated since awareness widgets only consume and visualize change sets. Moreover, a `Range` object [141] is deter-

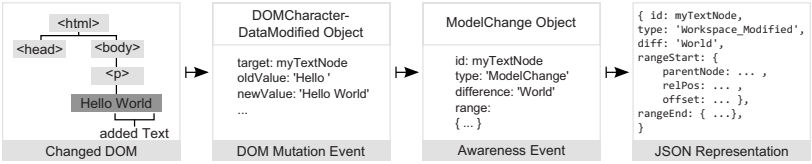


Figure 7.5: Example of the event generation process

DOM Event Type	Created Awareness Event Types
resize	EditorResize, WorkspaceResize
scroll	WorkspaceScroll
focusin	ArtifactSelect
focusout	ArtifactDeselect
mousedown	CaretMove
mouseup	CaretMove
click	CaretMove, ArtifactSelect
mouseover	CursorMove
mouseout	CursorMove
keydown	CaretMove, ModelChange
keypress	ModelChange
keyup	CaretMove, ModelChange
DOMNodeInserted	ModelChange
DOMNodeRemoved	ModelChange
DOMAttrModified	ModelChange
DOMCharacterDataModified	ModelChange

(a)

Awareness Widget	Used Awareness Event Types
Participant List	Join, Leave
Creation Coloring	Leave, WorkspaceResize, WorkspaceScroll, ModelChange
Telepointer	Join, Leave, CursorMove, EditorResize, WorkspaceResize, WorkspaceScroll
Radar View	Leave, WorkspaceModified
Telecaret	Leave, CaretMove, WorkspaceResize, WorkspaceScroll, ModelChange
Artifact Marking	Leave, ArtifactSelect, ArtifactDeselect, WorkspaceResize, WorkspaceScroll, ModelChange

(b)

Table 7.1: a) DOM event to awareness event mapping and b) Awareness widget to awareness event mapping

mined representing a minimal relative rectangle covering the newly created text *World*. Such a visual range is, for instance, of value for a creation coloring widget that highlights newly created text. Once the properties of the `ModelChange` event object are set, the network module serializes the object into a JSON representation that can be propagated to all clients.

7.2.3 Widget Modules

Besides capturing events, the awareness adapter has to notify about workspace changes and user interactions caused by other participants. Therefore, the awareness adapter can install a number of widget modules. Each widget module is in charge of the event processing workflow.

Event Processing Workflow

Carrying out the event processing workflow consists of (1) filtering, (2) processing and (3) visualizing awareness event objects.

Filtering Awareness Events: Awareness events as shown in Figure 7.4 can inform about a variety of manipulations and interactions in the shared space. Each widget is exclusively interested in a limited number of specific input sources (e.g. the participant list consumes the `Join` awareness event but neglects the `EditorResize` event). Thus, a publish-subscribe mechanism is offered to link widget modules to specific awareness events. Table 7.1b shows the distinct awareness widgets and the consumed awareness event types.

Processing Awareness Event Objects: Once an awareness event object is routed to a widget module, the awareness processing step is carried out by the widget module. The main task is to set up the awareness canvas to isolate awareness elements from the data model. This isolation is required because otherwise the model synchronization mechanism would also synchronize awareness objects. In contrast to model artifacts, awareness objects are individual for each user and should therefore not be synchronized. For example, the radar view depends on the local scrolling position which differs for each user and consequently, radar views should not be synchronized.

Visualizing Awareness Event Objects: After the awareness canvas is established, the widget module calls an update method to repaint the widget UI. Hence, end-users are aware of the current situation which promotes efficient collaborative work.

To illustrate the filter, process and visualize workflow, we revisit the example from Figure 7.5 where pasting the text *World* resulted in a `ModelChange` event. After the transmission and deserialization, the awareness event is dispatched to the creation coloring widget. As shown in Figure 7.6, a `<div>` element is created representing the awareness canvas. The specific `<div>` is the subtree root for all awareness elements that are added to the DOM. This awareness subtree is disjoint from the sync subtree spanned by the sync root node (cf. Figure 7.6). Finally, the update call from the creation coloring widget establishes a minimal transparent `<div>` highlighting the text *World* in the rendered web page.

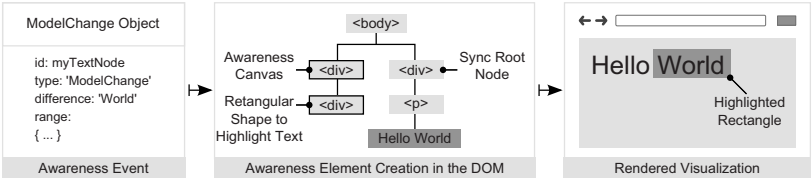


Figure 7.6: Example of the awareness event processing workflow

The Widget Interface

Even though the awareness adapter comes with a comprehensive awareness widget library encompassing a telepointer, telecaret, radar view, etc., certain applications may require augmenting the shared workspace with special-purpose awareness widgets (e.g. a multi-user scrollbar). Thus, the awareness adapter provides an extension mechanism materialized by the `IWidget` interface. All widgets that are managed by the awareness adapter have to implement the `IWidget` interface which is depicted in Figure 7.7. The following methods have to be implemented by an `IWidget` implementation:

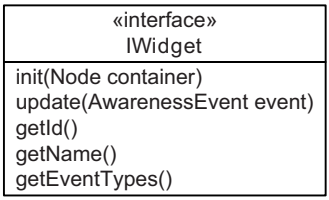


Figure 7.7: The IWidget interface

- `init(Node container)`: The initialization module (cf. Figure 7.2) calls this function to set everything up for the widget's operation mode. This includes drawing an initial widget view on the passed-in container node.

- `update(AwarenessEvent event)`: The `update` function is called to repaint the widget view taking into account the provided awareness event.

- `getId()`: To allow other awareness adapter modules to reference a specific widget, the `getId` function provides a unique identifier that is linked to the specific widget instance.

- `getName()`: To establish an expressive human-readable name for UI dialogs or headings, the `getName` function is offered.

- `getEventTypes()`: The `getEventTypes` function returns an array of relevant awareness events triggering widget updates. This list of awareness events is used to establish the publish-subscribe mechanism that informs about workspace changes and user interactions.

Widget Library

In addition to external widget providers, the presented `IWidget` interface is also implemented by widgets that are contained in the widget library. As part of the awareness adapter, the widget library provides reusable awareness widgets that have to be configured for each individual application. Figure 7.8 shows the implemented widgets: a telepointer, a radar view, a participant list, an artifact marking widget, a telecaret and a creation coloring widget.

Participant List: The participant list widget establishes awareness about who is currently in the shared workspace by listing all collaborators. Each participant is represented by a profile name and picture as well as a dedicated color. Assigning a color to each profile is an important aspect that is exploited by other awareness widgets. For example, the telepointer

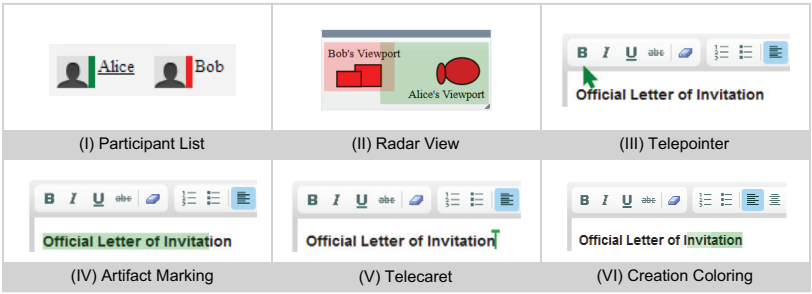


Figure 7.8: Awareness widgets encompassed in the widget library of the awareness adapter

or the telecaret embodiments are colored according to the color code established by the participant list widget. From an implementation point of view, the participant list is constructed using ordinary `<div>`, `<span>` and `<img>` elements that are added as child nodes to the `<div>` that represents the awareness canvas.

Radar View: To locate other participants in the shared space, the radar view shows semi-transparent end-user viewports. In addition to the color-coded viewports, constructed document artifacts are also depicted miniaturized. Scaling HTML documents is not trivial since HTML consists of a variety of differing media objects that cannot be miniaturized in a uniform manner (e.g. scaling fonts differs from scaling images). Hence, we used the html2canvas JavaScript library [142] to generate a pixel-based representation from the DOM view that can in turn be uniformly scaled to fit the radar view.

Telepointer: To be aware of the mouse cursor of other participants, the local cursor movements are emulated by a telepointer remotely. The path of the telepointer movement typically has to be adapted to the varying workspace settings (e.g. zoom level, window size or window resolution). Instead of leveraging x and y window coordinates, we use the underlying DOM node (e.g. a text node representing a heading or an SVG node visualizing a circle) as the reference point for positioning calculations. This results in more accurate positions in heterogeneous environments. To circumvent performance issues induced by SVG draw operations, the telepointer implementation uses a pixel-based HTML5 canvas layer to visualize the moving telepointer.

Artifact Marking: Local artifact selections (e.g. text selections) are highlighted remotely by the artifact marking widget that constructs a semi-transparent colored overlay. Again, for the implementation, absolute coordinates are not an option due to heterogeneous window sizes, scrolling positions, etc. Therefore, the HTML5 Editing API [141] provides a so called

`Range` object that allows specifying continuous selection parts based on content rather than on coordinates. These range objects are used to draw a properly dimensioned, semi-transparent `<div>` node on top of selected document artifacts.

Telecaret: Communicating the local text cursor position to teammates is the task of the telecaret which is achieved by drawing a blinking cursor. Corresponding to the overlay calculations of the artifact marking widget, the range object is also leveraged to compute the telecaret position. In case of the telecaret, the range object does represent a position and not a continuous selection. Positions are expressed through range objects by setting the `start` and `end` property to the same value. Finally, the visualization is realized through an extra `<div>` element that is added to the awareness canvas.

Creation Coloring: To establish awareness about newly created content (e.g. text segments or graphical shapes), the creation coloring widget highlights recently inserted artifacts. Commonly, the overlay highlighting new content fades out after a fixed time interval to avoid user interface clutter. Our implementation of the creation coloring widget again relies on the HTML5 Editing API [141] that allows calculating bounding boxes for the colored overlay. The `<div>` elements materializing the overlay are also added to the awareness canvas.

7.2.4 Support Modules

In addition to the described *main* modules, there are a number of *support* modules providing essential services to carry out the capture and visualize awareness event processes. The set of support modules comprises the identification, the participant, the network, the operation origin as well as the initialization module. These support modules will be discussed briefly in the following paragraphs.

Identification Module

The identification module represents one support module that is leveraged by various widget and event modules. Its primary task is to ensure that DOM nodes are addressable in an unambiguous manner, i.e. each DOM node is associated to a unique identifier. Such an identification service is required for all position and range calculations that reference the underlying DOM node. For instance, when a mouse pointer hovers over a toolbar icon materialized as `<img>` element, this `<img>` element is leveraged as the reference point for calculating the relative mouse cursor position. In order to position the remote telepointer accurately, the `<img>` element has to be determined remotely which necessitates a global unique identifier.

The algorithm to add identifiers to DOM nodes is conceptually equal to the global identification scheme leveraged by the DOM adapter (cf. Section

5.2.2). Relevant DOM nodes such as element nodes and text nodes are addressable. While element nodes expose an `id` attribute where the value field represents the actual identifier, text node identifiers are a combination of the parent node ID, which is always an element node, and the index position within the list of attached text nodes.

Participant Module

A second support module is the participant module which is apparently of interest for the participant list widget since it provides a collection of participants. Moreover, the services from the participant module are also consumed by other widgets (e.g. telecaret, telepointer, etc.) since it allows retrieving the color code that is associated to a specific participant.

To consume services from the participant module, we specified the `IParticipant` interface (cf. Figure 7.9). Consequently, an arbitrary software module implementing the `IParticipant` interface can act as the participant event provider (e.g. SAP Gravity). The `IParticipant` interface offers the method `getSessionId` to retrieve the identifier from the session where the local participant is active in. The `subscribeJoin` and `subscribeLeave` methods allow passing in functions that are called if participants join or leave the session. Furthermore, `getParticipants` and `getLocalParticipant` represent APIs to retrieve all participants or only the local one. And finally, the `getParticipantColor` retrieves the color code that is assigned to a specific participant.

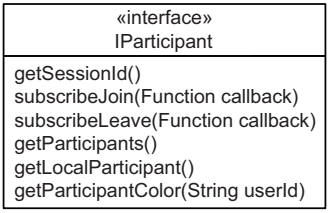


Figure 7.9: The IParticipant interface

Network Module

The network module is responsible for transporting awareness events from the originating client to other connected clients. Awareness event objects therefore have to be serialized into a text representation, propagated using a suitable protocol (e.g. the WebSocket protocol or HTTP) and deserialized to reconstruct the original awareness event.

In essence, the current network module implementation exposes three basic capabilities. First, the network module establishes communication

channels among all clients that are associated in the same session. Second, the broadcast capability allows distributing an awareness event object to all linked clients. And third, a publish-subscribe mechanism is offered to get notified about specific awareness events.

To implement bi-directional communication, we employ the JavaScript CometD library [115]. CometD is beneficial since it already exposes broadcast and publish-subscribe notification schemes. Additionally, CometD supports the modern two-way WebSocket protocol as well as the backwards-compatible HTTP providing bi-directional workarounds (e.g. HTTP polling or HTTP streaming techniques).

Operation Origin Module

As outlined in the GCI requirements section (cf. Section 2.3), synchronous groupware applications have to offer two distinct services: workspace awareness capabilities, on the one hand, and concurrency control, on the other hand. Concurrency control and workspace awareness are to a large extent independent of each other. Nevertheless, the workspace awareness adapter has to differentiate between local change operations initiated by end-users and replay operations triggered by the concurrency control engine. For example, inserting a text in a collaborative word processor locally represents relevant input for the creation coloring widget. However, the remote replay of the text insert operation should be neglected by installed awareness widgets. Therefore, the operation origin module provides information about the source triggering DOM manipulations.

To implement the operation origin service, the awareness adapter exposes the `IOperationOrigin` interface (cf. Figure 7.10). The interface offers the `beforeOperationReplay` method installing a function `callback` that triggers necessary prerequisites to neglect future DOM operations. Moreover, the `afterOperationReplay` method installs another `callback` function that activates listeners to again monitor future DOM operations.

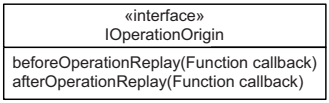


Figure 7.10: The IOperationOrigin interface

Initialization Module

The initialization module is in charge of bootstrapping the awareness adapter infrastructure which consists of the following steps.

Configuration Read: The initialization module reads and processes the dedicated configuration file *gaa.config* that collects setup properties. On

the one hand, the configuration file lists the JavaScript files that accommodate the implementation of the various modules (e.g. the implementation of the `IOperationOrigin` interface). On the other hand, the configuration contains a list of awareness widgets that should enrich the specific application. Note that awareness widgets do not increase efficiency in every collaboration scenario. For example, adopting telepointers for large-group collaborations may decrease collaboration productivity, since the plethora of telepointer embodiments can confuse participants.

Initialization of Support Modules: Before setting up event and widget modules, support modules have to be initialized. The initialization is carried out according to the following setup order: (1) participant module, (2) identifier module, (3) network module and (4) operation origin module. During this initialization, the session identifier and the participants are determined, DOM nodes are enriched with identifiers, communication channels are established, etc.

Initialization of Main Modules: Once the infrastructure bootstrap is finished, the awareness widgets and event modules are initialized. First, the awareness canvas is created representing the `<div>` container that encapsulates all awareness-related visuals. Second, the `init` method of each widget module is called which renders an initial widget user interface. Third, the `getEventTypes` method is invoked to retrieve the list of relevant awareness events. The publish-subscribe mechanism leverages this list to link widgets to the corresponding event modules.

Switch to Operation's Mode: After the initialization module concludes setup tasks, the awareness adapter starts to work in the operation's mode. Then, awareness events are created upon user interactions or model changes, event objects are distributed and dispatched and awareness widgets are updated.

7.2.5 Awareness Adapter Incorporation

Before the initialization module bootstraps the awareness infrastructure at runtime, developers have to carry out a number of design time tasks. This includes adopting a concurrency control provider, specifying awareness adapter properties in the configuration file and embedding the awareness adapter in the web application.

Figure 7.11 embeds the awareness adapter integration in the overall development methodology for collaborative web applications (cf. Section 4.4) and illustrates the individual steps for anchoring the awareness adapter in standards-based web applications. The overall methodology shown in Figure 7.11 starts with the concurrency control integration that depends on the development project type (transformation approach or from-scratch development). Once the concurrency control capabilities are successfully embedded (cf. Section 5.2 and Section 6.2), programmers have to select the appropriate

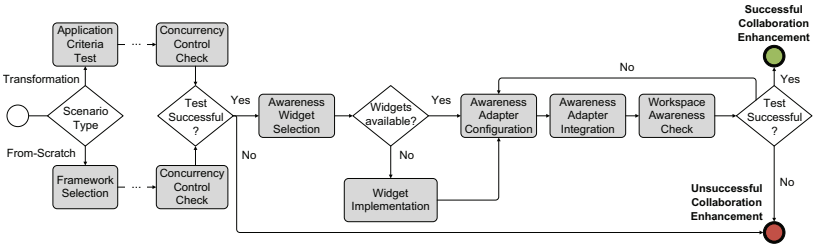


Figure 7.11: Workspace awareness integration embedded in the overall development methodology

awareness widgets from the widget library. If a required awareness widget is not available, further widgets can be implemented and the implementations have to adhere to the `IWidget` interface. The selection of awareness widgets has to be specified in the *gaa.config* file. Setup properties such as the URL of the central collaboration server, implementations of external event providers (e.g. the participant event provider) or the interface of the concurrency control provider also have to be defined in this configuration. After the configuration step, the *gaa.js* file materializing the workspace awareness adapter has to be embedded in the web application's HTML code. This requires including the *gaa.js* as an additional script in the `<head>` section of the application's HTML code. The enhanced web application can be executed using a modern browser (e.g. Google Chrome, Mozilla Firefox, etc.) and workspace awareness features can then be tested. If the awareness features do not work correctly, a re-configuration may fix the issue. However, the application may also suffer from an awareness adapter limitation (cf. Section 7.4.2) which impairs the awareness adapter adoption.

7.3 Evaluation

Besides showing the technical feasibility of a generic workspace awareness adapter, a second research question is the usability of reusable awareness widgets. To analyze the acceptance of awareness widgets from the widget library (cf. Section 7.2.3), we conducted a usability study where 20 participants assessed various software quality characteristics using two collaborative editors. In line with the usability study evaluating the DOM adapter (cf. Section 5.3.1), we again leveraged the collaborative graphics editor SVG-edit [36] and the multi-user word processor CKEditor [35].

In the following sections, we discuss the considered software quality characteristics, the evaluation procedure as well as the evaluation results.

7.3.1 Evaluation Characteristics

In this usability study assessing primarily the quality of facilitated workspace awareness support, we rely on the quality model established in the evaluation of the DOM adapter (cf. Section 5.3.1). This quality model incorporated a general software quality catalog (the ISO/IEC 9126 standard for product quality [117]) as well as a collaboration-centric catalog (the mechanics of collaboration [118]). The quality model defined in Section 5.3.1 embraces the ISO/IEC 9126 characteristics *functionality*, *reliability*, *usability* and *efficiency* as well as the mechanics of collaboration aspects *communication* and *coordination*.

Since we aimed to specifically assess the workspace awareness capabilities, we adjusted the weight for each individual quality characteristic by changing the number of questions targeting a certain aspect. First, in contrast to the DOM adapter evaluation, we highlighted the functionality aspect, since awareness widgets cover a broader functionality range. While it is simple to assess if the synchronization engine is working properly, awareness widgets inform about presence and identity of participants, document changes, collaborators' viewports, etc., and therefore, the functionality assessment is more complex. Second, the reliability facet covering maturity and recoverability did not receive as much attention as in the DOM adapter study, since implementing lightweight widgets like participant lists or telecarets is by no means as sophisticated and intricate as building a DOM adapter for OT-based synchronization. And third, because the literature [143, 144] already shows that adopting awareness widgets in collaboration scenarios increases efficiency, we also lowered the effort to assess the efficiency characteristic. The mechanics of collaboration aspects (communication and coordination) as well as the usability dimension are just as important for the analysis of the workspace awareness adapter as for the DOM adapter analysis. Thus, the significance of the communication, coordination and usability aspect did not change.

7.3.2 Evaluation Procedure

Before conducting the usability study, various evaluation aspects had to be planned in advance. These aspects include recruiting participants, selecting representative groupware applications, creating proper collaborative tasks, composing an evaluation schedule and designing a suitable questionnaire.

Participants

In essence, we attracted 20 volunteers to take part in the usability study. Recruited subjects for the usability study were students or employees from the SAP Research Center in Dresden. All participants had an information technology background, since they were still studying computer science or

they already had a degree in a related subject. To carry out the collaborative editing tasks, the multi-national group of 20 volunteers was divided into teams of two. We considered the provenance of participants and grouped people with the same cultural background.

Application Selection

To validate the workspace awareness adapter and the accompanied widget library, we had to select representative tools. Primarily, we aimed to cover multiple application domains to show the universality of the approach. Secondly, we wanted to design a usability study that was comparable to the DOM adapter evaluation because this allows proving the additional value of workspace awareness support for joint work. And third, the effort for the 20 participants should be moderate, i.e. not exceed a one-hour assessment.

Taking into account the listed objectives, we decided to reuse the collaborative word processor CKEditor and the multi-user graphics editor SVG-edit that were adopted in the DOM adapter evaluation. Both editors can benefit from the widget library since incorporated workspace awareness features are either rudimentary or not available at all. Moreover, because both tools already facilitate the second required collaboration service, namely, concurrency control, the editors can immediately take advantage of the workspace awareness adapter.

Tasks

When designing collaborative tasks, we again adhered to the task design principles introduced in the DOM adapter evaluation (cf. Section 5.3.1). Hence, (1) tasks should encourage participants to work continuously and interactively; (2) tasks should be independent of the subject's knowledge and skill set; (3) tasks should embrace realistic assignments and (4) tasks should not constrain participants in the way they use the groupware application. To leverage awareness widget features to a large extent, we divided the overall tasks into various task types [145] such as artifact construction, identification and exploration as well as organization and arrangement.

The first assignment aimed to reproduce a given graphic picturing a snowman. Consequently, teammates received a handout of the envisioned snowman graphic (cf. Appendix C.2). The initial collaborative SVG-edit was preloaded with a preliminary draft of the snowman vector graphic. This draft version included some parts of the final graphic (e.g. the hat or the scarf) that were arranged in an arbitrary manner. Other parts (e.g. the eyes or the nose) were missing and had to be constructed. Providing some parts and having to construct some parts of the graphic ensured the desired mix of task types (e.g. arranging and creating objects).

The second assignment asked teammates to author an invitation letter for a Christmas party. Again, participants received a printout picturing the expected result (cf. Appendix C.2). The invitation should accommodate varying material (formatted text, tables and graphics) where some artifacts were already given and some artifacts needed to be created. Using the multi-user version of the CKEditor, teammates had to coordinate and communicate responsibilities for creating, arranging and formatting the invitation letter.

Physical Setup and Schedule

In the evaluation of the workspace awareness adapter, we adopted the established physical setup that was used in the DOM adapter evaluation (cf. Figure 5.10). Hence, each participant worked on its own desk equipped with a PC running the collaborative web applications (SVG-edit and CKEditor). The two desks were located in the same room but separated by a room divider. Thus, participants could talk to each other without being able to see the other participant or the other participant's screen. This mimics a teleconference setting where participants are connected via an audio channel.

The schedule is also in line with the DOM adapter evaluation comprising (1) a 5 min tutorial, (2) a 15 min shared editing session and (3) a 10 min questionnaire completion phase. Both collaborative tools are evaluated according to the defined schedule which results in a one-hour assessment. First, the tutorial prepares each participant individually to get acquainted with the editor's capabilities. Therefore, written instructions are provided (cf. Appendix C.1) to carry out 10 simple tasks (e.g. draw a circle). Second, teammates work jointly on the given collaborative tasks (snowman graphic or invitation card) and third, a questionnaire has to be completed.

Data Collection

To measure the perceived awareness quality, we created the questionnaire (cf. Appendix C.3) depicted in Figure 7.12 and in Figure 7.13. The questionnaire comprises 22 questions divided into the software quality aspects *functionality*, *reliability*, *usability* and *efficiency* as well as the collaboration aspects *communication* and *coordination*. As stated in the evaluation characteristics section (cf. Section 7.3.1), we focused on the functionality characteristic and did not pay as much attention to reliability and efficiency. Therefore, in contrast to the DOM adapter evaluation, we increased or reduced the number of questions targeting the corresponding characteristic.

During the one-hour assessment procedure, each participant had to fill out two questionnaires. While one questionnaire addressed the awareness capabilities of the SVG-edit, the other questionnaire was tailored for the CKEditor. Participants of the usability study could rate questions according

to the seven-level Likert scale that offers the options: (1) strongly disagree, (2) disagree, (3) disagree somewhat, (4) neither agree nor disagree, (5) agree somewhat, (6) agree and (7) strongly agree.

7.3.3 Evaluation Results

To analyze the data captured by 40 completed questionnaires, we calculated the mean μ and the standard deviation σ for the SVG-edit and for the CKEditor assessment. Figure 7.12 and Figure 7.13 summarize the results (cf. Appendix C.4) derived from the questionnaires. In Figure 7.12 and Figure 7.13, the mean μ is represented by grey bars and the standard deviation σ by black error bars.

Functionality aspects in general received high ratings with $\mu \geq 5.10$. As expected, users were aware of the presence and identity of other participants (cf. Q1: $\mu_{svg} = 6.35$, $\mu_{ck} = 6.25$). Apparently, the participant list could precisely convey who is actually present in the shared workspace. Another awareness property answering the question where people are currently working also received high ratings (cf. Q6: $\mu_{svg} = 6.40$, $\mu_{ck} = 6.50$). This rating can mainly be attributed to radar views and telepointers. Even though ratings were slightly lower, the questions regarding the origin of changes and the work focus of collaborators still had compelling results (cf. Q2, Q3 and Q5: $\mu \geq 5.75$). Surprisingly, question Q7 considering the visibility of objects could not deliver exceptional results (cf. Q7: $\mu_{svg} = 5.55$, $\mu_{ck} = 5.70$). Maybe, the viewport visualization in the radar view was not clear to all

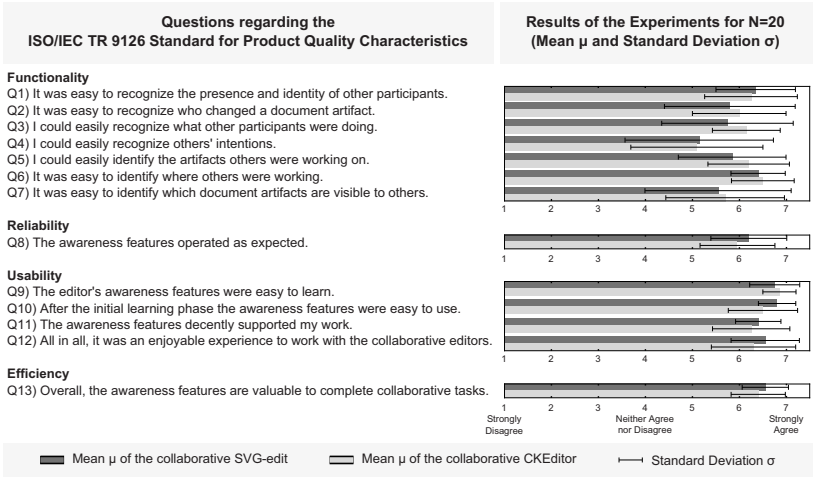


Figure 7.12: Questions and results regarding the ISO/IEC 9126 quality characteristics

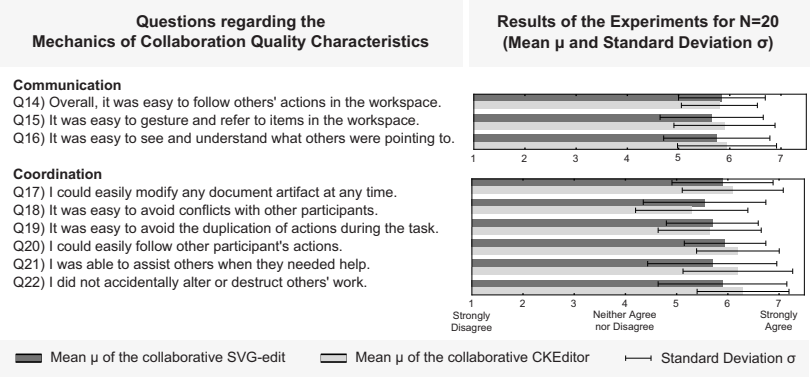


Figure 7.13: Questions and results regarding the mechanic of collaboration characteristics

participants. Recognizing the intention of others was moderately supported receiving the lowest functionality ratings with a large dispersion (cf. Q4: $\mu_{svg} = 5.15$, $\sigma_{svg} = 1.59$; $\mu_{ck} = 5.10$, $\sigma_{ck} = 1.41$). Note that conveying users' intentions is intricate in groupware applications because the intention is expressed through gaze, gestures and facial expressions that are difficult to capture with conventional workspace awareness widgets.

Usability ratings were the highest in the entire usability study. Subjects of the laboratory experiment confirmed that it was easy to learn and use awareness widgets (cf. Q8, Q9: $\mu \geq 6.50$). Moreover, people stated that their joint work profited from the provided awareness widgets (cf. Q10: $\mu_{svg} = 6.40$, $\mu_{ck} = 6.25$) and that the awareness support was suitable (cf. Q11: $\mu_{svg} = 6.55$, $\mu_{ck} = 6.30$).

Reliability was assessed by means of question Q12. The result of $\mu_{svg} = 6.20$ and $\mu_{ck} = 5.95$ demonstrates the quality of the awareness infrastructure in terms of system stability, fault tolerance and recoverability. Due to the low technical challenges that were imposed implementing awareness widgets, we expected these high ratings.

Efficiency in joint work scenarios strongly depends on the offered workspace awareness support which is again confirmed by the rating of question Q11 ($\mu_{svg} = 6.55$, $\mu_{ck} = 6.40$). We anticipated these results since they are in line with similar studies described in the literature [143, 144].

Communication results were in the range of $5.65 \leq \mu \leq 5.90$ representing a good rating. Following other users' actions or pointing to specific objects could easily be accomplished. In particular, the telepointer widget allows highlighting certain objects to ensure users communicate about the very same artifact and thus, the telepointer is mainly responsible for this high rating.

Coordination assesses how participants can steer their collaborative work which constantly received high ratings. The only slight outlier is represented by the lower rating for question Q18 asking if participants can easily avoid conflicts (cf. Q18: $\mu_{svg} = 5.55$, $\mu_{ck} = 5.30$). Since the synchronization mechanism always induces a minimal delay to incorporate remote changes, these conflicts might arise. The remainder of the coordination ratings addressing unconstraint changeability, the avoidance of work duplication, the monitoring and assistance of participants' actions as well as the protection from destructing others' work obtained high ratings $\mu \geq 5.65$.

In essence, this study shows the quality of workspace awareness support that can be reached using reusable awareness widgets. The high ratings demonstrate the suitability of application-agnostic widgets for real-life use cases. Moreover, the minimal rating difference between SVG-edit results and CKEditor results reveals that provided workspace awareness can indeed decently support various editors and multiple domains. The importance of workspace awareness could again be demonstrated comparing this study with the DOM adapter evaluation where only a participant list and creation coloring widget was supplied. Calculating the average for each analyzed quality characteristic (e.g. functionality, etc.) showed that all characteristic averages from the awareness adapter study are higher than the ones computed for the DOM adapter study (cf. Section 5.3.2). For example, the *communication* averages $\mu_{DOMAd.} = 4.99$ and $\mu_{Aw.Ad.} = 5.82$ differ substantially and the *usability* averages $\mu_{DOMAd.} = 6.02$ and $\mu_{Aw.Ad.} = 6.55$ still vary considerably.

7.4 Applicability

To assess whether the workspace awareness adapter including its widget library can be leveraged by a substantial fraction of web applications, we analyzed the awareness adapter applicability. When investigating the applicability, two aspects are of interest: (1) the range of awareness capabilities that is covered by the widget library and (2) the limitations that may impair the adoption of the workspace awareness adapter.

7.4.1 Awareness Element Coverage

In [63], the pioneers of workspace awareness Carl Gutwin and Saul Greenberg introduced a workspace awareness framework that consolidates and groups elements of workspace awareness (cf. Section 2.2.2). This awareness framework was developed "for the purpose of aiding groupware design" [63]. Since the established workspace awareness framework represents the de facto standard, our awareness coverage analysis is based on it.

The objective of this coverage analysis is to identify whether groupware applications are comprehensively supported by the widgets from the

awareness adapter library. Hence, we took into account the 6 widgets encompassed in the awareness library (cf. Section 7.2.3) and estimated to what extent they can support the specified elements of awareness [63]. Note that Gutwin and Greenberg explain in [63] which awareness element benefits from what awareness widget. We transformed the textual description into a 4-level scale dividing the awareness assistance into (1) no support, (2) minimal support, (3) good support and (4) excellent support (cf. Figure 7.14). The 10 awareness elements categorized into the *who*, *what* and *where* classes and our estimation of awareness support are depicted in Figure 7.14. Subsequently, we discuss the awareness elements and their support through library widgets.

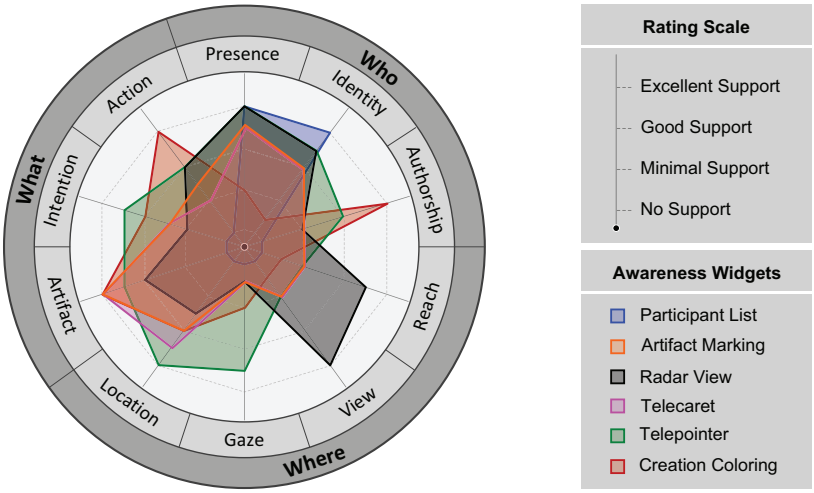


Figure 7.14: Awareness element support offered by library widgets

Presence illustrates if there is anybody in the shared workspace. As shown in Figure 7.14, widgets like the participant list or the telepointer indicate the presence of collaborators. Since presence is the most basic awareness element, numerous widgets are able to convey it reasonably.

Identity shows who is participating in the workspace. Primarily, the participant list identifies users with a name, a picture, etc. Furthermore, widgets adopting a color code (e.g. telepointer, artifact marking, etc.) may also convey the identity property to some extent.

Authorship depicts who is doing a create or change operation. Apparently, the dedicated creation coloring widget is in charge of displaying the authorship element. Moreover, the proximity of an action and a person's representation is also a strong authorship clue which qualifies the telepointer to report about authorship.

Action awareness exhibits what participants are doing. The best support for the action element provides the creation coloring widget that briefly highlights the modified artifact leveraging the color code from the participant list. But also a radar view showing the entire workspace including recently changed objects can announce actions. The telepointer's proximity to an action also represents a valuable indicator.

Intention communicates what the goal of an action is. Understanding the intention of other users requires being able to anticipate. Even though the intention support is not excellent, the telepointer movement allows predicting what people are going to do. For example, if a telepointer moves directly to a certain toolbar icon, it is likely that the icon will be selected.

Artifact awareness discloses what objects participants are working on. Artifact marking and creation coloring widgets transmit meaningful awareness clues about objects that are in the work focus. The proximity of a telepointer and the artifact can also be leveraged to derive insights about the work focus.

Location shows where people are working. The telepointer and the telecaret precisely report about the location of participants in the shared space. The creation coloring and the artifact marking widgets can only temporarily supply clues about the locations since participants do not always modify artifacts or select objects.

Gaze conveys where participants are looking at. Commonly, the participant's gaze and the telepointer's position correlate, which allows for a rough gaze estimate. The current widget library leaves room for improvement supporting the gaze element. For example, a video link showing other participants could decently inform about the gaze of others.

View awareness communicates which document part collaborators see. This is the main purpose of the radar view exposing viewports of other participants. Hence, visualizing viewports entails that participants are very aware of the view of others.

Reach discloses where people can immediately reach out to. The viewports exposed by radar views enable participants to assess the reach. Nevertheless, this estimation is only a rough indicator.

In essence, Figure 7.14 represents a good summary of the workspace awareness support offered by the awareness widget library. The trend becomes apparent that the 6 awareness widgets cover a broad range of the distinct elements of workspace awareness. Moreover, the widgets altogether can provide strong and comprehensive awareness support for collaborative web applications. However, the support for gaze, reach and intention is only modest and these awareness elements could profit from further widget implementations.

7.4.2 Limitations

Besides analyzing the awareness coverage of the widget library, we identified a set of limitations that may impair the applicability of the workspace awareness adapter.

Differing Document Representations: Web applications may incorporate components with changeable view representations. For instance, a spreadsheet application may provide the following options: (1) view the data in a table representation or (2) visualize the data in a chart. Such a representation switch incurs a DOM change, i.e. the DOM subtree representing for instance a table is disjoint from the DOM subtree showing the corresponding chart. Hence, position calculations for widgets (e.g. the telepointer) that leverage the underlying DOM nodes as reference points will break. Figure 7.15 illustrates the described situation. On the one hand, Figure 7.15a shows the table representation associated with the respective DOM subtree. And on the other hand, Figure 7.15b depicts the chart view and the associated DOM materialization. A mouse cursor pointing to the Google Chrome sector in the pie chart can currently not be mapped to the corresponding table field.

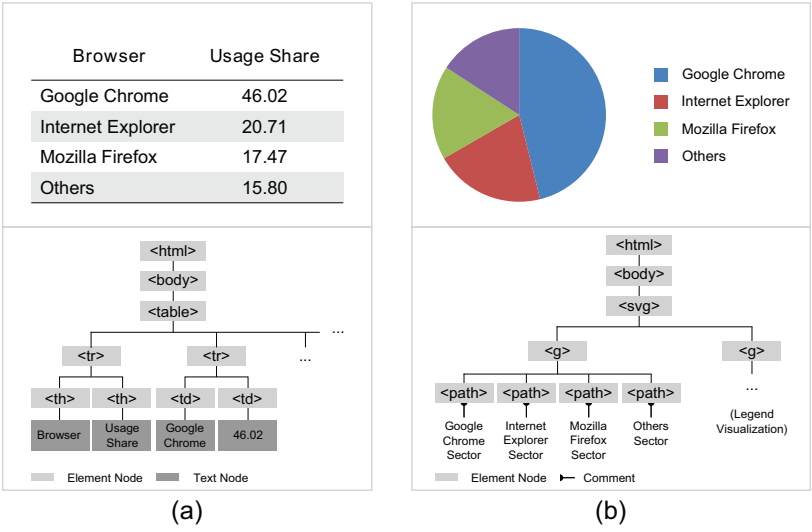


Figure 7.15: Example of differing component representations and the corresponding disjoint DOM subtrees

Cross-Browser DOM Inconsistencies: Differing browser engines (e.g. Internet Explorer, Mozilla Firefox, etc.) do not always produce the same document object model after applying the very same set of DOM manipulations. Generated inconsistent DOM instances may affect the oper-

ation's mode of the awareness adapter. For example, a concrete experiment showed that adding a line break in an HTML textarea results in splitting one text node into two text nodes. Removing this line break once again results in diverging DOM instances. While some browser engines merge the text nodes, other browser engines keep two separate text nodes. These DOM inconsistencies lead to reference problems if awareness widgets (e.g. telecaret, telepointer) refer to DOM nodes that are not available in all DOM instances.

Plugin-based Web Applications: As described in Section 7.2, the awareness adapter relies to a large extent on existing W3C specifications (e.g. DOM Core [40], DOM Events [106], HTML5 Editing API [107], etc.) that are implemented by all major browsers. For example, DOM events such as `mouseover` or `DOMNodeInserted` are used to capture workspace changes or DOM manipulations, the HTML5 Editing API is used to compute ranges for artifact marking or creation coloring widgets and regular DOM Core commands like `createElement` or `createAttribute` are adopted to highlight certain areas or to construct entire widgets (e.g. the participant list). However, plugin-technologies like Adobe Flash, Microsoft Silverlight or Java Applets do not comply with W3C specifications and are thus not eligible to adopt the workspace awareness adapter.

7.5 Summary

In this chapter, we elaborated on the workspace awareness adapter that allows enriching standards-based web applications with comprehensive awareness support. Besides offering numerous reusable awareness widgets in form of a pre-built library, the awareness adapter can be incorporated in a non-invasive manner. The presentation of the awareness adapter encompassed three major parts. First, we showed the technical feasibility of an application-agnostic awareness adapter. Second, a usability study with 20 participants demonstrated the decent awareness support provided by the current implementation of the awareness adapter. And third, we illustrated that the awareness widget library covers Gutwin's elements of workspace awareness [63] to a large extent. Additionally, we also presented limitations hindering the adoption of the workspace awareness adapter.

Chapter 8

Overall Evaluation

After having presented the generic collaboration infrastructure and its core components, we expose the overall evaluation in this chapter. The overall evaluation is based on the conducted evaluations of the DOM adapter (cf. Section 5.3), the framework adapter (cf. Section 6.3) and the workspace awareness adapter (cf. Section 7.3).

First, we revisit the introduced use cases: the SVG-edit single-user to multi-user transformation as well as the CoBAT from-scratch development (cf. Section 2.1). Thereby, we discuss to what extent the defined collaboration requirements (cf. Section 2.1.1 and 2.1.2) can be fulfilled. Second, we explore to what extent the overall methodology (cf. Section 4.4) meets the groupware development requirements that were established in Section 2.3.

8.1 Requirements Fulfillment Analysis for Introduced Use Case Scenarios

In Section 2.1, we introduced two development use cases for web-based groupware. Each use case aimed to analyze one prevalent development approach. While the graphics editor SVG-edit was selected to carry out a single-user to multi-user transformation, CoBAT represented the from-scratch development. The introduction of the two use cases was accompanied by a set of individual requirements that should ensure the resulting collaborative web applications are suited for shared editing. In the following, we will discuss the fulfillment of these requirements.

8.1.1 SVG-edit Transformation

In Section 2.1.1, we defined 5 concurrency control requirements (CCR1 - CCR5) and 2 workspace awareness requirements (WAR1, WAR2). Obeying these requirements and adopting the GCI, we were able to transform the SVG-edit into a collaborative multi-user version.

Thereby, we processed the concurrency control enhancement using the DOM adapter which includes checking necessary and critical transformation criteria, selecting the DOM sync tree as well as embedding and configuring the DOM adapter (cf. Figure 5.13). Additionally, we accomplished the workspace awareness enhancement adopting the workspace awareness adapter which encompassed selecting suitable awareness widgets as well as embedding and configuring the awareness adapter component (cf. Figure 7.11).

The converted SVG-edit was leveraged for the DOM adapter usability study as well as for the awareness adapter usability study. In summary, in these studies, 50 participants worked collaboratively in teams of two. The corresponding end-user ratings (cf. Section 5.3.2 and 7.3.3) overall show the decent support for concurrency control as well as workspace awareness. Nevertheless, we discuss each requirement in detail taking into account appropriate questions from the end-user studies.

CCR1 (support for joining and leaving participants): The most basic collaboration requirement allowing participants to enter or leave a collaborative session was effectively supported. All collaborative sessions in the two corresponding studies could be successfully initiated and collaborators could eventually work jointly.

CCR2 (support for single-user capabilities): Another basic requirement should ensure that the original single-user capabilities are also properly facilitated by the collaborative editor. Single-user editor operations like creating, resizing, moving or coloring shapes were extensively used in the draw floor-plan exercise (cf. Section 5.3.2) as well as in the arrange and complete snowman graphic assignment (cf. Section 7.3.2). In both studies, all questions targeting the usability characteristic received $\mu \geq 5.37$ demonstrating that basic editor operations did work as expected.

CCR3 (unconstrained manipulation support): The requirement targeting unconstrained manipulation demands that the edit operations of one user do not hinder the edit operations of other users. Specifically, question Q20 in the DOM adapter study tests this ability asking if people could easily start using any tool at any time. The rating of $\mu = 6.07$ illustrates that the multi-user SVG-edit was able to support unconstrained collaboration.

CCR4 (document synchronization): One of the most important requirements is document synchronization ensuring participants work on consistent document copies that are constantly synchronized in real-time. For instance, question Q12 of the DOM adapter study assessing whether participants were satisfied with the synchronization latency resulted in $\mu = 5.33$ which reveals the compelling sync support.

CCR5 (conflict resolution): Another crucial requirement is conflict resolution that allows automatically resolving a conflict from two change operations pursuing a contradictory intension (e.g. resize shape and delete shape). To some extent, question Q22 of the DOM adapter study assesses

this quality asking if collaborators were satisfied how they could jointly edit the document. Again, the rating $\mu = 5.77$ demonstrates convenient conflict resolution support.

WAR1 (participant list): A basic workspace awareness requirement is the availability of a participant list. As described in Section 7.2.3, the awareness adapter library accommodates such a widget. Subjects of the awareness adapter usability study also acknowledged and appreciated the presence information conveyed by the participant list. For instance, Q1 in the awareness adapter study asks whether participants could easily recognize presence and identity which was assessed with $\mu = 6.35$.

WAR2 (comprehensive awareness support): Besides the elementary participant list, the WAR2 requirement demands comprehensive awareness support exposed through further awareness widgets (e.g. telepointer, radar view or creation coloring widget). The widget library includes currently 6 widgets (cf. Section 7.2.3) and can thus deliver comprehensive awareness support. Again, the awareness adapter study underpins that the workspace awareness support is advanced. For example, question Q13 assessing whether awareness features are of value to complete collaborative tasks was rated with $\mu = 6.55$.

8.1.2 CoBAT From-Scratch Development

In contrast to the SVG-edit conversion scenario, where the evaluation focused on the usability of the resulting multi-user web application, the from-scratch evaluation aimed to analyze the development efficiency. The reason for the focus shift of the evaluation is twofold. First, creating a sophisticated application comparable to the transformed editors (e.g. SVG-edit, CKEditor, TinyMCE) is not realizable in a limited evaluation time frame since it requires several person years of development. Hence, we selected the modestly-sized CoBAT use case. Second, in contrast to adopting a configuration-based transformation approach, injecting collaboration capabilities using source code annotations is similar to the traditional invasive programming model leveraging libraries. Hence, a direct comparison of a conventional collaboration library and an annotation-based approach is of value since developers can understand advantages as well as disadvantages considering the related approaches. However, due to the focus shift of the evaluation, we cannot additionally use data derived from a usability study in order to validate CoBAT requirements (cf. Section 2.1.2).

Nevertheless, we briefly assess to what extent the defined CoBAT requirements could be fulfilled. Therefore, we leverage results from the developer study (cf. Section 6.3.3) where 8 developers had to implement the collaborative CoBAT application. The CoBAT requirements (cf. Section 2.1.2) included three single-user feature requirements (SUFR1 - SUFR3), four concurrency control requirement (CCR1 - CCR4) as well as one workspace

awareness requirement (WAR1). To implement the collaboration capabilities the enhanced Knockout framework was adopted which represents an embodiment of the framework adapter approach (cf. Chapter 6). Conducting the developer study, a tutor was in charge of assessing whether the requirements were or were not fulfilled. Finally, from 8 handed in prototypes only 1 was rejected, i.e. 7 prototypes fulfilled all CoBAT requirements. In the following, we concisely describe how students typically implemented the given requirements.

The single-user requirements (SUFR1 - SUFR3) included implementing the CoBAT user interface according to the given mockup (cf. Figure 2.3). Moreover, end-users should be able to add, remove or reorder cost-benefit factors. This was commonly accomplished using regular HTML elements. Solely for the drag-and-drop functionality allowing to reorder items, students usually adopted an extra JavaScript library (e.g. jQuery [27]).

The concurrency control requirements (CCR1 - CCR4) encompassed support for simultaneous edits, document synchronization as well as automatic conflict resolution. For the implementation, students used the given annotation vocabulary `@Sync` and `@Class` to tag the Knockout data model as well as the constructor functions. Due to the fact, that 7 out of 8 prototypes successfully materialized the concurrency control requirements, the given annotation vocabulary seems appropriate.

The workspace awareness requirement (WAR1) was limited to incorporate a participant list. Therefore, students configured the workspace awareness adapter to include the participant list from the widget library.

8.2 Suitability of the Development Methodology

The presented use cases are a means to validate the proposed development methodology for collaborative web applications. While the requirements fulfillment for the SVG-edit use case shows the feasibility of the transformation approach, meeting the CoBAT requirements demonstrates the applicability of the from-scratch approach.

Figure 8.1 illustrates the methodology validation leveraging the two use case scenarios. Initially, developers have to decide whether to adopt the transformation or the from-scratch development path. Leveraging the transformation approach (e.g. the SVG-edit use case) starts out adding concurrency control functionality by means of the DOM adapter (cf. Chapter 5) and proceeds integrating workspace awareness capabilities using the workspace awareness adapter (cf. Chapter 7). Adopting the from-scratch methodology (e.g. the CoBAT use case) differs in terms of adding concurrency control capabilities which is realized using the framework adapter (cf. Chapter 6). Figure 8.1 again highlights the two essential collaboration services: (1) concurrency control and (2) workspace awareness.

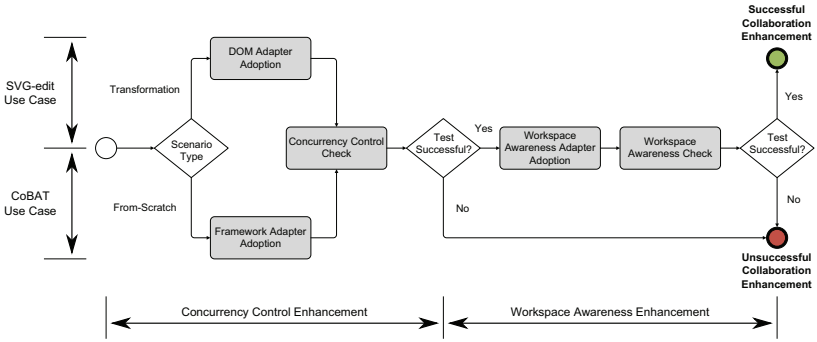


Figure 8.1: Development methodology validation including corresponding use cases and introduced collaboration services

Besides the supplementary methodology validation by means of use cases, the requirements defined in Section 2.3 are decisive for the evaluation of the development methodology. The requirements are divided into functional requirements (concurrency control and workspace awareness) as well as efficiency requirements (minimal invasiveness, encapsulation, learnability, reuse and universality). In the following, we assess the individual requirements exploiting the gained experience building collaborative applications as well as the findings from the three conducted evaluation studies which are (1) the DOM adapter usability study (cf. Section 5.3), (2) the framework adapter developer study (cf. Section 6.3) as well as (3) the awareness adapter usability study (cf. Section 7.3).

Concurrency Control: The GCI incorporates two capable concurrency control providers (the DOM adapter and the framework adapter) to embrace differing development approaches. To rate the quality of the delivered concurrency control, we considered three dimensions: the completeness, the flexibility as well as the integration dimension. Taking into account the fulfilled use case requirements (cf. Section 8.1) and the evaluation results (cf. Section 5.3 and 6.3), we conclude that the synchronization and conflict resolution support is complete and sound. Nevertheless, the framework adapter furnishes more robust concurrency control since it is not plagued by DOM inconsistencies which arise in cross-browser scenarios (cf. Section 5.4.2). Both concurrency control providers are flexible due to their broad data structure support (e.g. tree or graph models) and extensive application domain support (e.g. collaborative writing, shared graphics editing, etc.). From an integration point of view, three interfaces are offered: the DOM adapter's configuration-based interface as well as the wrapper interface and the annotation interface which are both provided by the framework adapter. All three interfaces are lightweight and can easily be combined with existing

development approaches which illustrates the ease of adoption furnished by the GCI concurrency control providers.

Workspace Awareness: Assessing the functional requirement workspace awareness, we again considered the completeness, flexibility and integration dimension. The workspace awareness adapter is the dedicated GCI component offering awareness capabilities (cf. Section 7.2) that are materialized by the awareness widget library. This widget library encompasses a participant list, a radar view, a telepointer, a telecaret as well as widgets for artifact marking and creation coloring. On the one hand, the completeness of awareness support can be demonstrated by means of the successful implementations of the two introduced use case scenarios (cf. Section 8.1). On the other hand, the analysis of the awareness element coverage for the widget library (cf. Section 7.4.1) also shows the awareness comprehensiveness and completeness. Since awareness widgets exclusively rely on the W3C APIs (cf. Section 7.2), they can be used for a broad range of standards-based web applications. This flexibility in terms of application support is only limited by not supporting plugin-based applications (cf. Section 7.4.2). Like the DOM adapter, the workspace awareness adapter also offers a configuration-based interface. Embedding a single JavaScript dependency and completing a specific configuration file facilitates a simple integration approach.

Minimal Invasiveness: The first development efficiency requirement is especially key for the transformation approach where existing single-user applications are exploited for collaborative work. Thereby, having to invasively change the source code requires familiarizing with the entire source code base which is a time-consuming endeavor, in particular, for large web applications comprising tens of thousands of lines of code. Thus, we designed the interfaces for the DOM adapter and the workspace awareness adapter, which are both adopted for the transformation approach, in a non-invasive manner. Both interfaces are configuration-based requiring to complete an external configuration file. The sole invasive operation is to anchor a JavaScript import in the main HTML file of the web application. Therefore, the transformation approach is indeed minimal invasive. In contrast, a from-scratch development can assume source-code familiarity and therefore, the invasiveness requirement is of secondary importance. Consequently, the devised wrapper approach (cf. Section 6.2.4) and the annotation-based approach (cf. Section 6.2.5) are more invasive than the transformation approach. Both strategies necessitate anchoring library calls (in form of function calls or annotations) directly in the application's source code.

Encapsulation: Since well-encapsulated software artifacts offering clear interfaces are easier to implement and maintain, we also considered encapsulation as beneficial for development efficiency. Devising the GCI, we targeted the encapsulation requirement in two distinct fashions. First, we implemented the GCI in a strictly component-based manner. For example, the three GCI core components (DOM, framework and workspace aware-

ness adapter) are clearly separated. Also the core components themselves adhere to component-based design which is for instance illustrated by the various modules (e.g. widget and event modules) that form the workspace awareness adapter (cf. Section 7.2). Second, we aimed to embrace the separation of concerns principle not only from an architecture point of view resulting in component-based design but also from an integration point of view. The non-invasive DOM and workspace awareness adapter require an external configuration that prevents intermingling single-user feature implementations with multi-user functionality. Incorporating annotations is also a capable means to inject collaboration capabilities. Thereby, annotations are apparently distinguishable from the single-user feature implementation. Only the wrapper approach (cf. Section 6.2.4) falls short of clearly separating single-user and multi-user functionality in the respective source code.

Learnability: To quickly adopt a development methodology, learnability is an essential requirement. Therefore, the interface design allowing using GCI core components is paramount for developers. Hence, we have to distinguish between the configuration-based interfaces for the DOM and workspace awareness adapter as well as the wrapper and annotation-based interfaces for the framework adapter. Learning to configure a number of properties in a distinct file induces little effort and therefore, the DOM and workspace awareness adapter can easily be adopted. Developers can also easily get acquainted with the API exposed by the wrapper approach since the multi-user API is actually equal to the original single-user API (cf. Section 6.2.4). The last interface materialized by a set of annotations is also lightweight in terms of learnability. As shown in Section 6.2.5, the annotation vocabulary for example for the Knockout framework only encompasses the two annotations `@Sync` and `@Class`. Developers have to solely understand where these annotations have to be placed.

Reuse: To lower development effort and maintenance resources, the reuse principle is one of the established measures. Devising the GCI, we strongly obeyed this principle. First of all, we designed our collaboration infrastructure in a *generic* fashion so that a plethora of web applications can be built on top of the GCI. We demonstrated the reuse of the GCI in multiple ways. On the one hand, we transformed numerous single-user applications (e.g. SVG-edit, CKEditor or TinyMCE) into collaborative ones that all use the very same GCI (cf. Section 5.4.1). On the other hand, we constructed two framework adapters. While the SAPUI5 framework adapter can be utilized to implement a multitude of collaborative SAPUI5 applications (cf. Section 6.2.4), the Knockout framework adapter can be adopted to create a variety of multi-user Knockout applications (cf. Section 6.2.5). In addition to the application-agnostic GCI core components (DOM, framework and workspace awareness adapter), the operational transformation engine SAP Gravity was for example also reused for all implemented shared editing applications.

Universality: The last considered requirement analyzed to what extent the proposed development methodology for web-based groupware can be applied to arbitrary development projects. Analyzing state of the art development approaches (cf. Chapter 3), we carved out the two predominant development strategies. There are from-scratch developments (supported by libraries, frameworks and web engineering approaches) as well as application conversions (supported by transformation approaches). As depicted in Figure 8.1, the current development methodology supports from-scratch as well as transformation projects. While from-scratch scenarios embrace the framework and workspace awareness adapter, the transformation scenarios adopt the DOM and workspace awareness adapter. The applicability discussions in Section 5.4.1, 6.4.1 and 7.4.1 underline the universal nature of the devised methodology.

In Table 8.1, the assessments for the GCI transformation and the GCI from-scratch approach are summarized and compared with existing state of the art approaches (cf. Section 3.5). Thereby, the results demonstrate that the GCI approaches outperform existing solutions in terms of provided

	Concurrency Control	Workspace Awareness	Minimal Invasiveness	Encapsulation	Learnability	Reuse	Universality
SAP Gravity	++	○	-	○	+	○	+
ShareJS	+	-	-	○	++	○	○
Apache Wave	+	○	-	+	-	+	-
MAUI Toolkit	○	++	-	+	○	+	○
GroupKit	+	+	-	○	○	+	○
Transparent Adaptation	++	○	++	+	-	○	+
Flexible JAMM	+	+	++	+	-	+	○
JEIS	○	-	++	○	+	+	+
WebML	○	○	-	++	○	++	○
UWE	-	-	-	++	○	+	○
OOHDM	-	-	-	++	-	+	-
WebComposition	○	○	-	++	+	++	○
GCI Transformation Approach	+	++	++	++	++	++	+
GCI From-Scratch Approach	++	++	○	+	++	++	++

Support Levels: - Poor ○ Sufficient + Good ++ Excellent

Table 8.1: Overall assessment comparing the devised GCI approaches with the state of the art (cf. Section 3.5)

collaboration functionality but also in terms of delivered development efficiency. Nevertheless, as already mentioned elaborating on the individual requirements as well as discussing entailed limitations (cf. Section 5.4.2, 6.4.2 and 7.4.2), the proposed GCI approaches also leave room for improvement (e.g. tackling DOM inconsistencies in cross-browser settings or broadening the applicability of the transformation approach).

8.3 Summary

In this chapter, we aimed to validate the proposed development methodology for collaborative web applications. Thereby, we first revisited the introduced use case scenarios (the SVG-edit transformation and the CoBAT from-scratch development) and showed that both use cases could be implemented adopting the GCI and the accompanied development methodology. After discussing the requirements fulfillment for two specific scenarios, we widened the methodology discussion deriving a general methodology assessment. This assessment took into account functional and efficiency requirements that were defined in Section 2.3. We concluded this chapter showing that the proposed transformation and from-scratch development approach for web-based groupware deliver required multi-user functionality and increase development efficiency in comparison to existing state of the art approaches.

Chapter 9

Conclusion

In this dissertation, we explored efficient means to develop collaborative web applications offering shared editing functionality, i.e. numerous users can edit the very same document simultaneously. Thereby, we analyzed a transformation as well as a from-scratch development approach to facilitate the required synchronous groupware capabilities: concurrency control and workspace awareness. Both development strategies exploited the devised generic collaboration infrastructure that accommodates the three core components, namely, the DOM adapter, the framework adapter as well as the workspace awareness adapter. Conducting three usability and developer studies demonstrated that implemented web-based groupware can, on the one hand, satisfy established software quality characteristics (e.g. reliability, usability, etc.). On the other hand, these studies showed that the proposed development process improves development productivity in comparison to state of the art development approaches for web-based groupware.

Subsequently, we summarize the main research contributions of this dissertation. Furthermore, we discuss open research questions that could be addressed in future work.

9.1 Research Contributions

In line with the research objectives defined in Section 1.4, we discuss the contributions of this dissertation.

Efficient Workspace Awareness Support: The workspace awareness adapter encapsulating a number of reusable workspace awareness widgets (e.g. telepointer, telecaret, etc.) represents a first major research contribution due to its non-invasive and generic nature. Being able to adopt pre-built awareness widgets by completing a simple configuration and without having to change the application's source code massively eases the development of web-based groupware and thus, substantially reduces the development effort. Traditionally, workspace awareness widgets were not only

implemented from-scratch but also required invasive source code changes to anchor workspace awareness functionality in collaborative applications. Besides the non-invasiveness of the approach, the generic applicability as shown in Section 7.4 and by means of the two use case scenarios (cf. Section 8.1) illustrates the potential of reusable awareness support that embraces a myriad of standards-based web applications. Thereby, the research contribution was to identify and leverage an application-agnostic API for workspace awareness functionality instead of adopting the conventional approach where workspace awareness features were implemented using application-specific APIs.

Efficient Concurrency Control Support for From-Scratch Development Projects: The devised framework adapter speeds up from-scratch groupware development through a framework enhancement concept that allows efficiently introducing concurrency control capabilities. Thereby, general-purpose web frameworks (e.g. Knockout, SAPUI5, etc.) are enhanced by a collaboration engine exposing a wrapper interface or an annotation-based interface (cf. Section 6.2.4 and 6.2.5). Thus, web developers may leverage their general-purpose web framework of choice not only to build conventional web applications but also to create synchronous multi-user applications. Adopting a framework developers are already familiar with reduces development effort since there is no need to get acquainted with an extra concurrency control library. Concurrency control functionality can be incorporated using a wrapped API, which does not induce additional learning effort since the original and the wrapped API are equal in terms of the exposed interface. Besides leveraging the wrapper approach (cf. Section 6.2.4), concurrency control can also be facilitated adopting a small set of annotations (cf. Section 6.2.5) that again entails minimal learning effort.

Efficient Concurrency Control Support for Transformation Projects: The established DOM adapter (cf. Chapter 5) represents an efficient means to integrate concurrency control non-invasively in existing single-user web applications. In contrast to traditional libraries that require anchoring concurrency control capabilities directly in the application's source code, the DOM adapter necessitates solely a completed configuration file which drastically reduces development effort. Again, the research contribution is to facilitate the concurrency control in a non-invasive fashion because the non-invasiveness frees developers from the time consuming task of having to understand and to adapt the existing source code. Thereby, we identified application-agnostic programming interfaces and managed to link a generic concurrency control provider to these application-agnostic APIs in a way that is transparent to the original application.

Development Methodology for Web-Based Groupware: Besides targeting development efficiency when integrating collaboration services for specific development approaches, we carved out an overall development methodology for web-based groupware. This methodology serves as a guide

for developers supporting the task of selecting the appropriate development strategy (from-scratch or transformation project) as well as choosing suitable GCI components (DOM, framework and workspace awareness adapter). Moreover, the methodology guides programmers through the incorporation of the distinct collaboration services (concurrency control and workspace awareness). In essence, the methodology emerged from the extensive experience that we have gained exploring the research field of groupware development and from implementing numerous collaborative web applications. This groupware development experience condensed in the presented development methodology allows avoiding common development pitfalls and thus also increases development efficiency.

In summary, the four main research contributions of this dissertation all support the common goal of increasing efficiency when developing synchronous collaborative web applications.

9.2 Future Work

In addition to addressing GCI limitations, which are exposed in Section 5.4.2, 6.4.2 and 7.4.2, future research regarding the efficient development of web-based groupware should mainly respond to ongoing industry trends. These trends include (1) the evolution of native browser capabilities, (2) the movement from on-premise hosting to cloud-based hosting as well as (3) the growing device proliferation.

Browser Capability Evolution: In the light of the HTML5 movement, various new capabilities such as the WebSocket protocol for bi-directional communication, the audio or video element for native media playback, etc., have emerged as new browser capabilities removing the need for additional browser plugins. This trend continues and currently the W3C standardization body finalizes the "WebRTC 1.0: Real-time Communication between Browsers" specification [146]. WebRTC proposes a set of APIs that allow for in-browser voice call or video chat applications. In addition to the primary collaboration services concurrency control and workspace awareness, which were discussed in this dissertation, voice or video calls also support distributed team work. Nevertheless, voice and video call services are currently realized via non-native applications or even via specific hardware (e.g. telepresence hardware). Exploring the native integration of voice or video call functionality by means of the WebRTC APIs is one research topic of interest. A tighter integration of existing collaboration services could promote efficiency when working collaboratively since groupware applications may offer sophisticated human-computer interfaces (e.g. speech- or gesture-based interfaces for shared editing).

Cloud-Based Collaboration Service Provisioning: Nowadays, providing collaborative web applications requires a dedicated server infras-

tructure that delivers collaboration services such as concurrency control or workspace awareness. Nevertheless, setting up and operating an extra collaboration infrastructure is time-consuming and costly. Offering collaboration services as a cloud-based solution yields various research questions. First, it is of interest to what extent operation expenses can be reduced using off-the-shelf collaboration services. Second, the impact on initial development expenses adopting pre-built services might also be investigated. However, a third research question analyzing usage patterns from data collected in hundreds or even thousands of recorded collaborative sessions might be of particular interest. Captured usage patterns can shrink feedback loops from end-users to developers and can help to rapidly improve collaborative web applications. For example, auto-completion mechanisms for development environments could be advanced leveraging interaction data or user interfaces could be modified according to end-user preferences.

End-User Device Proliferation: A third ongoing trend is the device proliferation producing a plethora of web-enabled devices that range from traditional PCs to tablets and smartphones. From a technical prerequisites point of view, all those devices are eligible for consuming collaborative web applications. Nevertheless, even though browser capabilities increased over time on all device types and keep converging to a common capable functionality set, the form factor ranging from small mobile displays to large computer displays remains a major difference. This is especially critical for the visualization of workspace awareness widgets that conventionally have been tailored for large displays (e.g. participant lists or radar views). Thus, one research question is how information that is conveyed by participant lists or radar views can be mapped to small-screen devices. Another research field that is worthwhile to investigate is the adaptation of concurrency control algorithms to varying network connectivity settings. Concurrency control algorithms have been established without taking into account rapidly changing connectivity settings which is common for mobile devices. Thus, shared editing without network connectivity produces major document differences that may induce merge issues or even information loss.

Bibliography

[1] Clarence A. Ellis, Simon J. Gibbs, and Gail L. Rein. Groupware: Some Issues and Experiences. *Communications of the ACM*, 34:39–58, 1991.

[2] Google Docs – Online Documents, Spreadsheets, Presentations, Surveys, File Storage and More. `http://docs.google.com`, 2013.

[3] Clarence A. Ellis and Simon J. Gibbs. Concurrency Control in Groupware Systems. In *ACM SIGMOD International Conference on Management of Data (SIGMOD '89)*, pages 399–407, 1989.

[4] Carl Gutwin and Saul Greenberg. The Effects of Workspace Awareness Support on the Usability of Real-Time Distributed Groupware. *ACM Transactions on Computer-Human Interaction*, 6:243–281, 1999.

[5] Douglas C. Engelbart and William K. English. A Research Center for Augmenting Human Intellect. In *Fall Joint Computer Conference (AFIPS '68)*, pages 395–410, 1968.

[6] Neil Fraser. Differential Synchronization. In *ACM Symposium on Document Engineering (DocEng '09)*, pages 13–20, 2009.

[7] Saul Greenberg, Carl Gutwin, and Mark Roseman. Semantic Telepointers for Groupware. In *Australian Conference on Computer-Human Interaction (OZCHI '96)*, pages 54–61, 1996.

[8] Carl Gutwin, Saul Greenberg, and Mark Roseman. Workspace Awareness Support with Radar Views. In *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '96)*, pages 210–211, 1996.

[9] Clarence A. Ellis, Simon J. Gibbs, and Gail L. Rein. Design and Use of a Group Editor. In *Conference on Engineering for Human-Computer Interaction (EHCI '90)*, pages 13–25, 1990.

[10] Steven Xia, David Sun, Chengzheng Sun, David Chen, and Haifeng Shen. Leveraging Single-User Applications for Multi-User Collaboration: the CoWord Approach. In *Conference on Computer Supported Cooperative Work (CSCW '04)*, pages 162–171, 2004.

[11] Sylvie Noël and Jean-Marc Robert. Empirical Study on Collaborative Writing: What Do Co-Authors Do, Use, and Like? *Computer Supported Cooperative Work*, 13:63–89, 2004.

[12] Antonietta Grasso, Jean-Luc Meunier, Daniele Pagani, and Remo Pareschi. Distributed Coordination and Workflow on the World Wide Web. *Computer Supported Cooperative Work*, 6:175–200, 1997.

[13] E. James Whitehead, Jr. and Yaron Y. Goland. WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web. In *European Conference on Computer Supported Cooperative Work (ECSCW '99)*, pages 291–310, 1999.

[14] Stephen Mogan and Weigang Wang. The Impact of Web 2.0 Developments on Real-Time Groupware. In *International Conference on Social Computing (SocialCom '10)*, pages 534–539, 2010.

[15] Richard Bentley, Thilo C. Horstmann, and Jonathan Trevor. The World Wide Web as Enabling Technology for CSCW: The Case of BSCW. *Computer Supported Cooperative Work*, 6:111–134, 1997.

[16] Most Popular Websites - Netcraft. `http://toolbar.netcraft.com/stats/topsites`, 2013.

[17] Michael Chui, James Manyika, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Hugo Sarrazin, Geoffrey Sands, and Magdalena Westergren. *The Social Economy: Unlocking Value and Productivity through Social Technologies*. McKinsey Global Institute, 2012.

[18] Kyle McNabb. *A Look at the Improvements and Shortcomings of Microsoft Office 2007 Desktop Applications*. Forrester Research, 2007.

[19] Andrea Lunsford and Lisa Ede. *Singular Texts/Plural Authors: Perspectives on Collaborative Writing*. Southern Illinois University Press, 1990.

[20] Yogesh Deshpande, San Murugesan, Athula Ginige, Steve Hansen, Daniel Schwabe, Martin Gaedke, and Bebo White. Web Engineering. *Journal of Web Engineering*, 1:3–17, 2002.

[21] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. *Computer Networks*, 33:137–157, 2000.

[22] Nora Koch, Alexander Knapp, Gefei Zhang, and Hubert Baumeister. Uml-Based Web Engineering – An Approach Based on Standards. In *Web Engineering*, pages 157–191. 2008.

[23] Daniel Schwabe and Gustavo Rossi. An Object Oriented Approach to Web-Based Applications Design. *Theory and Practice of Object Systems*, 4:207–225, 1998.

[24] Hans-Werner Gellersen, Robert Wicke, and Martin Gaedke. WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle. *Computer Networks*, 29:1429–1437, 1997.

[25] Martin Gaedke and Klaus Turowski. Specification of Components Based on the WebComposition Component Model. In *Data Warehousing and Web Engineering*, pages 275–284. 2002.

[26] Martin Gaedke. *Komponententechnik für Entwicklung und Evolution von Anwendungen im World Wide Web*. Shaker Verlag, 2000.

[27] jQuery: The Write Less, Do More, JavaScript Library. `http://jquery.com/`, 2013.

[28] Etherpad lite. `http://etherpad.org/`, 2013.

[29] Alan Rickayzen. Collaborative Process Modeling. `http://scn.sap.com/community/bpm/business-process-modeling/blog/2012/03/20/`, 2013.

[30] Knockout : Home. `http://knockoutjs.com/`, 2013.

[31] ShareJS – Live Concurrent Editing in your App. `http://sharejs.org/`, 2013.

[32] Apache Wave – Welcome to Apache Wave (incubating). `http://incubator.apache.org/wave/`, 2013.

[33] Google Web Toolkit – Google Developers. `http://developers.google.com/web-toolkit/`, 2013.

[34] Chrome Web Store. `http://chrome.google.com/webstore/`, 2013.

[35] CKEditor. `http://ckeditor.com/`, 2013.

[36] SVG-edit – A complete Vector Graphics Editor in the Browser. `http://code.google.com/p/svg-edit/`, 2013.

[37] Backbone.js. `http://backbonejs.org/`, 2013.

[38] SAPUI5 SDK. `https://sapui5.netweaver.ondemand.com/sdk/`, 2013.

[39] Chengzheng Sun, Steven Xia, David Sun, David Chen, Haifeng Shen, and Wentong Cai. Transparent Adaptation of Single-User Applications for Multi-User Real-Time Collaboration. *ACM Transactions on Computer-Human Interaction*, 13:531–582, 2006.

[40] Arnaud Le Hors, Philippe Le Hgaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document Object Model (DOM) Level 3 Core Specification. http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/, 2004.

[41] Firefox Marketplace. http://marketplace.firefox.com, 2013.

[42] TinyMCE – Home. http://www.tinymce.com/, 2013.

[43] Eclipse Orion. http://www.eclipse.org/orion/, 2013.

[44] CircuitLab – Online Schematic Editor & Circuit Simulator. https://www.circuitlab.com/, 2013.

[45] Adobe Illustrator. http://www.adobe.com/products/illustrator.html, 2013.

[46] Inkscape. http://inkscape.org/, 2013.

[47] Erik Dahlstrm, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, and Jonathan Watt. Scalable Vector Graphics (SVG) 1.1 (Second Edition). http://www.w3.org/TR/SVG/, 2011.

[48] ISO/IEC. ISO/IEC 25010 Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2010.

[49] Robin Berjon, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer, and Ian Hickson. HTML 5.1 – A Vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/html5/, 2013.

[50] ECMA International. ECMAScript Language Specification (5.1 Edition). http://www.ecma-international.org/ecma-262/5.1/, 2011.

[51] Alan Dix, Janet E. Finlay, Gregory D. Abowd, and Russell Beale. *Human-Computer Interaction (3rd Edition)*. Prentice Hall, 2003.

[52] Philip A. Bernstein, David W. Shipman, and James B. Rothnie Jr. Concurrency Control in a System for Distributed Databases. *ACM Transactions on Database Systems*, 5:18–51, 1980.

[53] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[54] Claudia Lavinia Ignat. *Maintaining Consistency in Collaboration over Hierarchical Documents*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2006.

[55] Philip Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[56] Saul Greenberg. Personalisable Groupware: Accommodating Individual Roles and Group Differences. In *European Conference on Computer Supported Cooperative Work (ECSCW '91)*, 1991.

[57] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.

[58] David Sun and Chengzheng Sun. Context-Based Operational Transformation in Distributed Collaborative Editing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 20:1454–1470, 2009.

[59] Aguido Horatio Davis, Chengzheng Sun, and Junwei Lu. Generalizing Operational Transformation to the Standard General Markup Language. In *Conference on Computer Supported Cooperative Work (CSCW '02)*, pages 58–67, 2002.

[60] Atul Prakash and Michael J. Knister. A Framework for Undoing Actions in Collaborative Systems. *ACM Transactions on Computer-Human Interaction*, 1:295–330, 1994.

[61] Haifeng Shen and Chengzheng Sun. Flexible Merging for Asynchronous Collaborative Systems. In *Conference on Cooperative Information Systems (CoopIS '02)*, pages 304–321, 2002.

[62] Paul Dourish and Victoria Bellotti. Awareness and Coordination in Shared Workspaces. In *Conference on Computer Supported Cooperative Work (CSCW '92)*, pages 107–114, 1992.

[63] Carl Gutwin and Saul Greenberg. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *Computer Supported Cooperative Work*, 11:411–446, 2002.

[64] Codoxware: Connecting People and Documents. `http://www.codoxware.com/`, 2013.

[65] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The World-Wide Web. *Communications of the ACM*, 37:76–82, 1994.

[66] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifiers (URI): Generic Syntax (Request for Comments: 2396). `http://www.ietf.org/rfc/rfc2396.txt`, 1998.

[67] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1 (Request for Comments: 2616). http://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999.

[68] Julian Aubourg, Jungkee Song, and Hallvord R. M. Steen. XMLHttpRequest (W3C Working Draft 6). http://www.w3.org/TR/2012/WD-XMLHttpRequest-20121206/, 2012.

[69] Matthias Heinrich and Martin Gaedke. Data Binding for Standard-Based Web Applications. In *ACM Symposium on Applied Computing (SAC '12)*, pages 652–657, 2012.

[70] Matthias Heinrich and Martin Gaedke. WebSoDa: A Tailored Data Binding Framework for Web Programmers Leveraging the WebSocket Protocol and HTML5 Microdata. In *International Conference on Web Engineering (ICWE '11)*, pages 387–390, 2011.

[71] Ian Fette and Alexey Melnikov. The WebSocket Protocol (Request for Comments: 6455). http://tools.ietf.org/rfc/rfc6455.txt, 2011.

[72] Node.js. http://nodejs.org/, 2013.

[73] Jason Hill and Carl Gutwin. The MAUI Toolkit: Groupware Widgets for Group Awareness. *Computer Supported Cooperative Work*, 13:539–571, 2004.

[74] Mark Roseman and Saul Greenberg. Building Real-Time Groupware with GroupKit, a Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, 3:66–106, 1996.

[75] Anthony Baxter, Jochen Bekmann, Daniel Berlin, Soren Lassen, and Sam Thorogood. Google Wave Federation Protocol Over XMPP. http://www.waveprotocol.org/protocol/draft-protocol-specs/draft-protocol-spec, 2009.

[76] Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. http://xmpp.org/rfcs/rfc3920.txt, 2004.

[77] Marshall Rose. Post Office Protocol - Version 3. http://tools.ietf.org/html/rfc1081, 1988.

[78] Google Wave Gadgets API. http://www.waveprotocol.org/wave-apis/google-wave-gadgets-api, 2010.

[79] OpenSocial Specification 2.0. http://opensocial-resources.googlecode.com/svn/spec/2.0/OpenSocial-Specification.xml, 2011.

[80] James Gosling and Henry McGilton. *The Java Language Environment – A White Paper*. JavaSoft, 1996.

[81] JBuilder. `http://www.embarcadero.com/products/jbuilder`, 2013.

[82] Graham Hamilton. *The JavaBeans API Specification*. Sun Microsystems, 1997.

[83] Tcl Developer Site. `http://www.tcl.tk/`, 2013.

[84] Kai Lin, David Chen, Chengzheng Sun, and R. Geoff Dromey. Leveraging Single-User Microsoft Visio for Multi-User Real-Time Collaboration. In *Conference on Cooperative Design, Visualization, and Engineering (CDVE '07)*, pages 353–360, 2007.

[85] Agustina, Fei Liu, Steven Xia, Haifeng Shen, and Chengzheng Sun. CoMaya: Incorporating Advanced Collaboration Capabilities into 3D Digital Media Design Tools. In *Conference on Computer Supported Cooperative Work (CSCW '08)*, pages 5–8, 2008.

[86] James Begole, Mary Beth Rosson, and Clifford A. Shaffer. Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems. *ACM Transactions on Computer-Human Interaction*, 6:95–132, 1999.

[87] James Begole, Craig A. Struble, and Clifford A. Shaffer. Leveraging Java Applets: Toward Collaboration Transparency in Java. *IEEE Internet Computing*, 1:57–64, 1997.

[88] Dietwig Lowet and Daniel Görgen. Co-Browsing Dynamic Web Pages. In *International World Wide Web Conference (WWW '09)*, pages 941–950, 2009.

[89] Greasemonkey. `https://addons.mozilla.org/de/firefox/addon/greasemonkey/`, 2013.

[90] Marco Brambilla, Sara Comai, Piero Fraternali, and Maristella Matera. Designing Web Applications with Webml and Webratio. In *Web Engineering*, pages 221–261. 2008.

[91] Peter P. Chen. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[92] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide (2nd Edition)*. Addison-Wesley, 2005.

[93] ISO/IEC. ISO/IEC 19501: Unified Modeling Language Specification (Version 1.4.2), 2005.

[94] ISO/IEC. ISO/IEC 19503: XML Metadata Interchange Specification (Version 2.0.1), 2005.

[95] ISO/IEC. ISO/IEC 19502: Meta Object Facility (MOF) Specification (Version 1.4.1), 2005.

[96] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification (Version 1.1), 2011.

[97] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture – Practice and Promise*. Addison-Wesley, 2003.

[98] Alexander Knapp, Nora Koch, Gefei Zhang, and Hanns-Martin Hassler. Modeling Business Processes in Web Applications with ArgoUWE. In *International Conference on the Unified Modelling Language (UML '04)*, pages 69–83, 2004.

[99] ArgoUML. `http://www.argouml.org`, 2013.

[100] Daniel Schwabe, Rita de Almeida Pontes, and Isbela Moura. OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW. *ACM SigWEB Newsletter*, 8:18–34, 1999.

[101] Roberto Ierusalimschy. *Programming in Lua (Third Edition)*. Lua.org, 2013.

[102] Deborah J. Armstrong. The Quarks of Object-Oriented Development. *Communications of the ACM*, 49:123–128, 2006.

[103] Martin Gaedke, Christian Segor, and Hans-Werner Gellersen. WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. In *ACM Symposium on Applied Computing (SAC '00)*, pages 748–755, 2000.

[104] Matthias Heinrich, Franz Lehmann, Franz Josef Grüneberger, Martin Gaedke, Thomas Springer, and Alexander Schill. Enriching Single-User Web Applications Non-Invasively with Shared Editing Support. *Science of Computer Programming*, 2013.

[105] Matthias Heinrich, Franz Lehmann, Thomas Springer, and Martin Gaedke. Exploiting Single-User Web Applications for Shared Editing: a Generic Transformation Approach. In *International World Wide Web Conference (WWW '12)*, pages 1057–1066, 2012.

[106] Travis Leithead, Jacob Rossi, Doug Schepers, Björn Höhrmann, Philippe Le Hégaret, and Tom Pixley. Document Object Model (DOM) Level 3 Events Specification. `http://www.w3.org/TR/DOM-Level-3-Events/`, 2012.

[107] Joe Kesselman, Jonathan Robie, Mike Champion, Peter Sharpe, Vidur Apparao, and Lauren Wood. Document Object Model (DOM) Level 2 Traversal and Range Specification. `http://www.w3.org/TR/DOM-Level-2-Traversal-Range/`, 2000.

[108] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading Style Sheets Level 2 (CSS 2) Specification. `http://www.w3.org/TR/CSS2/`, 2011.

[109] Ace – The High Performance Code Editor for the Web. `http://ace.ajax.org/`, 2013.

[110] SproutCore. `http://sproutcore.com/`, 2013.

[111] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. John Wiley & Sons, 2007.

[112] Web Technology Surveys – Usage statistics and Market Share of jQuery for Websites. `http://w3techs.com/technologies/details/js-jquery/all/all`, 2013.

[113] Matthias Heinrich, Franz Lehmann, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Analyzing the Suitability of Web Applications for a Single-User to Multi-User Transformation. In *International World Wide Web Conference (WWW '13) – Companion Volume*, pages 249–252, 2013.

[114] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). `http://tools.ietf.org/html/rfc4627`, 2006.

[115] Welcome to CometD Project. `http://cometd.org/`, 2013.

[116] Michael Twidale, David Randall, and Richard Bentley. Situated Evaluation for Cooperative Systems. In *Conference on Computer Supported Cooperative Work (CSCW '94)*, pages 441–452, 1994.

[117] ISO/IEC. ISO/IEC 9126-1: Software engineering - Product quality - Part 1: Quality model, 2001.

[118] David Pinelle, Carl Gutwin, and Saul Greenberg. Task Analysis for Groupware Usability Evaluation: Modeling Shared-Workspace Tasks with the Mechanics of Collaboration. *ACM Transactions on Computer-Human Interaction*, 10:281–311, 2003.

[119] Nigel Bevan. Quality in Use: Meeting User Needs for Quality. *Journal of Systems and Software*, 49:89–96, 1999.

[120] Rensis Likert. A Technique for the Measurement of Attitudes. *Archives of Psychology*, 22:5–55, 1932.

[121] Adobe Flash Platform. `http://www.adobe.com/flashplatform/`, 2013.

[122] Microsoft Silverlight. `http://www.microsoft.com/silverlight/`, 2013.

[123] jQuery.sheet – The Web-Based Spreadsheet. `http://code.google.com/p/jquerysheet/`, 2013.

[124] Daniel C. Burnett, Adam Bergkvist, Cullen Jennings, and Anant Narayanan. Media Capture and Streams. `http://www.w3.org/TR/mediacapture-streams/`, 2013.

[125] Philipp Hauer. Optimierung eines generischen Synchronisationsmechanismus für kollaborative Echtzeitanwendungen im Web. Master's thesis, Chemnitz University of Technology, 2013.

[126] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5:63–108, 1998.

[127] Chengzheng Sun and David Chen. Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems. *ACM Transactions on Computer-Human Interaction*, 9:1–41, 2002.

[128] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Exploiting Annotations for the Rapid Development of Collaborative Web Applications. In *International World Wide Web Conference (WWW '13)*, pages 551–560, 2013.

[129] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Enriching Web Applications with Collaboration Support Using Dependency Injection. In *International Conference on Web Engineering (ICWE '12)*, pages 473–476, 2012.

[130] AngularJS. `http://angularjs.org/`, 2013.

[131] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[132] Michael Pizzo, Ralf Handl, and Martin Zurmuehl. OData Version 4.0 Part 1: Protocol. `http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html`, 2013.

[133] Tim Bray, Jean Paoli, Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0. `http://www.w3.org/TR/REC-xml/`, 2008.

[134] James Clark and Steve DeRose. XML Path Language (XPath). `http://www.w3.org/TR/xpath/`, 1999.

[135] ISO/IEC. ISO/IEC 14977 – Syntactic Metalanguage – Extended BNF, 1996.

[136] Knockout Downloads on GitHub. `https://github.com/knockout/knockout/downloads`, 2013.

[137] George Gaylord Simpson, Anne Roe, and Richard C. Lewontin. *Quantitative Zoology: Revised Edition*. Dover Books on Biology, Psychology and Medicine. Dover Publications, 2003.

[138] GitHub. `https://github.com/`, 2013.

[139] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, and Martin Gaedke. Reusable Awareness Widgets for Collaborative Web Applications – A Non-invasive Approach. In *International Conference on Web Engineering (ICWE '12)*, pages 1–15, 2012.

[140] Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, Philipp Hauer, and Martin Gaedke. GAwI: A Comprehensive Workspace Awareness Library for Collaborative Web Applications. In *International Conference on Web Engineering (ICWE '13)*, pages 482–485, 2013.

[141] Aryeh Gregor. HTML Editing APIs. `https://dvcs.w3.org/hg/editing/raw-file/tip/editing.html`, 2012.

[142] html2canvas - Screenshots with JavaScript. `http://html2canvas.hertzen.com/`, 2013.

[143] Eevi E. Beck. A Survey of Experiences of Collaborative Writing. *Computer Supported Collaborative Writing*, pages 87–112, 1993.

[144] Deborah G. Tatar, Gregg Foster, and Daniel G. Bobrow. Design for Conversation: Lessons from Cognoter. *International Journal of Man-Machine Studies*, 34:185–209, 1991.

[145] Joseph Edward McGrath. *Groups: Interaction and Performance*. Prentice-Hall, 1984.

[146] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, and Anant Narayanan. WebRTC 1.0: Real-time Communication Between

Browsers. http://dev.w3.org/2011/webrtc/editor/webrtc.html, 2013.

Appendix A

DOM Adapter Evaluation Material

A.1 Tutorials

A.1.1 SVG-edit Tutorial

1. Open SVG-edit
   (a) Click on the `SVG-edit` bookmark.
2. Insert Objects
   (a) Select the `Rectangle` tool from the toolbar.
   (b) Click on the canvas and move your cursor keeping the left mouse button pressed.
   (c) Press `Shift` to draw a square.
   (d) Release the left mouse button.
3. Turn Objects
   (a) Select the square by clicking on it (additional tools will appear in the toolbar at the top).
   (b) Turn the square by dragging and dropping the small green handle.
4. Draw Polygons
   (a) Select the `Path` tool from the left toolbar.
   (b) Click on the canvas.
   (c) To create a polygon, add path corners by clicking at various canvas positions.
   (d) Finish the polygon by double-clicking on the canvas.
5. Change Objects' Stroke Color
   (a) Select the polygon by clicking on it.

   (b) Change the stroke color using the color picker from the toolbar at the bottom.

6. Move Objects
   (a) Drag the created polygon and drop it on top of the square.
7. Change Objects' Fill Color
   (a) Select the polygon.
   (b) Change the fill color using the color picker from the toolbar at the bottom.
8. Draw Freehand Lines
   (a) Select the `Pencil` tool from the left toolbar.
   (b) Draw a freehand line on the canvas.
9. Clone and Delete Objects
   (a) Select the freehand line.
   (b) Clone the freehand line using the `Clone Element` tool from the toolbar at the top.
   (c) Delete the freehand line selecting the `Delete Element` tool in the toolbar at the top.
10. Delete all Objects
   (a) Select all objects using the `Select` tool and afterwards drawing a selection box.
   (b) Delete the selected objects by using the `Delete Element` tool or by pressing `Delete`.

A.1.2 CKEditor Tutorial

1. Open CKEditor
   (a) Click on the `CKEditor` bookmark.
2. Insert Text
   (a) Click on the document canvas to set the cursor at the desired position.
   (b) Write text by typing characters.
   (c) Insert a new line by pressing `Enter`.
3. Style Text
   (a) Select some characters of the text.
   (b) Click on the `Bold` tool to highlight the selected text.
   (c) Change the background and text color of the selected text using the `Text Color` and `Background Color` tools.
   (d) Re-select a part of the styled text and use the `Remove Format` tool to remove the styling.

4. Insert Lists

   (a) Move the caret to the desired position by clicking on the document canvas.

   (b) Use one of the `List` tools to insert a list element.

   (c) Insert a new line after each list entry to extend the list.

5. Insert Tables

   (a) Move the caret to the desired position by clicking on the document canvas.

   (b) Use the `Table` tool to insert a new table.

6. Extend Tables

   (a) Insert text into table cells.

   (b) Do a right mouse-click in one of the table cells.

   (c) Select `Row > Insert Row after` and a new row will appear.

A.2 Tasks

To furnish a convenient work atmosphere, your manager wants you and one of your colleagues to make suggestions for improvements. Therefore, you have to present the current office setting and propose a new improved office setting leveraging the two collaborative web-based tools SVG-edit and CKEditor.

A.2.1 Shared Editing Task A: Draw and Enrich your Office Using the Collaborative SVG-edit

1. Draw the office floor plan.

2. Add existing inventory items (desks, chairs, PCs, etc.) as detailed as possible and save this graphic.

3. Freely rearrange inventory items in your office to improve the work atmosphere.

4. Add inventory items (e.g. plants) to improve the work atmosphere and save this graphic.

A.2.2 Shared Editing Task B: Write a Letter Capturing Proposals Using the Collaborative CKEditor

1. Compile all inventory items that are currently available in a list or table representation and estimate the purchase price, maintenance costs as well as the lifespan.

2. Add proposed inventory items and estimate the required information regarding the purchase price, maintenance costs as well as the lifespan.

3. To inform your manager about the proposals, rearrange your document to create a letter in an appealing format.

A.3 Questionnaire

	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree
Functionality							
Q1) Overall, the editor supported me completing the task in a collaborative way.	☐	☐	☐	☐	☐	☐	☐
Reliability							
Q2) The editor's resynchronization mechanism did not significantly hinder my work.	☐	☐	☐	☐	☐	☐	☐
Q3) Required browser page reloads did not significantly hinder my work.	☐	☐	☐	☐	☐	☐	☐
Q4) Whenever an error occurred, I was able to recover easily.	☐	☐	☐	☐	☐	☐	☐
Q5) I am satisfied how fast the system recovered after a browser page reload.	☐	☐	☐	☐	☐	☐	☐
Usability							
Q6) The editor's collaboration feature was easy to learn.	☐	☐	☐	☐	☐	☐	☐
Q7) Overall, the editor's collaboration feature was easy to use.	☐	☐	☐	☐	☐	☐	☐
Q8) Actions by other participants did not significantly hinder my work.	☐	☐	☐	☐	☐	☐	☐
Q9) Overall, working collaboratively with the editor was an enjoyable experience.	☐	☐	☐	☐	☐	☐	☐
Efficiency							
Q10) I am satisfied with how fast the editor responds to my input.	☐	☐	☐	☐	☐	☐	☐
Q11) I am satisfied with how fast changes by others were displayed on my monitor.	☐	☐	☐	☐	☐	☐	☐
Q12) It was appealing how the editor could quickly synchronize the document among all participants.	☐	☐	☐	☐	☐	☐	☐
Communication							
Q13) I could easily recognize the presence of other participants.	☐	☐	☐	☐	☐	☐	☐
Q14) I could easily recognize changes made to document objects.	☐	☐	☐	☐	☐	☐	☐
Q15) I could easily recognize actions of other participants.	☐	☐	☐	☐	☐	☐	☐
Q16) It was appealing how the editor highlighted the work of other participants.	☐	☐	☐	☐	☐	☐	☐
Coordination							
Q17) I could easily start editing any document object at any time.	☐	☐	☐	☐	☐	☐	☐
Q18) I could easily edit any document object as long as I desired.	☐	☐	☐	☐	☐	☐	☐
Q19) The editor preserved changes, I made to the document.	☐	☐	☐	☐	☐	☐	☐
Q20) I could easily start using any tool at any time.	☐	☐	☐	☐	☐	☐	☐
Q21) I could easily use any tool as long as I desired.	☐	☐	☐	☐	☐	☐	☐
Q22) It was appealing how I could collaboratively edit the document.	☐	☐	☐	☐	☐	☐	☐
	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree

A.4 Results

A.4.1 SVG-edit Questionnaire Data

This data was collected from 30 completed questionnaires. Note that questions marked with an asterisk (*) were not answered correctly by all study participants and thus, the number of collected data sets does not add up to 30.

	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree
Functionality							
Q1) Overall, the editor supported me completing the task in a collaborative way.	0	3	1	1	4	11	10
Reliability							
Q2)* The editor's resynchronization mechanism did not significantly hinder my work.	1	0	1	5	5	5	4
Q3)* Required browser page reloads did not significantly hinder my work.	2	4	2	5	3	7	4
Q4)* Whenever an error occurred, I was able to recover easily.	2	0	3	5	3	8	5
Q5)* I am satisfied how fast the system recovered after a browser page reload.	1	0	2	5	2	10	6
Usability							
Q6) The editor's collaboration feature was easy to learn.	0	0	0	0	3	4	23
Q7) Overall, the editor's collaboration feature was easy to use.	0	0	0	1	5	11	13
Q8) Actions by other participants did not significantly hinder my work.	0	1	3	2	9	8	7
Q9) Overall, working collaboratively with the editor was an enjoyable experience.	0	0	1	0	3	10	16
Efficiency							
Q10) I am satisfied with how fast the editor responds to my input.	0	0	3	2	3	10	12
Q11) I am satisfied with how fast changes by others were displayed on my monitor.	1	2	1	5	3	9	9
Q12) It was appealing how the editor could quickly synchronize the document among all participants.	1	3	1	1	4	14	6
Communication							
Q13) I could easily recognize the presence of other participants.	2	6	4	7	3	2	6
Q14) I could easily recognize changes made to document objects.	1	0	3	4	7	8	7
Q15) I could easily recognize actions of other participants.	1	4	4	6	4	8	3
Q16)* It was appealing how the editor highlighted the work of other participants.	4	6	3	4	1	4	1
Coordination							
Q17) I could easily start editing any document object at any time.	1	1	2	5	6	6	9
Q18) I could easily edit any document object as long as I desired.	0	2	2	3	7	7	9
Q19) The editor preserved changes, I made to the document.	1	0	2	4	3	5	15
Q20) I could easily start using any tool at any time.	0	0	0	3	3	13	11
Q21) I could easily use any tool as long as I desired.	0	0	1	2	3	12	12
Q22) It was appealing how I could collaboratively edit the document.	0	2	1	0	4	15	8
	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree

A.4.2 CKEditor Questionnaire Data

This data was collected from 30 completed questionnaires. Note that questions marked with an asterisk (*) were not answered correctly by all study participants and thus, the number of collected data sets does not add up to 30.

	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree
Functionality							
Q1) Overall, the editor supported me completing the task in a collaborative way.	0	0	0	3	4	10	13
Reliability							
Q2)* The editor's resynchronization mechanism did not significantly hinder my work.	0	3	2	5	3	5	4
Q3)* Required browser page reloads did not significantly hinder my work.	1	1	0	5	7	2	7
Q4)* Whenever an error occurred, I was able to recover easily.	0	6	1	5	0	4	7
Q5)* I am satisfied how fast the system recovered after a browser page reload.	1	0	0	3	5	8	4
Usability							
Q6) The editor's collaboration feature was easy to learn.	0	0	0	1	5	7	17
Q7) Overall, the editor's collaboration feature was easy to use.	0	0	1	2	4	7	16
Q8) Actions by other participants did not significantly hinder my work.	0	2	3	4	5	10	6
Q9) Overall, working collaboratively with the editor was an enjoyable experience.	0	1	1	3	4	8	13
Efficiency							
Q10) I am satisfied with how fast the editor responds to my input.	0	2	0	3	3	9	13
Q11) I am satisfied with how fast changes by others were displayed on my monitor.	1	2	2	1	4	12	8
Q12) It was appealing how the editor could quickly synchronize the document among all participants.	0	3	0	2	4	14	7
Communication							
Q13) I could easily recognize the presence of other participants.	0	1	1	4	7	7	10
Q14) I could easily recognize changes made to document objects.	0	0	2	2	4	10	12
Q15) I could easily recognize actions of other participants.	0	1	0	0	12	6	11
Q16)* It was appealing how the editor highlighted the work of other participants.	0	2	1	4	7	7	8
Coordination							
Q17) I could easily start editing any document object at any time.	0	0	1	1	7	10	11
Q18) I could easily edit any document object as long as I desired.	0	0	1	2	6	7	14
Q19) The editor preserved changes, I made to the document.	0	0	0	6	5	10	9
Q20) I could easily start using any tool at any time.	0	1	1	1	2	7	18
Q21) I could easily use any tool as long as I desired.	0	0	1	0	4	8	17
Q22) It was appealing how I could collaboratively edit the document.	0	2	1	3	4	7	13
	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree

Appendix B

Framework Adapter Evaluation Material

B.1 Questionnaire

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
Functional Suitability					
Q1) The Gravity API/the KCA provides the necessary functionality to develop collaborative applications.	☐	☐	☐	☐	☐
Q2) The functionality of the Gravity API/the KCA eases the development of collaborative applications.	☐	☐	☐	☐	☐
Compatibility					
Q3) The Gravity API/the KCA restricted the choice of other technologies.	☐	☐	☐	☐	☐
Q4) Other selected technologies worked well with Gravity/the KCA.	☐	☐	☐	☐	☐
Usability					
Q5) I could easily learn how to use the Gravity API/the KCA.	☐	☐	☐	☐	☐
Q6) After the initial learning phase, the Gravity API/the KCA was easy to use.	☐	☐	☐	☐	☐
Q7) The Gravity API/the KCA prevented me from making mistakes during the development of the collaborative web application.	☐	☐	☐	☐	☐
Maintainability					
Q8) The Gravity API/the KCA helped me to separate sync code from the rest of the application.	☐	☐	☐	☐	☐
Q9) The impact of changes made to the application's source code could be foreseen easily.	☐	☐	☐	☐	☐
Q10) The cause of failures could be reconstructed with modest effort.	☐	☐	☐	☐	☐
Q11) Parts of a developed application that have to be modified could be easily determined.	☐	☐	☐	☐	☐
Q12) The developed application could be easily enriched with additional application features without introducing defects.	☐	☐	☐	☐	☐
Satisfaction					
Q13) The Gravity API/the KCA is useful to develop collaborative applications.	☐	☐	☐	☐	☐
Q14) The Gravity API/the source code annotations lead to the expected behavior.	☐	☐	☐	☐	☐
Q15) The callback mechanism/the property exclusion mechanism is convenient.	☐	☐	☐	☐	☐
Q16) It is very easy to use the Gravity API/the KCA to add collaboration capabilities to an application.	☐	☐	☐	☐	☐
Q17) The Gravity API/the KCA is a comfortable means for developing real-time applications.	☐	☐	☐	☐	☐
	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree

B.2 Results

B.2.1 KCA Questionnaire Data

This data was collected from 8 completed questionnaires.

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
Functional Suitability					
Q1) The KCA provides the necessary functionality to develop collaborative applications.	0	0	1	2	5
Q2) The functionality of the the KCA eases the development of collaborative applications.	0	0	1	2	5
Compatibility					
Q3) The KCA restricted the choice of other technologies.	1	4	3	0	0
Q4) Other selected technologies worked well with the KCA.	0	0	2	4	2
Usability					
Q5) I could easily learn how to use the KCA.	0	1	2	4	1
Q6) After the initial learning phase, the KCA was easy to use.	0	1	0	5	2
Q7) The KCA prevented me from making mistakes during the development of the collaborative web application.	0	2	2	4	0
Maintainability					
Q8) The KCA helped me to separate sync code from the rest of the application.	0	0	2	5	1
Q9) The impact of changes made to the application's source code could be foreseen easily.	0	2	2	3	1
Q10) The cause of failures could be reconstructed with modest effort.	0	2	4	2	0
Q11) Parts of a developed application that have to be modified could be easily determined.	0	1	3	2	2
Q12) The developed application could be easily enriched with additional application features without introducing defects.	1	1	2	2	2
Satisfaction					
Q13) The KCA is useful to develop collaborative applications.	0	0	1	3	4
Q14) The source code annotations lead to the expected behavior.	1	0	2	2	3
Q15) The property exclusion mechanism is convenient.	0	1	3	1	3
Q16) It is very easy to use the KCA to add collaboration capabilities to an application.	0	1	1	3	3
Q17) The KCA is a comfortable means for developing real-time applications.	0	0	2	1	5

| | Strongly Disagree | Disagree | Neither Agree nor Disagree | Agree | Strongly Agree |

B.2.2 Gravity Questionnaire Data

This data was collected from 8 completed questionnaires.

	Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
Functional Suitability					
Q1) The Gravity API provides the necessary functionality to develop collaborative applications.	0	1	1	4	2
Q2) The functionality of the Gravity API eases the development of collaborative applications.	0	3	2	1	2
Compatibility					
Q3) The Gravity API restricted the choice of other technologies.	1	3	4	0	0
Q4) Other selected technologies worked well with Gravity.	0	2	3	2	1
Usability					
Q5) I could easily learn how to use the Gravity API.	1	1	4	2	0
Q6) After the initial learning phase, the Gravity API was easy to use.	0	1	2	4	1
Q7) The Gravity API prevented me from making mistakes during the development of the collaborative web application.	1	3	2	2	0
Maintainability					
Q8) The Gravity API helped me to separate sync code from the rest of the application.	0	5	1	2	0
Q9) The impact of changes made to the application's source code could be foreseen easily.	1	3	1	2	1
Q10) The cause of failures could be reconstructed with modest effort.	2	3	2	1	0
Q11) Parts of a developed application that have to be modified could be easily determined.	0	2	4	2	0
Q12) The developed application could be easily enriched with additional application features without introducing defects.	0	3	1	4	0
Satisfaction					
Q13) The Gravity API is useful to develop collaborative applications.	0	2	1	2	3
Q14) The Gravity API lead to the expected behavior.	0	2	0	5	1
Q15) The callback mechanism is convenient.	0	2	0	6	0
Q16) It is very easy to use the Gravity API to add collaboration capabilities to an application.	1	3	2	2	0
Q17) The Gravity API is a comfortable means for developing real-time applications.	1	3	2	0	2

| | Strongly Disagree | Disagree | Neither Agree nor Disagree | Agree | Strongly Agree |

Appendix C

Workspace Awareness Adapter Evaluation Material

C.1 Tutorials

C.1.1 SVG-edit Tutorial

1. Start SVG-edit
   (a) Start SVG-edit by clicking on the bookmark `SVG-edit (Tutorial)` in your browser.
2. Create Rectangles / Ellipses
   (a) Select the tool `Rectangle` or `Ellipse` from the toolbar on the left hand side.
   (b) Click on the editor's drawing canvas and drag a rectangle/ellipse while keeping the mouse button pressed.
   (c) To create a square or a circle press the `Shift` key additionally.
   (d) To finally create the element, release your mouse button.
3. Create Lines
   (a) Select the tool `Line` from the toolbar on the left hand side.
   (b) Click on the editor's drawing canvas and drag a line while keeping the mouse button pressed.
4. Create Freehand Lines
   (a) Select the tool `Pencil` from the toolbar on the left hand side.
   (b) Draw a freehand line by clicking somewhere on the drawing canvas, moving your mouse around and finally releasing the mouse button.

5. Create Shapes
   (a) Select the tool `Path` from the toolbar on the left hand side.
   (b) Click on the editor's drawing canvas to start your shape.
   (c) Expand your shape by adding additional corner points via mouse click.
   (d) Complete your shape by double clicking somewhere on the canvas.
6. Select Objects
   (a) Select the tool `Select` from the toolbar on the left hand side.
   (b) Click on the object you want to select.
   (c) You can select multiple elements by dragging a selection box around all elements to select.
7. Rotate Objects
   (a) Select the object you want to rotate (additional tools will appear in the upper toolbar).
   (b) Rotate the object by dragging the small green handle.
8. Change Stroke Color and Thickness
   (a) Select the object you want to modify.
   (b) Change the stroke color using the stroke color picker from the lower toolbar.
   (c) Change the stroke thickness by modifying the thickness value.
9. Change Fill Color and Opacity
   (a) Select the object you want to modify.
   (b) Change the fill color using the fill color picker from the lower toolbar.
   (c) Change the opacity by modifying the opacity value.
10. Clone Objects
    (a) Select the object you want to clone.
    (b) Clone the object using `Clone Element` tool from the upper toolbar.
11. Delete Objects
    (a) Select the object you want to delete.
    (b) Delete the object using the `Delete Element` tool from the upper toolbar or by pressing the `Delete` key.
12. Change the Visibility of Awareness Widgets
    (a) Click on the awareness widget configuration button.
    (b) In the appearing menu you can toggle the visibility of the awareness widgets by pressing the button next to the widget name.
    (c) To finalize the configuration click on the awareness widget configuration button again.

13. Move Awareness Widgets
    (a) Click on the awareness widget move and resize button.
    (b) Move a widget around using the awareness widget handle.
    (c) To stop moving widgets around click on the widget move and resize button again.
14. Resize Awareness Widgets
    (a) Click on the awareness widget move and resize button.
    (b) Resize a widget using the resize handle at the lower right corner.
    (c) To stop resizing widgets click on the widget move and resize button again.

C.1.2 CKEditor Tutorial

1. CKEditor Start
    (a) Start the CKEditor by clicking on the bookmark `CKEditor (Tutorial)` in your browser.
2. Insert Text
    (a) Click on the document canvas to set the writing caret to a specific position.
    (b) Insert some text by typing characters.
    (c) Insert a new line by pressing the enter key.
3. Delete Text
    (a) Click on the document canvas to set the writing caret to a specific position.
    (b) Delete some text using the backspace or delete key.
4. Format Text
    (a) Select some text.
    (b) Change the style of the text using the style buttons from the toolbar.
    (c) Change the alignment using the text orientation buttons.
    (d) Change the color of the text using the text color picker.
    (e) Change the background color of the text using the background color picker.
    (f) Change the paragraph format using the paragraph formatter.
    (g) Change the font using the font selector.
    (h) Change the font size by means of the font size selector.
    (i) Remove formats leveraging the `Remove Format` button.

5. Create Lists
   (a) Click on the document canvas to set the writing caret to the desired position.
   (b) Use the list tools to begin an ordered or unordered list.
   (c) Add a new list entry by pressing the enter key at the end of an existing entry.
6. Insert Tables
   (a) Click on the document canvas to determine the insertion position for the table.
   (b) Click on the tool `Table` in the toolbar.
   (c) Use the table creation dialog to specify the amount of rows and columns as well as the width of the table.
   (d) To insert and delete rows or columns right click on a table cell and use the context menu entries.
7. Insert Images
   (a) Click on the document canvas to determine the insertion position for the table.
   (b) Click on the tool `Image` in the toolbar.
   (c) Use the image insertion dialog to insert an image.
8. Insert Separators
   (a) Click on the document canvas to determine the insertion position.
   (b) Click on the tool `Horizontal Line` to insert the separator.
9. Change the Visibility of Awareness Widgets
   (a) Click on the awareness widget configuration button.
   (b) In the appearing menu you can toggle the visibility of the awareness widgets by pressing the button next to the widget name.
   (c) To finalize the configuration click on the awareness widget configuration button again.
10. Move Awareness Widgets
    (a) Click on the awareness widget move and resize button.
    (b) Move a widget around using the awareness widget handle.
    (c) To stop moving widgets around click on the widget move and resize button again.
11. Resize Awareness Widgets
    (a) Click on the awareness widget move and resize button.
    (b) Resize a widget using the resize handle at the lower right corner.
    (c) To stop resizing widgets click on the widget move and resize button again.

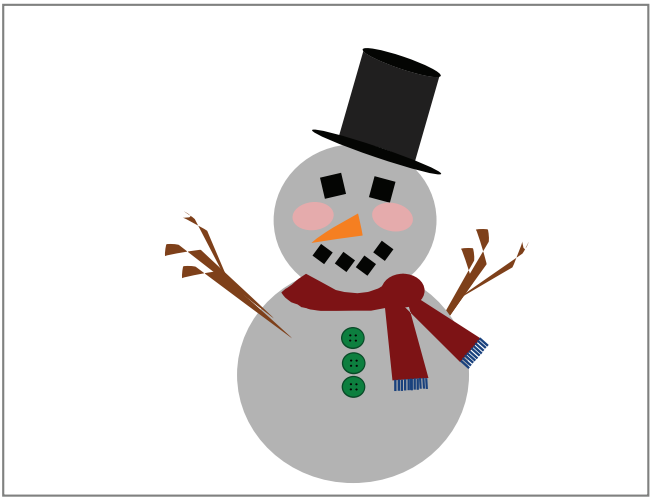
C.2 Tasks

C.2.1 Shared Editing Task Using SVG-edit

Assume you are a member of the organization team for the company's Christmas party. You agreed to send out invitation cards with a snowman graphic. Since the available snowman graphics in the Internet are not free of charge, a colleague started to create a snowman graphic. However, some parts of the graphic are still missing while others are available but not at the right position. In order to help your colleague finishing the snowman graphic, you have to do the following tasks with your partner using the collaborative SVG-edit:

1. Assemble the already available pieces to form the snowman graphic as depicted below.
2. Complete the snowman graphic by drawing the missing parts on your own.
3. Draw a Christmas tree to enhance the snowman graphic.

Please be as accurate as possible when constructing the snowman graphic. Moreover, be creative authoring the Christmas tree.



C.2.2 Shared Editing Task Using CKEditor

Assume you are a member of the organization team for the company's Christmas party. You agreed to send out invitation cards with a snowman graphic. Your colleagues already created a nice snowman image. Now, you and your partner are in charge of creating the actual invitation letter. Therefore, you have to accomplish the following tasks using the collaborative CKEditor:

1. Write and style the invitation letter as depicted below. The snowman image can be found here: `http://10.55.8.184:1081/CKEditor/snowman.png`.
2. If you have finished task 1, continue by listing the tasks that have to be done in preparation for the party accompanied by the responsible person and the estimated costs.

Please be as accurate as possible when writing the invitation letter. Moreover, be creative authoring the task list.

Official Letter of Invitation

On behalf of this year's christmas party organization team, we take great pleasure in inviting you to attend this year's christmas party at **December 11th, 2012** in **Dresden** starting at **5 pm**.

Attached you will find a short invitation card summarizing the main facts of the party.

We are looking forward to your attendence and participation.

Cheers,

the christmas party organization team

Invitation Card

When	Tuesday, December 11th, 2012
Where	Dresden
Time	5 pm
Why	Because it is almost Christmas ...

C.3 Questionnaire

	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree
Functionality							
Q1) It was easy to recognize the presence and identity of other participants.	☐	☐	☐	☐	☐	☐	☐
Q2) It was easy to recognize who changed a document artifact.	☐	☐	☐	☐	☐	☐	☐
Q3) I could easily recognize what other participants were doing.	☐	☐	☐	☐	☐	☐	☐
Q4) I could easily recognize others' intentions.	☐	☐	☐	☐	☐	☐	☐
Q5) I could easily identify the artifacts others were working on.	☐	☐	☐	☐	☐	☐	☐
Q6) It was easy to identify where others were working.	☐	☐	☐	☐	☐	☐	☐
Q7) It was easy to identify which document artifacts are visible to others.	☐	☐	☐	☐	☐	☐	☐
Reliability							
Q8) The awareness features operated as expected.	☐	☐	☐	☐	☐	☐	☐
Usability							
Q9) The editor's awareness features were easy to learn.	☐	☐	☐	☐	☐	☐	☐
Q10) After the initial learning phase the awareness features were easy to use.	☐	☐	☐	☐	☐	☐	☐
Q11) The awareness features decently supported my work.	☐	☐	☐	☐	☐	☐	☐
Q12) All in all, it was an enjoyable experience to work with the collaborative editors.	☐	☐	☐	☐	☐	☐	☐
Efficiency							
Q13) Overall, the awareness features are valuable to complete collaborative tasks.	☐	☐	☐	☐	☐	☐	☐
Communication							
Q14) Overall, it was easy to follow others' actions in the workspace.	☐	☐	☐	☐	☐	☐	☐
Q15) It was easy to gesture and refer to items in the workspace.	☐	☐	☐	☐	☐	☐	☐
Q16) It was easy to see and understand what others were pointing to.	☐	☐	☐	☐	☐	☐	☐
Coordination							
Q17) I could easily modify any document artifact at any time.	☐	☐	☐	☐	☐	☐	☐
Q18) It was easy to avoid conflicts with other participants.	☐	☐	☐	☐	☐	☐	☐
Q19) It was easy to avoid the duplication of actions during the task.	☐	☐	☐	☐	☐	☐	☐
Q20) I could easily follow other participant's actions.	☐	☐	☐	☐	☐	☐	☐
Q21) I was able to assist others when they needed help.	☐	☐	☐	☐	☐	☐	☐
Q22) I did not accidentally alter or destruct others' work.	☐	☐	☐	☐	☐	☐	☐
	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree

C.4 Results

C.4.1 SVG-edit Questionnaire Data

This data was collected from 20 completed questionnaires.

	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree
Functionality							
Q1) It was easy to recognize the presence and identity of other participants.	0	0	0	1	2	6	11
Q2) It was easy to recognize who changed a document artifact.	0	0	3	1	1	7	8
Q3) I could easily recognize what other participants were doing.	0	0	2	3	2	4	9
Q4) I could easily recognize others' intentions.	1	1	0	4	4	6	4
Q5) I could easily identify the artifacts others were working on.	0	0	2	0	3	9	6
Q6) It was easy to identify where others were working.	0	0	0	0	1	10	9
Q7) It was easy to identify which document artifacts are visible to others.	0	2	0	3	2	6	7
Reliability							
Q8) The awareness features operated as expected.	0	0	0	1	2	9	8
Usability							
Q9) The editor's awareness features were easy to learn.	0	0	0	0	1	3	16
Q10) After the initial learning phase the awareness features were easy to use.	0	0	0	0	0	4	16
Q11) The awareness features decently supported my work.	0	0	0	0	0	12	8
Q12) All in all, it was an enjoyable experience to work with the collaborative editors.	0	0	0	1	0	6	13
Efficiency							
Q13) Overall, the awareness features are valuable to complete collaborative tasks.	0	0	0	0	0	9	11
Communication							
Q14) Overall, it was easy to follow others' actions in the workspace.	0	0	0	1	6	8	5
Q15) It was easy to gesture and refer to items in the workspace.	0	0	0	3	6	6	5
Q16) It was easy to see and understand what others were pointing to.	0	0	1	1	5	8	5
Coordination							
Q17) I could easily modify any document artifact at any time.	0	0	1	0	5	8	6
Q18) It was easy to avoid conflicts with other participants.	0	0	1	3	6	4	6
Q19) It was easy to avoid the duplication of actions during the task.	0	0	0	2	6	8	4
Q20) I could easily follow other participant's actions.	0	0	0	1	4	10	5
Q21) I was able to assist others when they needed help.	0	1	0	2	4	7	6
Q22) I did not accidentally alter or destruct others' work.	0	0	2	1	2	7	8
	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree

C.4.2 CKEditor Questionnaire Data

This data was collected from 20 completed questionnaires.

	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree
Functionality							
Q1) It was easy to recognize the presence and identity of other participants.	0	0	0	1	5	2	12
Q2) It was easy to recognize who changed a document artifact.	0	0	0	2	4	6	8
Q3) I could easily recognize what other participants were doing.	0	0	0	0	4	9	7
Q4) I could easily recognize others' intentions.	0	1	2	3	6	4	4
Q5) I could easily identify the artifacts others were working on.	0	0	0	1	3	7	9
Q6) It was easy to identify where others were working.	0	0	0	0	2	6	12
Q7) It was easy to identify which document artifacts are visible to others.	0	0	1	3	5	3	8
Reliability							
Q8) The awareness features operated as expected.	0	0	0	1	4	10	5
Usability							
Q9) The editor's awareness features were easy to learn.	0	0	0	0	0	3	17
Q10) After the initial learning phase the awareness features were easy to use.	0	0	0	0	3	4	13
Q11) The awareness features decently supported my work.	0	0	0	1	2	8	9
Q12) All in all, it was an enjoyable experience to work with the collaborative editors.	0	0	0	1	3	5	11
Efficiency							
Q13) Overall, the awareness features are valuable to complete collaborative tasks.	0	0	0	0	1	10	9
Communication							
Q14) Overall, it was easy to follow others' actions in the workspace.	0	0	0	1	5	11	3
Q15) It was easy to gesture and refer to items in the workspace.	0	0	0	2	5	6	7
Q16) It was easy to see and understand what others were pointing to.	0	0	0	3	1	10	6
Coordination							
Q17) I could easily modify any document artifact at any time.	0	0	0	1	6	3	10
Q18) It was easy to avoid conflicts with other participants.	0	0	0	7	3	7	3
Q19) It was easy to avoid the duplication of actions during the task.	0	0	0	4	3	9	4
Q20) I could easily follow other participant's actions.	0	0	0	1	2	9	8
Q21) I was able to assist others when they needed help.	0	0	0	2	4	2	12
Q22) I did not accidentally alter or destruct others' work.	0	0	0	1	3	5	11
	Strongly Disagree	Disagree	Disagree Somewhat	Neither Agree nor Disagree	Agree Somewhat	Agree	Strongly Agree

Doctoral Dissertations in Web Engineering and Web Science

(1) Heinrich, Matthias (2014)
Enriching Web Applications Efficiently with Real-Time Collaboration Capabilities
ISBN 978-3-944640-25-9
http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-149948