
Post-Desktop Interaction for Web Applications

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von Dipl. Inform. Daniel Schreiber aus Aachen
August 2011 — Darmstadt — D 17



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Telekooperation

Post-Desktop Interaction
for Web Applications

Genehmigte Dissertation von Dipl. Inform. Daniel Schreiber aus Aachen

1. Gutachten: Prof. Dr. Max Mühlhäuser
2. Gutachten: Prof. Dr. Kris Luyten

Tag der Einreichung: 05.04.2011

Tag der Prüfung: 19.04.2011

Darmstadt — D 17

Acknowledgements

First and foremost, I thank my advisor Max Mühlhäuser for his continuous support, excellent advice and his faith in my work. I am grateful to Kris Luyten for acting as a second referee and also for supporting the cooperation between me and his group. It was a special pleasure to be able to work with Geert Vanderhulst. It's always good to see that other people around the world deal with similar problems.

I thank all my colleagues in the TK and RBG group with whom I had the pleasure of working over the years. I'd like to express special thanks to my dear colleague and friend Melanie Hartmann. She was never short of encouraging words, thoughtful remarks and red ink — each applied in the right dose and at the right moment. Without Erwin Aitenbichler and his MundoCore middleware this work would not have been possible. I thank all my other colleagues and former colleagues for their support, proofreading of drafts, and tolerance of my musical tastes (the latter is especially true for Tobias Klug, Fernando Lyardet, Jürgen Steimle, Markus Weimer, Marcus Ständer, Syed Zahid Ali and Markus Miche who had the dubious pleasure of sharing an office with me). I thank my students Andreas Goeb, Felix Heinrichs, Andreas Richter, Michael Zinn who helped implement and evaluate parts of MundoWeb. Discussions with you were always fruitful, in the rare cases we both managed to be there for an appointment on time. I thank our collaborators in the AUGUR project at SAP Darmstadt, Manuel Görtz, Andreas Faatz and the rest of the SAP CEC Darmstadt Crew. Further, I would like to acknowledge Thomas Kühne and Louenas Hamdi. It was a pleasure to work with you.

I am deeply grateful for the support of my parents Anne and Heinz and my sister Christine. I promise I will call more often, now that the thesis is completed. Finally, I thank my wife Julia for organizing a wedding while I was busy doing research, and for enduring me in the stressful times of writing this thesis.

Ehrenwörtliche Erklärung ²

Hiermit erkläre ich, die vorgelegte Arbeit zur Erlangung des akademischen Grades “Dr. Ing.” mit dem Titel “Post-desktop Interaction for Web Applications” selbständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben. Ich habe bisher noch keinen Promotionsversuch unternommen.

Darmstadt, den 05.04.2011

Daniel Schreiber

² Gemäß §9 Abs. 1 der Promotionsordnung der TU Darmstadt

Zusammenfassung

Anwendungsprogramme ‘verlassen’ zunehmend den Schreibtisch und werden stattdessen in einer Vielzahl unterschiedlicher Umgebungen eingesetzt. Bei der Entwicklung einer Anwendung ist es unmöglich vorherzusehen in welchen Umgebungen sie einmal verwendet werden wird. Ein gutes Beispiel hierfür liefern Web Anwendungen. Auf sie greift der Nutzer mit unterschiedlichen Browsern zu. Die Umgebungen aus denen auf Anwendungen zugegriffen wird, sind zunehmend keine typischen Schreibtisch Umgebungen mehr, sondern sogenannte *post-desktop* Umgebungen. In ihnen werden viele verschiedene Interaktionsgeräte jenseits von Tastatur, Maus und Bildschirm verwendet und sogar Föderationen aus mehreren Geräten zur Interaktion mit einer einzigen Anwendung. *Post-desktop* Umgebungen verfügen in der Regel nicht über eine schnelle Netzwerkverbindung, so dass der Nutzer störende Latenzzeiten bei der Interaktion mit Anwendungen in Kauf nehmen muss.

Die vorliegende Arbeit befasst sich mit der System Infrastruktur, die notwendig ist, um sicherzustellen, dass Anwendungen komfortabel in beliebigen *post-desktop* Umgebungen verwendet werden können. Dabei ist das Ziel, die vorhandene System Infrastruktur des Webs zu erweitern. Im einzelnen heißt das, es müssen i) die *neuartigen Interaktionsgeräte* in *post-desktop* Umgebungen unterstützt werden, ii) Mittel bereitgestellt werden, um mit der Dynamik dieser Umgebungen, in denen Ressourcen nur noch *lose gekoppelt* sind, umzugehen, und iii) die vom Nutzer empfundene *Latenz* bei der Interaktion muss reduziert werden.

Im Hinblick auf diese Ziele liefert die vorliegende Arbeit folgende Beiträge.

Ein **neues theoretisches Modell** zur Beschreibung und zum besseren Verständnis des Problems der Verwendung von Anwendungen in *post-desktop* Umgebungen.

Um die unterschiedlichen Ressourcen in *post-desktop* Umgebungen zu nutzen, muss eine Anwendung gleichzeitig Eingabedaten von beliebigen Ressourcen verarbeiten und die Ausgabe an den Benutzer modifizieren können. Existierende Ansätze unterstützen nur jeweils einen der beiden Aspekte gleichzeitig. Entweder sind sie optimiert für die Verarbeitung von Eingabe, wobei sie auf eine Pipeline-Architektur zurückgreifen und dann keine Möglichkeiten zur Modifikation der Ausgabe zur Verfügung stellen, oder, sie sind optimiert für die Modifikation der Ausgabe. In diesem Fall können sie lediglich Eingabedaten von Maus und Tastatur verarbeiten. Diese Arbeit schlägt einen Ansatz vor, der diese Beschränkungen aufhebt. Mit dem Konzept der *Interaktionsstrategien*, umgesetzt im *MundoMonkey System* kann man, dank der Anbindung der MundoCore Middleware, Eingabedaten beliebiger Quellen flexibel verarbeiten und gleichzeitig die Ausgabe zum Benutzer unter Verwendung der mächtigen Document Object Model (DOM) Schnittstelle des Webbrowsers modifizieren. Dieser Ansatz wurde evaluiert, indem mehrere Interaktionsstrategien für ausgewählte *post-desktop* Umgebungen implementiert wurden. So z.B. eine

Strategie für sprachbasierte Interaktion mit Webseiten, die sich in einer Studie gegenüber herkömmlichen Ansätzen als überlegen herausstellte.

In post-desktop Umgebungen muss mit Ressourcendynamik umgegangen werden. Ressourcen erscheinen und verschwinden in diesen Umgebungen. Der Nutzer muss die Möglichkeit haben, die verwendeten Anwendungen und Ressourcen effizient zu kontrollieren. Diese Arbeit stellt ein **funktional vollständiges meta User Interface (UI)** vor, das es dem Nutzer ermöglicht, Anwendungen und die post-desktop Umgebung zu kontrollieren, was mit bisher existierenden Ansätzen nicht möglich ist. Das vorgestellte MineManager meta UI ist das erste, das ermöglicht, ein digitales meta UI vollständig synchron mit den physischen Veränderungen in der Umgebung zu halten. Eine Implementierung für das iPhone und die MundoCore Middleware, genannt MineExplorer, wurde evaluiert. Studien haben die Tauglichkeit des Konzeptes belegt.

Das Problem der Latenz bei der Interaktion mit Web Anwendungen wurde schon in zahlreichen anderen Arbeiten behandelt. Der in dieser Arbeit vorgeschlagene Ansatz **Mundo-Proxy** ist der erste, der in mobilen Umgebungen eingesetzt werden kann und der zudem keine Änderungen bei der Implementierung des back-ends der Webanwendung oder der Webanwendung selbst voraussetzt. Unabhängigkeit von der Webanwendung wird durch die Verwendung von **intelligenten Vorhersagealgorithmen**, die zukünftige Nutzeranfragen auf Basis vorhergehender Anfragen vorhersagen, erreicht. Die Eigenheiten mobiler Umgebungen werden durch den **Huckepack Ansatz** adressiert, der es ermöglicht die Netzwerkbandbreite und Batterielaufzeit des Mobilgeräts effizient zu nutzen. Die Evaluierung zeigt, dass sich mit diesem Ansatz die vom Benutzer bemerkte Latenz um bis zu 25% reduzieren lässt.

Schließlich beschreibt diese Arbeit das **MundoWeb** System, das alle drei oben beschriebenen Komponenten in eine abwärtskompatible Erweiterung der Web Infrastruktur integriert, die die Anforderungen zur Interaktion in post-desktop Umgebungen erfüllt.

Abstract

Software applications are about to ‘leave’ the desktop and to be used in a variety of different settings instead. When an application is developed, it cannot be foreseen from which environments it will eventually be accessed. A good example for this are Web applications. They are accessed and used with many different Web browsers, depending on the user’s choice. Many of the environments in which applications are accessed are becoming so-called *post-desktop* environments, i.e., environments that contain a variety of different interaction devices, beyond mouse, keyboard and screen, or even combinations of multiple interaction devices federated into a single UI. These environments are not necessarily equipped with a good network connection, so that the user may perceive disrupting latency due to the slow network connection, while interacting with applications

The object of research of this thesis is the system infrastructure necessary to ensure that applications can be accessed in arbitrary post-desktop environments. We seek to augment and extend the Web system infrastructure to achieve this goal. This requires i) means to utilize *novel interaction resources* found in post-desktop environments, ii) coping with the dynamics of those environments, where resources are *loosely coupled* and iii) mitigating *latency* perceived by the user while interacting with Web applications in post-desktop environments. In pursuing this goal, the thesis makes the following contributions.

In order to better understand the problem space, we developed a **novel theoretical framework** for describing the problem space of post-desktop access and use of applications.

In order to handle the wealth and diversity of resources found in post-desktop environments one needs to be able to process input from arbitrary resources and to modify the output presented to the user at the same time. Existing approaches only support one of these two features at the same time. They are either optimized towards processing of input, in which case they rely on a pipeline architecture but then do not provide means for structured output modification. Or, they are optimized for modifying output, in which case their input-handling capabilities are limited to mouse and keyboard. The approach proposed in this thesis overcomes these limitations. Our **concept of interaction strategies** as **realized in the MundoMonkey system** can flexibly handle input from arbitrary sources, thanks to the connection to the MundoCore middleware, *and* at the same time can perform modifications of the UI using the rich DOM of Web applications. The approach has been evaluated by implementing several interaction strategies for interesting post-desktop environments, e.g., a **strategy for voice-based interaction** with Web applications proved superior to state-of-the-art solutions in a user study.

In federated post-desktop environments the user has to deal not only with the diversity of resources, but also with the dynamics of the environment. Resources appear and disappear; the accessed applications must be easy to control for the user. We provide a single **functionally complete meta UI** to the user, allowing her to manage and control the post-

desktop environment, which is not possible with existing approaches. The MineManager framework developed in this thesis is the first to enable the use of a digital meta UI in synchronization with physical changes to the access environment. An instance of this framework, called MineExplorer, has been implemented for the iPhone and the MundoCore middleware. It has been evaluated with user tests that showed the validity of the underlying concepts.

The problem of user-perceived latency has been addressed for Web applications before. However, **MundoProxy** which is based on the concepts developed in this thesis is the only solution that can be used in mobile access environments, and which additionally does not require any changes to the implementation of the back-end services or the Web application. Application independence is achieved by using **intelligent prefetching algorithms** that predict future user requests based on observed previous requests. The peculiarities of mobile access environments are addressed by the **piggybacking mechanisms** that enable efficient use of the network bandwidth as well as of the energy of the mobile device battery. The evaluation shows that the user-perceived latency can be reduced up to 25% with our approach.

Finally, we propose **MundoWeb**, which integrates the three components explained above into a conservative extension to the Web access system addressing the requirements of post-desktop environments.



Contents

1	Introduction	1
1.1	Motivation	1
1.1.1	Research Goal	2
1.1.2	Realttime Enterprise	2
1.2	Research Approach and Structure of the Thesis	3
1.3	Publication Record	5

<hr/>	
2	Background and Requirements 7
2.1	Definitions 8
2.1.1	Access Environment 8
2.1.2	Post-desktop Access Environments 10
2.1.3	Access System 12
2.1.4	User Interface 14
2.1.5	Business Applications 15
2.2	Access System of Service Oriented Architecture (SOA) Backed Web Applications 16
2.2.1	SOA-backed Web applications 17
2.2.2	Components of the Web Access System 17
2.2.3	Comparison to Other Access Systems 18
2.2.4	Conclusion 20
2.3	Case Study 20
2.3.1	User Study with a UI Prototype 21
2.3.2	Applying Augmenting User Interfaces with Prediction, Modality and Context through Reasoning (AUGUR) to Make Existing Web Applications Context-aware 26
2.3.3	Conclusions 27
2.4	Requirements for a Post-Desktop Access System 28
2.4.1	Existing Web Applications 29
2.4.2	Supporting Existing and Novel Access Environments 29
2.4.3	Mitigate User-Perceived Latency 31
2.4.4	Management and Configuration Operations 32
2.5	Discussion of the Requirements 33
2.5.1	Requirements of Trewin et al. 33
2.5.2	Requirements for Personal Unified Controller (PUC) 35
2.5.3	Conclusion 37
2.6	Chapter Summary 38
<hr/>	
3	State of the Art 41
3.1	Model Driven User Interface Development 42
3.1.1	Early Model-Driven Approaches 42
3.1.2	UI Plasticity and Models at Runtime 43
3.2	Access Systems for Multiple Devices 47
3.2.1	Homogeneous Device and Multi-Monitor Access Systems 47
3.2.2	Access Systems for Inhomogeneous Devices 48
3.2.3	Discussion 50
3.3	Toolkits for Post-Desktop Hardware 50
3.3.1	Sensor Hardware Abstraction 51
3.3.2	Input Data Processing 52

3.4	Assistive Computing	55
3.4.1	Non Web-based Systems	55
3.4.2	Web-Based Systems	57
3.5	Extensions of the Web Access System	57
3.6	Discussion	58
3.6.1	Overview	58
3.6.2	Addressing the Limitations	60
3.7	Chapter Summary	65
<hr/>		
4	The Post-desktop Web	67
<hr/>		
4.1	Extending the Web Access System	68
4.2	Interacting with Web Applications in Post-Desktop Access Environments	72
4.2.1	Interaction Resource	73
4.2.2	Interaction Strategies	74
4.2.3	Composition of Interaction Strategies	75
4.2.4	Resulting Extensions to the Web Access System	77
4.3	Meta Operations for Managing Web Applications and Post-Desktop Access Environments	78
4.3.1	Meta Operations for Managing Applications in Desktop and Mobile Access Environments	78
4.3.2	Meta Operations for Managing Post-Desktop Access Environments	79
4.3.3	Meta Operations for Multitasking in Post-Desktop Environments	80
4.3.4	Meta Operations for Access System Customization	80
4.3.5	Resulting Extensions to the Web Access System	81
4.4	Reducing User-Perceived Latency for Web Applications	82
4.4.1	Probabilistic Prefetching	83
4.4.2	Resulting Extensions to the Existing Web Access System	84
4.4.3	Piggybacking Cache Synchronization	84
4.5	Chapter Summary	85
<hr/>		
5	Models, Designs, Algorithms	87
<hr/>		
5.1	Interaction Strategies for Web Applications in Post-Desktop Access Environments	88
5.1.1	Developing Interaction Strategies	89
5.1.2	Composing Interaction Strategies at Runtime	93
5.1.3	Implementation Design	95
5.2	System Model for Augmenting the Web Browser with Meta Operations	97
5.2.1	Meta Model and Meta Object Protocol for Post-Desktop Access Environments	97
5.2.2	Proactive Meta UI	106

5.3	Latency-Reduction Algorithms	109
5.3.1	Proactive Prefetching of SOA Requests	110
5.3.2	Piggybacking and Distributed Cache	114
5.4	Chapter Summary	119
<hr/>		
6	Implementation and Evaluation	121
<hr/>		
6.1	MundoMonkey Evaluation	122
6.1.1	Voice Input Strategy Case Study	122
6.1.2	Pen-Input-Strategy Case Study	126
6.1.3	Federated Access Environment Scenario Case Study	129
6.2	MineManager Implementation and Evaluation	130
6.2.1	Implementation of the MineManager Framework for a Ubiquitous Computing Middleware	131
6.2.2	User Experiments	133
6.3	Evaluation of MundoProxy	138
6.3.1	Integration into an Enterprise SOA	140
6.3.2	Quantifying Latency Reduction in AjaxWeaver	140
6.3.3	Simulation with Realistic Workload	148
6.3.4	Experimental Evaluation	152
6.4	Chapter Summary	156
<hr/>		
7	Conclusion	159
<hr/>		
7.1	Thesis Contributions	160
7.2	Revisiting the Requirements	161
7.3	Outlook	162
7.3.1	Further Challenges in the Topic of the Meta UI	162
7.3.2	Generalization to Other Types of Back-Ends and Access Environments	163
7.3.3	Controlling a Smart Environment from a Web Browser	163
<hr/>		
A	Tasks from the MineExplorer User Study	165
<hr/>		
A.1	Aufgaben	165

Chapter 1

Introduction

1.1 Motivation

Software applications are no longer exclusively used at the desktop workplace. On the contrary, computers for accessing applications come in all shapes and sizes: from smart phones to slates and netbooks to TV set-top boxes and in-car systems, to name just a few. From the user's point of view the most distinguishing feature of all these different platforms for using software applications are their input and output capabilities [Bux02].

For example, commercially available computers provide small or large displays for output, accompanied by sound and sometimes tactile (vibration) output. And this list is just a subset of all available output options. The same diversity can be observed for input mechanisms: keyboards of all shapes and sizes, touch-screens, stylus and pen, etc (see [KM07] or [Bux83] for a more complete survey of in- and output techniques). We witness the dawn of a new era of computing, in which interaction with computers happens in so called *post-desktop* settings [Wei99], i.e., with these new in- and output devices in a host of different situations, not just in front of a desk. Besides the new interaction devices, post-desktop environments often have a worse network connection compared to desktop environments. This introduces problems, e.g., latency perceived by the user. Supporting arbitrary post-desktop settings with UI toolkits and system software has been a long-standing goal in the ubiquitous computing research community, and this thesis is situated in this context.

However, the number of applications that have been implemented using post-desktop toolkits is comparably small. On the other hand, many Web applications are already used in post-desktop settings today. Apparently, Web applications possess many desirable characteristics — otherwise their widespread success would be difficult to explain. This thesis seeks to identify the traits that make Web applications succeed, and to leverage and augment Web technology to support post-desktop interaction. We think utilizing Web applications as a base technology leads to more meaningful results, as novel concepts can be tried out with a myriad of existing applications.

1.1.1 Research Goal

The research goal of the present thesis can be described as follows:

Adapt access to Web applications in arbitrary post-desktop environments.

This comprises

- using the *novel interaction resources* found in those environments,
- coping with the dynamics of those environments, where resources are *loosely coupled*, and
- handling the *latency* perceived by the user while interacting with Web applications.

While the concepts developed in this thesis are general, there is a clear focus on the evaluation and application of these concepts to business applications in the realtime enterprise. This area of application is of great importance, as many existing applications, e.g., Customer Relationship Management (CRM) applications, serve business needs. We will detail this domain of focus in the following section.

1.1.2 Realtime Enterprise

Enterprises are organized along business processes; every business activity in an enterprise belongs to a business process, and every business process is associated with a high-level business goal, e.g., generating revenue or maintaining market leadership [vdAHW03]. To ensure business processes are executed fast and with high quality, they are often automated and supported with IT systems.

State-of-the-art IT systems supporting such enterprise organizations rely on SOA: process steps that can be automated are realized by calling services in the enterprise SOA back-end. Despite this automation effort, many if not all business processes, require intervention by human users. The human users participate in the IT-supported business process by means of *interactive business applications*. Thus, designing, developing and deploying highly usable interactive business applications are core activities in any enterprise. This thesis deals with the challenges for IT systems when these activities are carried out in the so called *realtime enterprise*.

Following the tendency towards the realtime enterprise [MG08], interactive business applications are increasingly *accessed and used in post-desktop environments*, e.g., on mobile devices [THLT07]. This way, employees can provide data to running business process

instances instantly, instead of reporting the data after they return to their desktop workplace. However, becoming *realtime* not only affects the deployment and use of interactive business applications.

Becoming *realtime* also has implications at the level of *processes*. The processes, and with them the applications supporting them, need to be changed in realtime to adapt to emerging business needs. As a consequence, supporting *business users*, i.e., end-users with some minimal technical knowledge, in developing and deploying interactive business applications becomes an important feature of the IT infrastructure [SSF08].

The technology of SOA-backed Web applications supports this need to some degree. The SOA infrastructure contains services that provide functionality at a granularity meaningful to business users [ABB⁺09]. Based on this architecture, business users are able to develop and deploy business applications rapidly using mash-up technology [HWDB08, HGJSS09].

However, SOA-backed Web applications have several limitations concerning their use in post-desktop environments.

- Existing Web browsers are hard to customize to new non-desktop settings. As a result, Web-based business applications cannot be deployed by business users into non-desktop settings that prevent usage of mobile phones, e.g., to information kiosks intended for customer use.
- If a mobile phone can be used, the browser exclusively relies on interaction capabilities of the mobile phone. The browser does not take advantage of the resources available in rich post-desktop environments, despite the fact that these have the potential to increase application usability [BM05a]. Failure to use this opportunity frustrates end-users, and as a consequence they resort to reporting data in bulk at the end of their working day instead of reporting them live and on-site.

We argue that it is necessary to overcome the limitations of current Web application technology with respect to post-desktop environments to realize the vision of the realtime enterprise. Business users should be enabled to develop interactive business applications, to deploy them into post-desktop environments, and to use the rich resources provided by those environments. As a solution, we propose several extensions to SOA-backed Web applications.

1.2 Research Approach and Structure of the Thesis

A considerable amount of research on the subject of interactive applications in post-desktop environments has been done, e.g., published in the UIST, CHI, DIS and EICS conference series proceedings. At the same time, an even larger body of free and commercial software components providing relevant functionality is readily available. To be able to describe

the state of the art in a coherent way, we first need to establish a **theoretical framework**, capable of describing and classifying systems in production use as well as those proposed in the research literature. We develop such a framework in **Chapter 2**.

Furthermore, we derive **requirements** for post-desktop interaction with Web applications. These requirements are **based on a case study conducted by the author**, but also reflect the state of the art on requirements for post-desktop interaction. In this case study, a Web browser was extended in a way adding context-awareness [CCDG05] functionality to Web applications [HSK07, SHF⁺07, HSM08, HS08, HSM09]. The list of requirements and the theoretical framework are the main outcome of Chapter 2.

Chapter 3 proceeds with a review of existing systems from different areas of research. We compare the systems proposed in the research literature to the requirements derived in the previous chapter. We perform a **qualitative comparison** of existing systems based on the published system description and the requirements. The analysis of the existing systems provided insights for the design of MundoWeb, the system proposed in this thesis. The chapter concludes with a discussion of the weaknesses of existing systems that are overcome by MundoWeb, pointing out where MundoWeb progresses beyond the state of the art.

Chapter 4 presents the overall **concepts behind MundoWeb**. As outlined in the motivation, MundoWeb makes heavy use of existing Web technologies. While this may seem an unnecessary restriction, the results from our case study and the review of the state-of-the-art systems suggests the opposite, i.e., that Web technology provides a solid foundation to build upon, and that using that foundation leads to new insights and thus to superior concepts.

Chapter 5 introduces the algorithms, models and data structures needed to realize the overall approach proposed in Chapter 4 in more detail. These two chapters contain the main research contributions of this thesis. Three components necessary to realize the proposed MundoWeb concept, called **MundoMonkey**, **MineManager** and **MundoProxy**, are described in detail in this chapter.

Chapter 6 reports the **evaluation results**. Most parts of MundoWeb have been implemented as a proof-of-concept system. Thus, the research approach for this thesis is formulative with some concept implementations, a common approach for the fields of Computer Science and Software Engineering [GRV04].

Finally, **Chapter 7** discusses the results presented in this thesis with respect to remaining limitations and possible future work.

1.3 Publication Record

In summary, this thesis makes the following contributions.

- We provide a consolidated set of requirements for interactive systems in post-desktop environments. It is the first such set to take into account the lessons learned from the huge success of Web-based systems. Thereby, this thesis contributes to the understanding of the design space of interactive systems for post-desktop environments.
- Based on these requirements, we provide a conceptual design for interactive applications for post-desktop access environments, called MundoWeb, again taking into account the existing achievements of Web-based systems.
- The proposed system comprises an adaptive and prefetching caching mechanisms that outperforms state-of-the-art-solutions.
- As MundoWeb has been designed to work with existing Web-based applications, the evaluation of the caching mechanism could be performed with realistic data sets, adding another level of validity to our findings, compared to the evaluation results found in the literature.
- Furthermore, the system comprises the conceptual design as well as the implementation of the MundoMonkey extension for the Firefox browser. This extension achieves a unification of the output-modification capabilities of end-user scripting for the Web with adaptive, pipeline-based input processing, overcoming limitations in existing systems for adapting application UIs to post-desktop environments.
- The MundoWeb system is completed by a meta UI especially designed for Web applications in post-desktop access environments.

The results of this thesis have been published in various national and international journals, conferences and workshops. The AUGUR case study has been described in [HSK07,SHF⁺07,HSM08,HS08,HS09,HSM09,SHF⁺08]. Furthermore, technology from the AUGUR case study was applied in [RS08], contributing to the understanding of requirements. The conceptual design of MundoWeb and the MundoMonkey extension have been published in [SH09]. The MundoMonkey extension is presented in more detail in [SHM09]. The meta UI has been described in [SH08] and [VSL⁺09]. The proactive prefetching was first described in [GSH⁺09] and in more depth in [SAGM10]. Improvements, including the evaluation with realistic workloads, have been published in [SGAM10].

Four other publications of the author [KS07,ABB⁺09,MSH09,HSSM10] are only loosely related to this thesis and will not be mentioned further.



Chapter 2

Background and Requirements

As pointed out in the introduction, many software applications are about to ‘leave’ the desktop and to be used with a variety of different interaction devices or even with combinations of multiple interaction devices federated into a single UI. In this context, both appropriate architectures and appropriate system software are important areas of research. The present thesis emphasizes *interactive business applications* in *post-desktop environments*. This chapter contributes to this issue by deriving a set of requirements for the *access system*, which is backed by requirements taken from the research literature and a case study.

This chapter is structured as follows. Basic terms and a theoretical framework are introduced in Section 2.1. Section 2.2 explains the Web as one instance of the theoretical framework. Web applications are chosen as an example because they i) have many desirable characteristics, and ii) are widely employed in production systems.

Section 2.3 reports the results of a case study in which the author was involved. In this case study, the access system of Web applications was augmented to overcome some of its limitations with respect to its support for *post-desktop environments*.

From the results of the case study, a new set of requirements is derived in Section 2.4. These requirements serve as the basis for the concepts described in this thesis. Section 2.5 compares the set of requirements used in this thesis to other sets of requirements from the literature. In conclusion, we find that although the access system of Web applications has shortcomings, it addresses a substantial subset of the requirements from the literature. Thus, further research should use the Web access system as a starting point rather than adopting a clean-slate approach. The chapter concludes with a summary in Section 2.6.

2.1 Definitions

We first need to define some basic terms that are important for understanding the concepts developed in this thesis. A key term used throughout this thesis is *interactive application*, for which we adopt the definition of [UIM92] and define as follows:

Definition 1 (Interactive Application)

An interactive application is computer software that

- *realizes domain functionality desired by a user, and*
- *is able to receive input from a human user and present output to a human user.*

An example for an interactive application is an accounting program that performs payroll calculations (i.e., payroll management is the domain functionality) and additionally interacts with an accountant through a Graphical User Interface (GUI). Thus, an interactive application comprises two functional subsystems: the domain functionality and the user interaction functionality. The latter part is also sometimes called the application's UI.

In this thesis we only consider *interactive* applications, even if we at times omit the attribute *interactive*. The combination of *receiving input* from a user and *presenting output* to a user is often referred to as *interacting* with a user.

2.1.1 Access Environment

Interactive applications depend on system software to provide services. For example, interactive applications typically make use of a file system service for storing and retrieving data. The entirety of services on which the application depends is called the *operating environment* of the application [KBWA94]. In monolithic systems, a single *operating system* implements all services in the operating environment.

Historically, interaction with the user, i.e., access to *physical* interaction devices, has been just another service provided by the operating system. For example, [UIM92] speaks of the *User Interface Runtime System* but means the operating environment of the whole application, comprising domain *and* user-interaction functionality.

The classic desktop workplace setup — comprising a screen, a keyboard and a mouse as in- and output devices — has become a de-facto standard, and thus these physical devices are considered as part of the operating environment. The desktop setup is so common that it gave name to a whole era of interactive applications, called the ‘desktop computing’

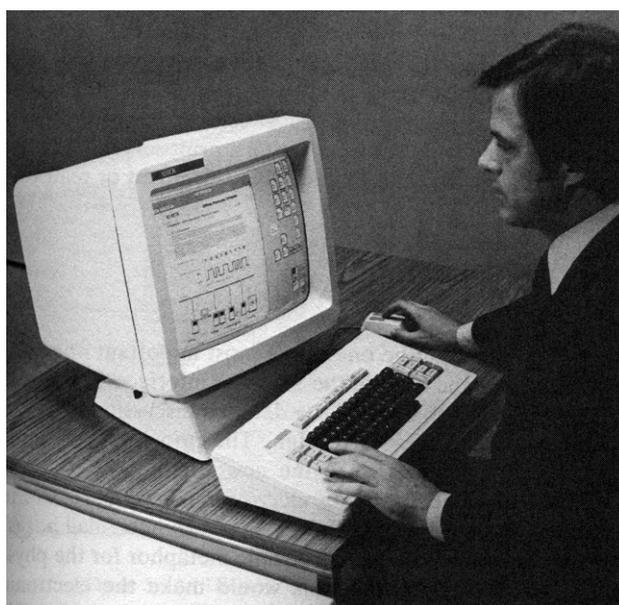


Figure 2.1.: Xerox Star user interface: The beginning of the Desktop Computing era.

era. The ‘desktop computing’ era is said to have started in 1981 with the introduction of the Xerox Star (see figure 2.1) [JRV⁺89] and is still the predominant environment for applications today.

Considering physical devices as part of the operating environment falls short when we try to describe the operating environment for non-monolithic applications, like Web applications. If interaction devices are considered as essential parts of the operating environment, then the Web browser would have to be considered as part of the operating environment of a Web application. However, only a minimal part of the Web application actually *runs* in the browser. It seems more intuitive to consider the operating system of the Web server as operating environment. This operating environment is unaffected by changes at the level of the browser, e.g., when switching from a desktop browser to a mobile phone browser.

Therefore, we propose a novel conceptual framework, clearly separating *virtual* and *physical* aspects of the operating environment, and introduce the concept of *access environment*. The access environment captures all physical aspects of the environment in which the user interacts with an application (see Figure 2.4). We define as follows.

Definition 2 (Access Environment)

The access environment comprises all resources available to the user for interacting with interactive applications.

Discussion

In this section we will discuss the distinction between the concept of an access environment and the existing concept of *context of use* as used by Calvary et al [CCT⁺03] and Blumendorf [Blu09].

At first sight, the definition of access environment resembles that of *context of use* from Calvary et al. and Blumendorf as defined in [CCT⁺03]. However, the definition in [CCT⁺03] intermingles the *computational* platform and *software* platform an application runs on with the *physical* environment and *interaction* capabilities. The definition of access environment clearly separates the two, which helps to analyze distributed applications. Changing the Web server from a Linux to a Windows operating system would alter the context of use, while the access environment remained the same.

We argue that clearly distinguishing between access and operating environment (which may also be called computational platform or software environment) is crucial to understanding the requirements for interactive applications in post-desktop settings. Although both, access and operating environment, are changing towards post-desktop settings, different problems arise. For example, porting applications to *post-desktop operating environments* may mean leveraging cloud computing infrastructures for running SOA services. In this thesis, we do not aim to address these problems. On the other hand, the shift towards *post-desktop access environments* comes with its own set of problems, which we will examine in the following.

2.1.2 Post-desktop Access Environments

As predicted by Mark Weiser [Wei99], we see interactive applications being used in more and more diverse settings. Using the definition above, we characterize these settings as post-desktop access environments and define as follows.

Definition 3 (Post-desktop Access Environment)

A post-desktop access environment is an access environment which differs from the classic desktop environment

- *in terms of the resources contained, or*
- *in terms of the procedure used for its construction, or*
- *in both of these aspects.*

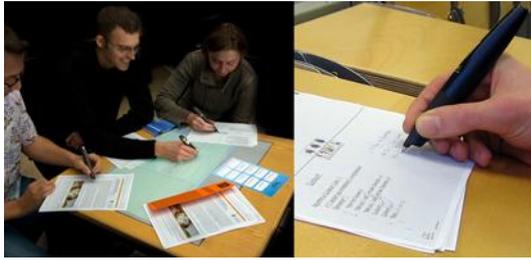


Figure 2.2.: Pen-and-paper-based operation environment.



Figure 2.3.: Voice-based operation environment in a living room.

For example, voice-based access environments are a type of post-desktop access environment that is already in use today. In a typical such environment, information is presented as spoken text, information is requested from the user via prompts, and user utterances are processed with the help of speech recognition grammars [CGB04]. Another example of a type of post-desktop access environment use pen-and-paper interfaces [Ste09a] for interacting with the user. Two instances of such access environments are shown in Figures 2.2 and 2.3.

The desktop era is associated with one particular style of interaction called Windows, Icons, Menus, and Pointers. Therefore one often finds the term post-Windows, Icons, Menus, and Pointers (WIMP) interfaces in the literature for referring to interactive systems designed for post-desktop environments. This is however not completely accurate, as the WIMP style of interaction can be used in post-desktop access environments, e.g., on large touch displays and vice versa, post-WIMP interfaces exist for the desktop environment, e.g., using mouse gestures.

Important Types of Post-desktop Access Environments

Post-desktop access environments of special importance for the realtime enterprise are *mobile access environments*. Typically, they contain a single mobile phone. In such environments, energy considerations are extremely important, as they rely on battery-powered devices.

Definition 4 (Mobile Access Environment)

A mobile access environment is a special type of post-desktop access environment consisting of a single mobile device with a small screen and built-in keyboard.

Apart from the energy constraints and limited built-in interaction capabilities, mobile access environments only provide low-bandwidth, high-latency network connections.

Another important class of post-desktop access environments are *federated post-desktop access environments*. These access environments are constructed from a loosely coupled set of components, connected by means of a communication middleware. *Interactive [work] spaces* or *smart spaces* are synonyms for these types of environments used in the literature [SdF07, JFW02, BDB⁺04, Bal08]. Supporting the bottom-up construction of such environments is an essential requirement for supporting these environments. [RB03, AE08].

Definition 5 (Federated Access Environment)

A federated access environment is a special post-desktop access environment consisting of multiple components (mobile and/or stationary) connected by means of a communication middleware.

The communication middleware masks the heterogeneity of the underlying communication network and provides location transparency, so that applications accessed in a federated access environment do not need to know where exactly a component is located [CDK05]. Typical components found in a federated access environment are output devices, such as screens, but also input devices such as pointing devices. By using other interaction devices from the environment, mobile access environments can become federated access environments [BM05a, MMBE00, MNWM04, NMH⁺03].

2.1.3 Access System

As we briefly mentioned before, interactive applications comprise functionality from two categories: domain functionality, implemented in software by the *domain code*, and interaction functionality, which is implemented in the *interaction code* (see Figure 2.4).

In desktop computing, the interaction code uses services from the operating system for accessing interaction devices. There is a large potential for reuse of interaction-related functionality [MR92], since *all* interactive applications need interaction functionality. To facilitate this reuse, interaction-related code has been pulled out of the application and put into middleware, the operating system or code libraries. We call the resulting system encapsulating this functionality *access system*, and define as follows.

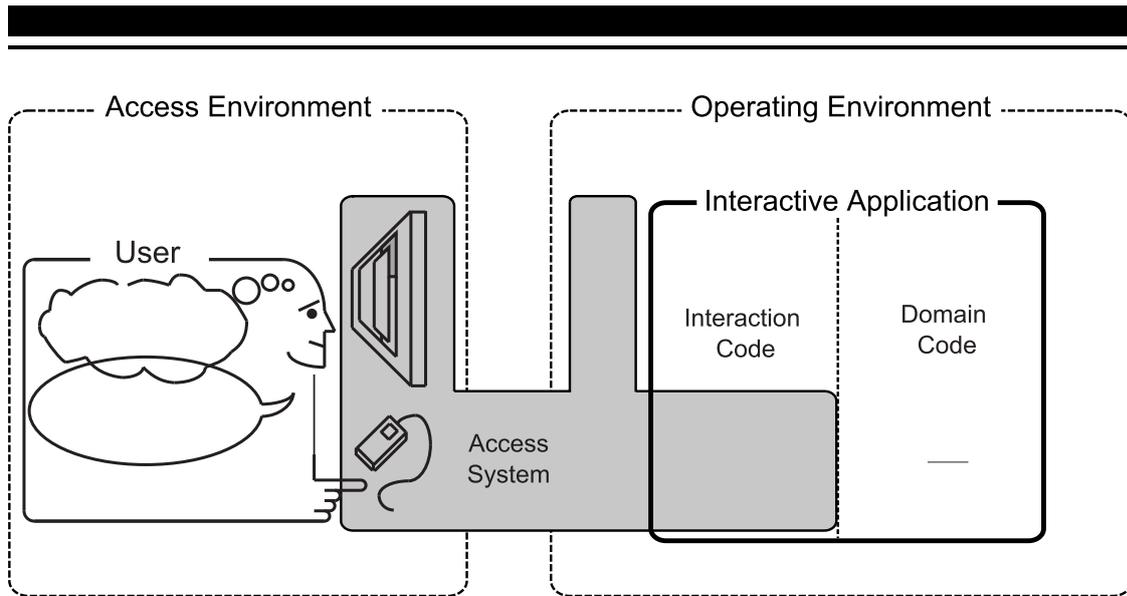


Figure 2.4.: An interactive application contains domain code and a interaction code. It is executed in an operating environment and accessed by the user in an access environment. The access system may comprise operating system services and library code. It may also be distributed, connecting to the access environment over a network connection. In this case, the access system runs in another operating environment in the access system, which is not drawn in the picture.

Definition 6 (Access System)

The access system is software (libraries, architectures, system processes) that is reused in multiple applications for providing interaction functionality.

Early in desktop computing, the access system was just the part of the operating systems responsible for connecting to the interaction devices, i.e., graphics and mouse drivers. Shortly after, UI code libraries, i.e., toolkits emerged.

Discussion

In this section we discuss the relationship between the concept of an access system and similar concepts found in the literature.

UIMS

We do not use the term UIMS, which has been proposed already in [Kas82], to refer to the access system, because UIMS have the connotation of providing a very high level of abstraction to the application developer. For example, UIMS allow the application developer to specify the dialog between the application and the user with state-charts [Wel89]. UIMS systems have not gained widespread acceptance [MHP00], as practice has shown that developers prefer fine-grained control over the interactions with the user over a more abstract specification language.

Runtime Architectures

One important aspect of an access system is its *runtime architecture* [Blu09]. Different architectures have been proposed in the literature, ranging from monolithic [UIM92] to highly distributed [Cou87]. The most successful access systems in use today heavily rely on library code, i.e., GUI toolkits such as WPF [Mic10] or Qt [Nok10] that have to be linked with the application and thus executed as part of the application process.

Although the runtime architecture is a part of the access system, the access system also comprises other aspects. For example, the Web access system that we will discuss in Section 2.2 comprises infrastructure for deploying applications into access environments.

2.1.4 User Interface

We distinguish the UI from the access system. While the access system remains the same for all applications, the UI is a part of the application and therefore differs from application to application.

On the one hand, the UI interfaces the access system, and on the other hand it interfaces the domain code of the application. Here, we take a code-centric view of the UI, i.e., defining it as a piece of software, which is common in the interactive systems engineering community [MR92]. Please note that other understandings of UI exist, i.e., where the UI is something the user perceives while interacting with an application. This understanding is also commonly found in the Human Computer Interaction (HCI) community.

The access system provides the execution environment for UIs. Just as the Java Virtual Machine provides a unified execution environment for Java byte-code on any underlying system platform, the access system should provide an UI execution environment in any access environment.

Every access system comes with a User Interface Description Language (UIDL), which the developer employs to specify the UI. Thereby, we use the term UI *Description Language*, as it is often a special-purpose language of descriptive nature. Note, however, that the term *description* should not be understood as only referring to UIDLs following a descriptive

or markup style. The UIDL can also be procedural or object-oriented. Likewise, *language* does not imply that the UIDL resides on a meta-level, i.e., the UIDL can be realized by a library inside a host language. An example is the Java Swing API that provides a language for describing UIs and uses Java as host language. This UIDL is object-oriented and implemented as a library in the Java language.

The UIDL gives rise to a syntactic structure of the UI. Thereby, *elements* in the UI, corresponding to atomic expressions in the UIDL. These may be grammar rules, GUI widgets or constraint rules.

Reflective and Non-Reflective Access Systems

If the representation of a UI in the UIDL differs from the representation of the UI accessible to the access system at runtime, we call the access system *non-reflective*, otherwise we call it *reflective* [Mae87]. Real access systems are often situated between these two extremes. For example, if we consider the Java Virtual Machines as an access system (for the Swing UIDL), it provides structural reflection capabilities, meaning one can manipulate the objects making up the UI at runtime. However, it does not provide behavioral reflection, meaning one cannot alter the behavior of these objects, at least not without additional tools for byte-code manipulation [Chi00].

We say the access system *adapts* a UI to a particular access environment. Using the verb *adapt* in this context is common in the research literature on UIs for post-desktop access environments. The access environment at hand is one target for the adaptation process performed by the access system [HI01]. If one would adopt a less UI-centric approach, one would say the UIDL representation is *interpreted* (possibly after compilation to an intermediary format, like Java byte-code) in the access environment.

For a (partly) reflective access system, the question arises whether the adaptation should be *transparent*, i.e., the structure and behavior accessible through introspection should not be aware of the adaptations taking place, or whether the adaptation itself should have an effect on the reflected UI.

2.1.5 Business Applications

Although this thesis deals with general interactive applications, it has been motivated by the needs of the real-time enterprise. In this context, a subset of interactive applications is especially important, which we call interactive business applications.

Definition 7 (Interactive Business Application)

An interactive business application is computer software that

-
-
- *serves a business process, and*
 - *is able to receive input from a human user and present output to a human user.*

The definition distinguishes interactive business applications from other kinds of computer software, for example

- system software that helps make the computer system operational, or
- interactive games, which do not serve a business process.

A large share of interactive business applications is implemented as SOA-backed Web application, and we also conducted the case study in that context. Therefore, we will discuss the Web access system in some depth in the next section, Section 2.2.

Before that, we take a look at one important stakeholder interacting with such applications: the *business user*.

Definition 8 (Business User)

A business user is an employee of an enterprise who is savvy on using software for business and organisation purposes, i.e., more a user who is more knowledgeable than an end-user, but who is not a technical expert, i.e., not a trained software engineer. Business users may be capable to do some programming, e.g., in spreadsheet applications or using a graphical or scripting programming language.

Business users may act as end-users, i.e., just use business applications. However, in the real-time enterprise, business users are expected to develop (or at least customize existing) interactive business applications to match changing business processes, and to deploy these interactive business applications. Currently, this need can be supported by composition environments for web applications, such as [HWDB08] or business process management tools [RRMvdA05].

2.2 Access System of SOA Backed Web Applications

Many interactive business applications are currently implemented as SOA-backed Web applications. We will define this term more precisely in this section and then proceed to analyze the access system of SOA-backed Web applications.

2.2.1 SOA-backed Web applications

As there is no commonly accepted definition of what *the Web* is or what constitutes a *Web application*, we make the interpretation used in this thesis explicit:

An SOA-backed Web application is an interactive business application where

- the domain code uses services from an enterprise SOA infrastructure,
- the interaction code runs on a Web server and generates Hypertext Markup Language (HTML) documents (optionally including Cascading Stylesheet (CSS) and JavaScript),
- the user downloads and interacts with these documents via a *Web browser* in the access environment,
- the communication between the browser and the Web server is performed over Hypertext Transfer Protocol (HTTP)

Other definitions of Web application may include web services that are not meant to be accessed with a browser or extend the definition to comprise any application delivered via HTTP, including Java applets that use proprietary protocols for communication later. However, we feel that this definition is not unreasonable and applies to most if not all of the existing interactive business applications that are commonly considered web applications.

2.2.2 Components of the Web Access System

Interaction with an SOA-backed Web application proceeds as follows: The user initiates interaction by accessing a Web application with a Web browser, for example, by typing the Uniform Resource Locator (URL) of the application into the address bar. The browser sends an HTTP request over the network to the Web server responsible for the URL. At the Web server, the request is processed. In the simplest case, the request points to a static file on the server from which the reply is read. For interactive business applications, the processing usually involves more complicated steps for computing a reply, for example, calling services in the enterprise SOA back-end. Once the reply is prepared, it is sent back to the browser as an HTTP response. This response contains an HTML document, which is rendered by the browser so that the user can interact with it. The next HTTP request to the Web server may include the results of this interaction, such as the values filled in a form. By using Asynchronous JavaScript and XML (AJAX) technology, subsequent requests may also be initiated and handled by the JavaScript code contained in the first response.

From this description the main components of the Web access system become apparent. The Web access system comprises

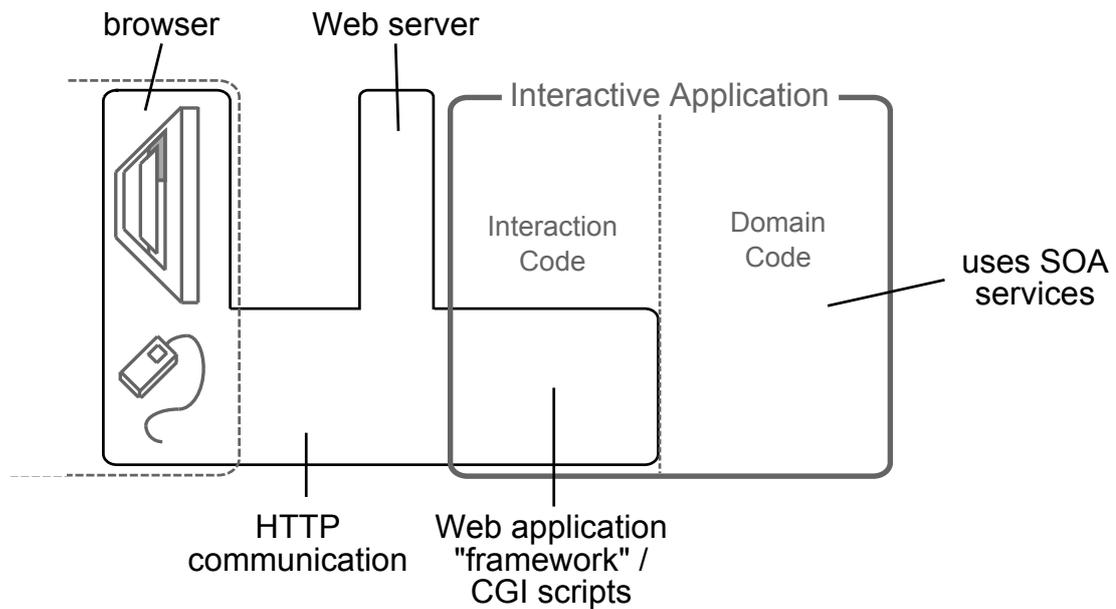


Figure 2.5.: Components of the Web access system used in SOA backed Web applications comprise the Web browser, the Web server and optionally a Web application framework, like Ruby on Rails [ROR].

- the *Web browser* (Firefox, Safari for iPhone OS, Lynx, etc.)
- that implement standards such as HTML, CSS and DOM for rendering UIs
- and that furthermore implement the HTTP protocol for communicating with the *Web server*;
- the *Web server* itself, again implementing the HTTP protocol and also
- implementing an interface to applications, such as the Common Gateway Interface (CGI);
- and optionally, *libraries* facilitating the dynamic generation of HTML documents.

Figure 2.5 shows an overview of the main components in the Web access system. There is no distinction made between the Web browser application and the operating system on which this application runs. As component in the Web access system, *Web browser* is used to refer to the combination of the two.

2.2.3 Comparison to Other Access Systems

The Web access system has immense practical relevance, and its components are widely known. Model-driven engineering [CCT⁺03, DS01] is also commonly used as a framework to describe interactive systems and their respective access systems. The Web access system

can be interpreted and described using this framework, and we will provide the necessary conceptual mappings in this section. This allows us to analyze and compare approaches from this camp later in Chapter 3.

Model-Driven User Interfaces

Model-driven development (MDD) embodies the idea of raising the level of abstraction of artifacts intended to instruct computer hardware. Instead of requiring developers to think and specify at the programming-language level — e.g., the level of abstraction afforded by Java — they should be allowed to use more abstract ways of specifying a solution. [Küh07]

In the Model Driven Architecture (MDA) standard, the most popular variant of Model Driven Development (MDD) standardized by the Object Management Group (OMG), a system is first described with a Platform Independent Model (PIM) which is then transformed into a Platform Specific Model (PSM). The MDA standard supports multiple stages of refinement, whereby the PSM resulting from one transformation stage plays the role of a PIM for the next transformation stage. This allows the developer to provide a refinement hierarchy with arbitrary levels.

Model Driven User Interface Development (MDUID) applies the principles of MDD to the development of interactive applications. The Cameleon framework [CCT⁺03] defines a canonical standard of four levels of PIMs and PSMs for MDUID, as shown in Figure 2.6: the concepts and tasks level, the Abstract User Interface level, the Concrete User Interface level, and the Final User Interface level.

The Cameleon framework does not prescribe which languages and meta models should be used to describe the models at each level. Popular choices in the literature are, e.g., CTT [Pat99] for the task and concepts level, UsiXML [LVM⁺05] for the Abstract User Interface and Concrete User Interface level, and, e.g., HTML for the Final User Interface [CCT⁺03, PSMM08]. However, other choices are possible, e.g., UML and Java are used in [LSHA08].

The UIDL of MDUID systems is defined by the meta models, which prescribe the form of the models. MDUID systems provide tools supporting the developer in creating these models. The models from the developer serve as the initial PIM, which is then transformed into a final PSM that can be rendered in the access environment. This transformation can comprise multiple steps that may be executed automatically, semi-automatically or manually. The transformation steps perform the distribution of the single input model to the components of a federated access environment and re-mold the source UI model into a form suitable to the access environment components.

The distinction between the model-driven systems

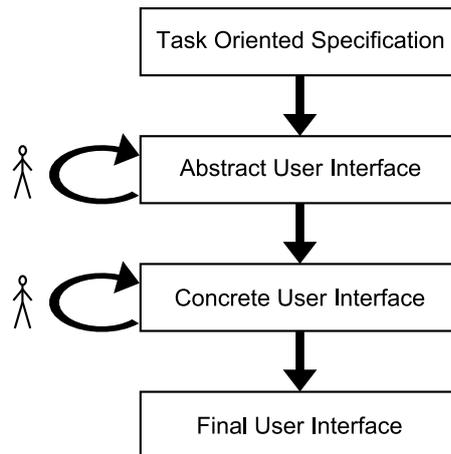


Figure 2.6.: The Cameleon framework standardizes four layers of models for UI development.

2.2.4 Conclusion

The Web access system has several similarities to systems using MDUID. The combination of HTML, CSS and JavaScript used in the Web can be seen as the representation formalism for UIs at the Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI) level. Only the task and concepts level is not supported by the Web access system. The static structure of HTML documents is specified by the HTML schema, which acts as a meta model for HTML. The semantics of HTML are formally defined by the DOM standard [DOM03]. In practice, competing semantics are provided by the different browser implementations.

The Web

2.3 Case Study

One novel aspect of post-desktop access environments compared to desktop access environments is the presence of other input sources than keyboard and mouse. One subset of such sources are *sensors* that enable applications to react to the context of interaction. Applications thereby become context-aware [CCDG05, SDA99]. Web applications are normally not context-aware. Thus, making web applications context-aware is one step towards the goal of this thesis.

The problem of adding context awareness to Web applications was investigated in the AUGUR project. The goal of the AUGUR project was to leverage contextual information from sensors to increase the usability of Web applications. AUGUR considered a wide variety of context sensors. These sensors also comprise *virtual* sensors, and one very

important virtual sensor is the user's own interaction history with a Web application [HS07].

A design goal of the AUGUR system was to avoid requiring premodeled information about applications and context sensors to make Web applications context-aware. The meaning of the sensors and possible connections to the Web application is inferred from a comparison of the sensor data and the user's interaction [HM09]. Alternatively, the end-user can provide a mapping between sensors and Web application in a graphical editor [HSM09].

The AUGUR system serves as a case study for the approach proposed in this thesis and helped to clarify two important questions.

- Contextual information is unreliable in nature [HI01]. Still, using it can often increase usability. The question is how one can use the inherently unreliable context information in a UI so that the net effect is positive, i.e., the benefits of using correct contextual information are not outweighed by the drawbacks of using wrong contextual information.
- The second question is whether it is feasible to integrate contextual information in the UI of arbitrary Web applications simply by accessing their HTML UI in the browser, thereby achieving independence from context sources and independence from Web applications at the same time.

2.3.1 User Study with a UI Prototype

To answer the first question, we conducted an extensive user study. We tested different options for integrating contextual information in Web UIs and measured their effect on the perceived usability of Web applications [SHF⁺08].

Method

The experiment was conducted online. We set up a Web site for the study that resembles a Web application for train ticket reservations. The Web pages of the application were instrumented, and recorded detailed timing data as well as user input.

Participants

Potential participants were sent an invitation email with a link to the online study. In total, 40 persons participated in the experiment, most of them computer science students, but also faculty members and persons without computer science background, who would have been hard to reach without a Web-based test setup.

The screenshot shows a web interface for a travel booking scenario. At the top, there are three tabs: "Reisedaten", "Verbindungsauswahl", and "Bezahlung". The "Reisedaten" tab is active. Below the tabs, the form is divided into several sections:

- Start & Ziel / Datum & Uhrzeit:**
 - * Von: [text input]
 - Über: [text input]
 - * Nach: [text input]
 - * Hinfahrt: 2007 | Mai | 18 | 11 | 20 | Abfahrt
 - Rückfahrt: Jahr | Monat | Tag | Stunden | Minuten | Abfahrt
- Verbindungen & Reisende:**
 - * Verkehrsmittel:
 - Standard
 - Ohne ICE
 - Ohne ICE/IC/EC
 - Nur Nahverkehr
 - * Reisende: 1 Erwachsener | Keine BahnCard | 2. Klasse
 - [-> Weitere Reisende hinzufügen](#)
- Ergebnisse:**
 - Verbindungen Suchen

To the right of the form, there is a grey box titled "Ihre Aufgabe" (Your Task) containing the following information:

Von: Darmstadt
Nach: Frankfurt
Wann: 26.07.2007
Abfahrt: 18:30 oder später
Bahn Card: BC 50, 2. Klasse
BC Nummer: 912837465
Ticket: 1 Erwachsener, 2. Klasse

Figure 2.7.: First step of the scenario

Design

In total we tested three conditions,

- inactive: no use of contextual information - the baseline interface
- correct: correct contextual information was used in the interface
- wrong: wrong contextual information was used in the interface

Each participant was assigned to either the *correct* or *wrong* condition, so this was varied between subjects. As a control condition, every participant performed the very same task in the *inactive* condition, so that the inactive / context-aware condition was varied within subject. To control for learning effects, half of the participants performed the task in the *inactive* condition first, half of them started with their respective flavor of the *context* condition.

In each condition, several variants for using contextual information in the UI were used, ranging from a highlighting of the most relevant entry field through a simple prefill of the relevant entry field to the complex display of multiple suggestions.

We obtained measurements for the time on task by instrumenting the website. Errors were counted manually by comparing the entered information to the correct information that was required in the scenario. Finally, user satisfaction and prospective long-term effect were measured with a six-level Likert scale.

Procedure

Each participant ran through a series of form filling tasks that were drawn from a hypothetical travel booking scenario. Participants had to enter connection details (see Figure 2.7),

Reisedaten		Verbindungsauswahl		> Bezahlung					
Details	Bahnhof/Haltestelle	Datum	Zeit	Dauer	Umsteigen	Produkte	Preis (nur für die Hinfahrt)	Buchung	
<input checked="" type="checkbox"/>	Darmstadt Hbf Frankfurt (Main) Hbf	Do, 26.07.2007 Do, 26.07.2007	ab 18:30 an 18:48	00:18	0	RB	6,70 EUR keine Fahrradmitnahme Online Buchung nicht möglich.	<input type="button" value="→"/>	
<input checked="" type="checkbox"/>	Darmstadt Hbf Frankfurt Hbf (tief)	Do, 26.07.2007 Do, 26.07.2007	ab 18:35 an 19:13	00:38	0	S	6,70 EUR Online Buchung nicht möglich.	<input type="button" value="→"/>	
<input type="checkbox"/>	Darmstadt Hbf Frankfurt (Main) Hbf	Do, 26.07.2007 Do, 26.07.2007	ab 18:57 an 19:15	00:18	0	IC	7,00 EUR	<input type="button" value="→"/>	
<input checked="" type="checkbox"/>	Darmstadt Hbf Frankfurt (Main) Hbf	Do, 26.07.2007 Do, 26.07.2007	ab 19:06 an 19:24	00:18	0	RE	6,70 EUR Online Buchung nicht möglich.	<input type="button" value="→"/>	
<input type="checkbox"/>	Darmstadt Hbf Frankfurt (Main) Hbf	Do, 26.07.2007 Do, 26.07.2007	ab 19:24 an 19:40	00:16	0	IC	7,00 EUR	<input type="button" value="→"/>	

Figure 2.8.: Second step of the scenario

Reisedaten		Verbindungsauswahl		Bezahlung	
BahnCard <input checked="" type="radio"/>		CreditCard <input type="radio"/>		ec-Card <input type="radio"/>	
Kartennummer: 912837465		Kartennummer: Control Bits (auf der Rückseite der Karte zu finden):		Kontonummer:	
		Gültig bis 01 2007		Bankleitzahl:	
Bestätigung des Online-Kaufs					

Figure 2.9.: Third step of the scenario

select a connection (see Figure 2.8) and provide payment information (see Figure 2.9). In the *correct* and *wrong* condition the contextual information was injected into the UI. After each run through the scenario, participants filled in the questionnaire.

Apparatus

All three conditions of the Web application used the same HTML markup. The different conditions were implemented by means of JavaScript functions that did or did not introduce the context-aware enhancements to the UI. These modifications were thus performed entirely in the browser, without any changes to the underlying Web application. The context data did not come from real sensors, but simulated data was used instead.

The JavaScript code also took timestamps of the user interaction, similar to the approach of [AS07]. The questionnaire results were stored by server-side PHP Hypertext Preprocessor (PHP) scripts. The whole apparatus was implemented in an extensible manner so that it could be reused for another study [RS08].

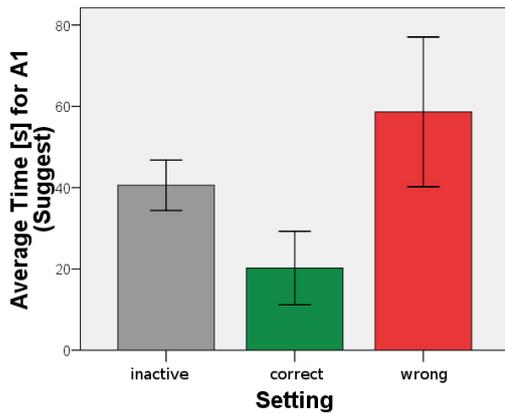


Figure 2.10.: Effect of the context-aware suggestions in the first step of the scenario (see Figure 2.7)

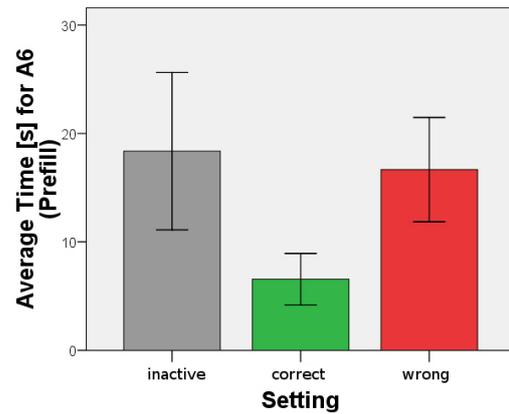


Figure 2.11.: Effect of the context-aware prefilling in the second step of the scenario (see Figure 2.8)

Study Results

We measured the time it took the user to fill out the three forms of the scenario. The effects of the different implementations of context-awareness are shown in Figures 2.10 to 2.12. For all forms, the least amount of time for filling was needed when the correct proactive support was provided to the user. An adverse effect of using wrong context information to support the user was observed. However, in the case of prefilling, wrong support improved the efficiency compared to the inactive version of the form. This is highly counterintuitive, and we believe the wrongly filled in data was used as a template by the users. Users did not make more errors in the *wrong* setting compared to the inactive setting, see Figure 2.13. In general, the results on efficiency suggest that implementing context awareness in this particular way would indeed improve Web applications.

The results from the questionnaire survey was inconclusive with regard to whether such a behavior did disturb users, or would disturb them in the long term (questions *disturbed* and *disturbslongterm* in Figure 2.14). Surprisingly, even the support in the *wrong* condition was considered *helpful* by the users (question *helpful* in Figure 2.14). As expected, users did generally *like* (question *like* in Figure 2.14) the *correct* support more. In contrast, the results for the *wrong* support were also acceptable to users. We conclude that users benefit from context-aware support in Web applications, even though the support is inevitably wrong at times. Therefore, the first question raised in the beginning of the section could be answered positively: using context information, although unreliable in nature, is a feasible way to improve the usability of Web applications.

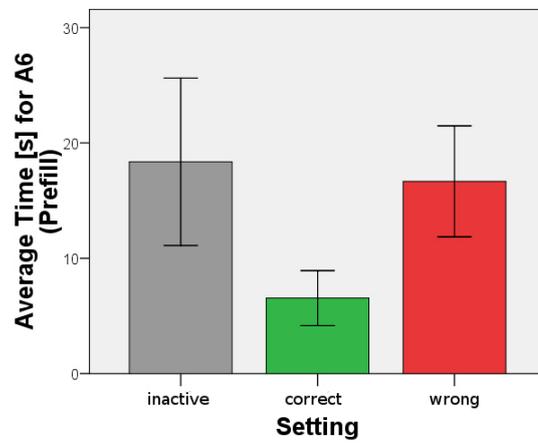


Figure 2.12.: Effect of context-aware highlighting in the third step of the scenario (see Figure 2.9)

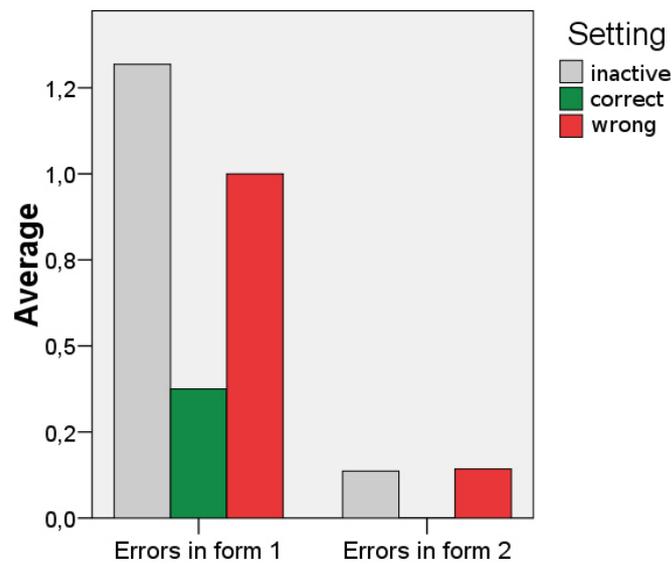


Figure 2.13.: Participants did not make more errors in the *wrong* setting compared to the inactive baseline. With correct context-aware support, the number of errors decreased.

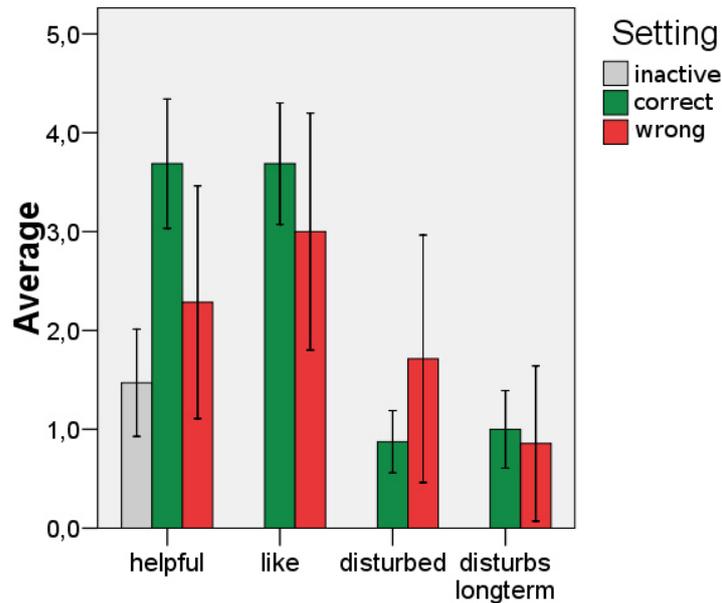


Figure 2.14.: Results from the questionnaire.

2.3.2 Applying AUGUR to Make Existing Web Applications Context-aware

The user study helped to answer the first question raised in the beginning of this section. However, the second question, whether it is technically feasible to introduce such extensive UI manipulations as the AUGUR suggestions (see Figure 2.15) into an existing Web application in an automatic way, remains open. To clarify this, we applied the AUGUR system to existing Web applications in a proof-of-concept trial: the ticket booking application of the Deutsche Bahn, and a CRM application by SAP.

As the nature of the available context sensors is only known at runtime in the access environment, and the augmented Web applications cannot be aware of the sensors, the necessary modifications have to be performed in the browser. The AUGUR system therefore contains generic code that parses the HTML of the Web application and injects the data from context sensors dynamically discovered from the environment [HSM09].

Using this technique, context-aware suggestions could be used with the existing Web applications, i.e., the ticket booking portal of the Deutsche Bahn, see Figure 2.15 and the SAP CRM on-demand application, which is implemented as a SOA-backed Web application, see Figure 2.15A

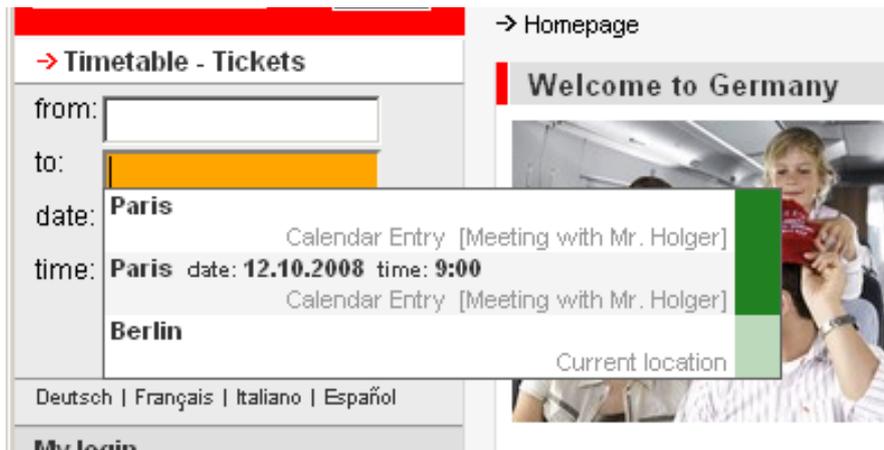


Figure 2.15.: Contextual information injected into the Deutsche Bahn Web Application

AUGUR Architecture

The AUGUR system uses the HTML DOM representation of the Web page in the browser to inject the context-aware features into the application. To be able to do so, all Web applications must be loaded through the AUGUR proxy that injects a small snippet of JavaScript into every loaded Web page. Besides changing the UI, this snippet also communicates with the AUGUR proxy via HTTP, to get the values from the context sensors.

This setup requires a proxy server to be installed on the same machine as the Web browser. Furthermore, the browser must be configured to use this proxy server for all Web applications, where context-aware support is desired. The proxy server communicates with the sensors using the MundoCore middleware [AKM07]. Thus sensors were dynamically discovered, forming a federated access environment.

2.3.3 Conclusions

The experience from the AUGUR system shows that it is technically feasible *and* desirable from a usability perspective, to integrate context-awareness into the access system of Web applications. This integration can be done at the level of the Web browser, without requiring changes on the side of the Web application. This approach has several advantages:

- it is easy to extend. New sensors and also new forms of presentation (besides suggestions and prefilling) can be integrated without changing the application. The AUGUR system is therefore an open adaptive system [OGT⁺99].

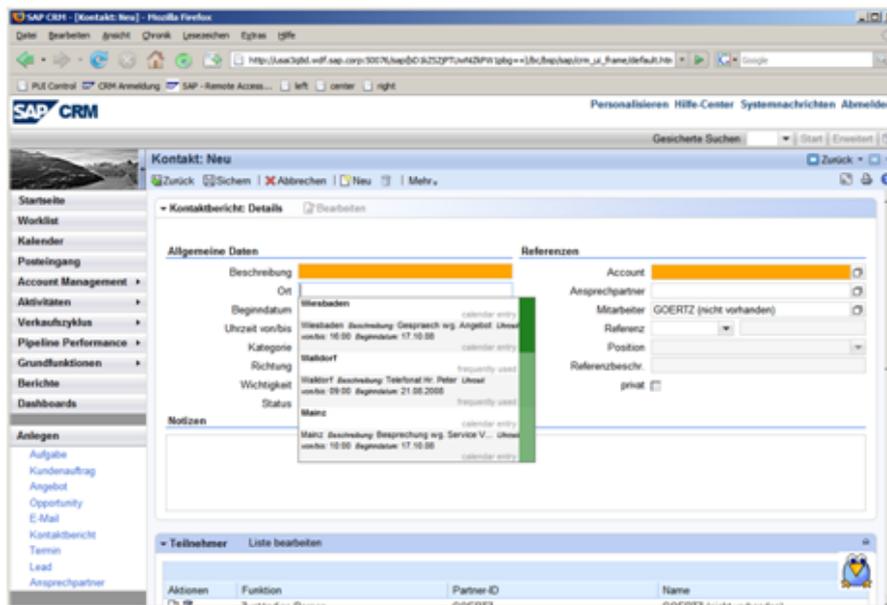


Figure 2.16.: Contextual information injected into an enterprise CRM application.

- the expressiveness of HTML is sufficient to implement UIs for many business applications, and it is sufficient for realizing context-awareness at runtime.

From the experience with the AUGUR system, we conclude that context awareness should be a general feature of a post-desktop access system, not of any particular application accessed in this environment.

2.4 Requirements for a Post-Desktop Access System

The Web access system is limited with respect to support for post-desktop access environments. In the AUGUR case study, we found that these limitations can be overcome for one specific aspect of post-desktop access environments, i.e., the handling of context information to support system use, by extending the Web browser. In this section, we will derive general requirements for an access system suitable to support all aspects of post-desktop access environments. These requirements are based on the findings from our case study. In Section 2.5, we will discuss the relation between the requirements introduced in this thesis and other sets of requirements from the literature.

2.4.1 Existing Web Applications

We require that an access system for post-desktop access environments should remain compatible with existing Web applications. While this is an arbitrary constraint, it makes a lot of sense for the following reasons.

- Web applications already meet many of the requirements postulated in the literature. This close match was achieved not by good up-front design, but rather through iterative refinement and improvements coming from practical application. This is in contrast to many systems proposed in the academic literature that suggest a clean-slate approach yet lack robust implementations. The Web access system can be considered as much more mature and is thus a good stepping stone for further exploration.
- Second, any solution allowing the reuse of the huge amount of existing Web applications is of much higher value than a solution requiring reimplementing of existing applications. Solutions that extend infrastructure components like the Web browser are of more value than solutions that require changes on behalf of the application developer.

Thus, we formulate a first requirement.

Requirement R1 (Reuse of Existing Web Applications)

Existing Web applications shall remain usable with the post-desktop access system, without any implementation changes.

This implies that the new access system stays compatible with existing Web access system components at the protocol level. The AUGUR case study showed that this can be achieved by augmenting the Web browser.

2.4.2 Supporting Existing and Novel Access Environments

The problem of instantiating a UI for a potentially federated post-desktop access environment can be separated into the the following two sub-problems [CBA⁺07, CLC05]:

- UI (re-)molding and
- UI (re-)distribution

Thereby, remolding refers to both, structural modifications of the UI, e.g., adding and removing UIDL elements during the process of rendering and the rendering itself, e.g.,

interpretation of the UIDL for concrete interaction devices found in the access environment. UI distribution is the process of allocating different parts of the UI to the different components in a federated access environment. Distribution goes hand in hand with remodeling, as once a UI part is distributed to a component, the component has to render the UI part, thereby remodeling it. Note that distribution may also refer to using input data from different components in the access environment and does not mean the distribution of output alone.

Federated Access Environments

We recall that post-desktop access environments do not contain a standard set of interaction devices as was the case with desktop access environments. Mobile devices do contain a fixed set of interaction capabilities but they should be augmented by using interaction capabilities from the environment to increase application usability [BM05a, MMBE00, MNWM04, NMH⁺03]. We conclude that the normal post-desktop access environment is a federated access environment.

We state that support for federation should be handled within the access system and not at the level of the operating system. The reason is that this allows use of the UI runtime model for integrating the components in the federation.

Requirement R2 (Federated Access Environments)

The post-desktop access system shall be able to integrate arbitrary resources that are connected via a middleware.

Remolding and Redistribution Capabilities

Components in the federated access environment can be coupled to each other in various ways. If one component represents a single interaction modality, the CARE model [CNS⁺95] can be used to describe the coupling options. However, in reality a single modality is often realized by multiple components, e.g., the pointing modality uses a pointing device and a screen showing the pointer. The AUGUR system showed how federations of context sources can be used in the access system.

Non-standard resources found in post-desktop access environments can only be supported adequately if the processing of input data and the output and structure of the UI can be modified at the same time.

For example, to support recognition-based interfaces [MHA00], disambiguation options need to be displayed to the user (output / structure modification) letting her distinguish

between voice recognition hypotheses (input handling). As the concrete access environment is only known at runtime, the modifications can only be performed at runtime. This is also backed by our experience with AUGUR, where supporting context awareness required extensive manipulation of the output presented to the user, e.g., suggestions had to be introduced to the UI. At the same time, the AUGUR system had to process input from various context sources in the access environment.

Following our experiences with the AUGUR system, we require that the post-desktop access system should be open adaptive [OGT⁺99], i.e., the (business) user must be able to deploy new strategies for distributing the UI and for remolding the UI to match particular post-desktop access environments. Therefore, we introduce the following two requirements:

Requirement R3 (Changing the Input Processing)

The post-desktop access system shall enable the user to introduce changes to the processing of input data based on the resources found in an access environment without recompiling any applications.

Requirement R4 (Modification of Output)

The post-desktop access system shall enable the user to introduce changes to the output based on the resources found in an access environment without recompiling any applications. These changes include changes at the level of the UIDL, e.g., introducing new UIDL elements.

2.4.3 Mitigate User-Perceived Latency

Rich redistribution and remolding capabilities in an access system help to increase application usability. For example, the context-aware support implemented in AUGUR enabled users to accomplish their tasks faster, and thus increasing the usability of the CRM application. Thereby, the improved usability could be observed while interacting with a single Web page. However, usability of Web applications, especially in mobile access environments, is also adversely affected by the latency perceived by the user [PK08]. High latency not only reduces the efficiency for interacting with the application but also badly affects the user's satisfaction with the system [PK08].

An access system suitable for post-desktop access environments should thus incorporate mechanisms that address user-perceived latency. However, like the remolding and redistribution strategies, the latency reduction must be done in such a way that it does not complicate the development of applications.

The latter is especially important for interactive business applications, which are frequently implemented and deployed by business users instead of expert programmers. Web browser and Web server implementations hide the network communication from the developer, and thus help to keep development complexity to a minimum. The low bandwidth of mobile network connection in mobile access environments leads to long download times if the granularity of Web pages is too high, i.e., the Web pages incorporate results from many complex SOA services in the back-end. This deteriorates the user experience of Web applications. On the other hand, very fine-grained Web pages will lead to a much higher rate of network access. For each such access the user will perceive the high latency of the mobile network connection, which again results in a bad user experience. Thus, the developer of an interactive business application suitable for mobile access environments must exert great care to define Web pages at the right level of granularity, which distracts him from concentrating on business aspects. A suitable access system for general post-desktop access environments should address this by utilizing appropriate latency-reduction techniques.

Requirement R5 (Application-Independent Reduction of User-Perceived Latency)

The post-desktop access system shall reduce user-perceived latency without introducing additional complexity for application development, compared to the existing Web access system.

2.4.4 Management and Configuration Operations

Web applications are often said to require *zero installation*. However, technically there is not much difference between downloading an HTML document from the Internet plus interpreting it in a browser and, e.g., downloading a Java program plus interpreting it in a Java Virtual Machine (VM). The reason why Web applications are nevertheless perceived as more user-friendly is that the Web browser incorporates a UI for finding, accessing, starting and stopping applications.

Supporting adequate management and configuration operations is a key factor for the success of Web applications. The Web access system comprises methods to access applications, e.g., via the browser address bar or bookmarks, as well as methods to control applications, e.g., via browser tabs. However, in order to support federated access en-

vironments, additional management and configuration operations are needed that are related to the management of the access environment itself. An access system suitable for post-desktop access environments should thus comprise additional management and configuration operations.

The research challenge is to define the necessary primitive management and configuration operations that should be supported by the access system. As one can see from the example of browser tabs that duplicate functionality already available as part of the operating systems window manager, just *supporting* the needed management and configuration operations is not enough. The operations must be provisioned to the user in a convenient way, matching the special situation of post-desktop access environments.

Requirement R6 (Application and Environment Management and Configuration)

The post-desktop access system shall support management and configuration operations, giving the user control over the used applications and components in the access environment.

2.5 Discussion of the Requirements

As the requirements derived in the previous section are only backed by one case study, the question arises whether they are generally justified or whether they are specific to this one application. To answer this question, we compare the requirements introduced above to other sets of requirements from the literature that have been proposed with a similar intent. The results of this comparison are presented in Table 2.1 and Table 2.2.

2.5.1 Requirements of Trewin et al.

In [TZV03], Trewin et al. present requirements for UIDLs suitable for universal access. The requirements for universal access overlap with those of post-desktop access environments, as in both cases non-standard interaction devices need to be supported by the access system. Although Trewin et al. explicitly target UIDLs with their requirements, many of the requirements have an impact on the whole access system. For example, no UIDL alone can provide *Run Time and Remote Control* — as this is a runtime feature, a whole access system using the UIDL is necessary to satisfy this requirement. Therefore, we treat these requirements as applying to the access system as a whole. To compare the Web access system to these requirements, we treat HTML as its UIDL.

Table 2.1 shows the result of this comparison. The requirements addressed by the existing Web access system are marked in the second column.

requirements from [TZV03]	covered by requirement
Applicable to Any Target	R1
Interface Elements	R1
Separation of Interface Elements from their Presentation	R1
Presentation-Related Information	R1
Run Time and Remote Control	R1 (with AJAX)
Simple	(R1), R2, R3, R4
Applicable to Any Delivery Context	(R1), R2, R3, R4
Personalizable	R3, R4
Flexible	R3,R4
Extensible	R3, R4
-	R5
-	R6

Table 2.1.: The table shows how the requirements elicited in [TZV03] map to the requirements used in this thesis.

Applicable to Any Target: *Applicable to Any Target* means that UIs for any application can be described. The sheer amount of existing Web-based interactive applications is evidence that this is the case for all practical purposes.

Separation of Interface Elements from their Presentation: The prevailing opinion in the literature sees HTML as a layout-dependent format only suitable for the FUI level [CCT⁺03, PSMM08]. However, this is actually not the case. The ACID tests [Pro09] define a standardized layout, but browser implementations can and do depart from it in arbitrary ways to better match the access environments at hand (e.g., mobile Safari on the iPhone and `lynx` in desktop access environments). Therefore, the Web access system already *separates interface elements* (specified in HTML documents) *from their presentation* (created by the browser). However, HTML and especially CSS allow the application developer to include detailed *presentation-related information* with the UI that acts as hint for the browser.

Interface Elements: *Interface Elements* means that all UIDL documents can be interpreted without any knowledge about the application they belong to, which is the case for HTML documents that are interpreted in a Web browser.

Run Time and Remote Control: Traditional, turn-taking-style Web applications did not allow for *Run Time and Remote Control*. However, modern Web applications by default use AJAX technology to achieve this functionality. At the lowest level, communication is still based on client requests, but AJAX frameworks like COMET [Rus06] provide server push functionality to applications.

Simple: The *Simple* requirement means that it should be easy to render the UIDL. For HTML, the UIDL of the existing Web access system, this is apparently the case. At least implementing a browser for desktop and mobile access environments has been simple enough to permit high-quality implementations of browsers.

Applicable to Any Delivery Context: For other post-desktop environments this is not the case, indicated by the parentheses around the requirement in the table. Likewise, HTML is currently already *applicable to many delivery contexts*, whereas a delivery context is equivalent to an access environment. This means, however, that with the current access system they are not applicable to *any* deliverable context; therefore, this requirement is again put in parentheses.

Although also not mentioned by Trewin et al., applicability to any delivery context should also include techniques to reduce *latency*. Even though the employed UIDL has an impact on this (i.e., verbose UIDLs would add to latency), the impact is only minor.

Personalizable, Flexible and Extensible: The requirements *Personalizable*, *Flexible* and *Extensible* all imply that adaptation of the UI (input and output) happens at runtime, albeit for different reasons. Using the examples from [TZV03], this entails translating the UI to the user's language (*Personalizable*), adapting to the user's level of experience (*Flexible*), or incorporating knowledge which was not available when the UI was developed, e.g., about novel interaction devices (*Extensible*).

As one can see, many of the requirements elicited in [TZV03] are already covered by the existing Web access system. However, we conclude that the five latter requirements are only partly addressed, if at all, by the existing Web access system. We think that these requirements cannot be addressed at the level of the UIDL, but must be tackled at the level of the browser implementation, e.g., supporting more flexible adaptation of output and processing of input within the browser.

2.5.2 Requirements for Personal Unified Controller (PUC)

Nichols et al. postulate requirements for an access system for appliances in non-desktop settings, e.g., a living room [NMH⁺02]. Thus, applications for controlling appliances are considered instead of interactive business applications. However, the environment in which they are accessed are post-desktop environments, as they are dynamic in nature. Table 2.2 compares the requirements of [NMH⁺02] to the Web access system.

requirements from [NMH ⁺ 02]	covered by requirement
No Specific Layout Information	R1
Two-Way Communication	R1 (with AJAX)
Dependency Information	R1
Sufficient Labels	(R1), R3, R4
Simultaneous Multiple Controllers	R2
Actions as State Variables and commands	-
Shared High-Level Semantic Knowledge	(R1), -
-	R5
-	R6

Table 2.2.: The table shows how the requirements elicited in [NMH⁺02] map to the requirements used in this thesis.

No Specific Layout Information: As discussed above, the HTML documents used in the Web access system may contain specific layout information, but these do not have to be used by the browser. However, a browser can also generate interfaces from HTML without specific layout information.

Two-Way Communication: *Two-Way Communication* can be realized in the Web using AJAX techniques.

Dependency Information: *Dependency Information* means that the UIDL provides ways to express relationships among UI elements, e.g., for defining groups of elements that are active or available together. HTML provides the `fieldset` tag for exactly this purpose.

Sufficient Labels: HTML and browsers can show a number of simple text labels, e.g., as tool tips. However, the *Sufficient Labels* requirement also means being able to render these labels in the voice modality. Therefore, flexible output adaptation also needs to be applied to cover this requirement in addition to the existing Web access system.

Simultaneous Multiple Controllers: Being able to employ *multiple controllers* in parallel is deemed important by Nichols et al. The intention is to synchronize these controllers inside the access environment, i.e., the access environment is a federation of multiple controllers. Nichols et al. only consider parallel use of the controllers, corresponding to the *equivalence* and *redundancy* types of coupling from the CARE properties [CNS⁺95]. This requirement is clearly not met in existing browsers.

Actions as State Variables and Commands: The requirement of representing *Actions as State Variables and Commands* does not make sense for interactive business applications. Unlike appliances, the business processes implemented using interactive business applications cannot be described with just state variables, as they are more complex, requiring completely different interface structures in different steps. Adopting this requirement for an access system would make it unusable for interactive business applications.

High-Level Semantic Knowledge: Nichols et al. require that a UI description should convey *High-Level Semantic Knowledge* about the application. However, the examples used in [NMH⁺02] are not very high-level (dates are entered in a special format). At this level, the requirement is already covered by existing HTML input tags that allow the developer to specify the `date` type among other types. Requiring that the UI description should contain application domain knowledge, which is then used for rendering, does not seem reasonable. It would make implementation of renderers immensely complex (cf. requirement *Simple* of [TZV03]). And Nichols et al. state:

Despite all of the previous requirements, we must concede that it is impossible to encode all the information into an appliance specification that a human would use to design an interface.

Thereby, appliance specification refers to the UI description of an application.

2.5.3 Conclusion

As conjectured, the Web access system addresses many requirements that were proposed for post-desktop access systems. Consequently, Web applications are the prevalent technology to make interactive business applications accessible in post-desktop environments. However, the Web access system as it exists today is limited with respect to post-desktop access environments: The way Web applications are rendered in the access environment is completely static. The browser component of the Web access system provides little support for adapting the interaction with the Web application to the dynamics and variety of resources encountered in federated post-desktop access environments.

As we can see from the review of existing requirements, Web applications do not support all post-desktop access environments (*Simple* and *Applicable to Any Delivery Context* from [TZV03] not met and *Sufficient Labels* from [NMH⁺02], respectively). This is due to the static nature of existing browsers, which include hard-coded optimizations for a certain type of access environment. This can be overcome by fulfilling requirements 3 and 4 in the new access system.

Requirement 6 has not been considered, neither by Trewin et al. nor by Nichols et al. We think that this is a deficit in those sets of requirements, and that this requirement is still valid and important. Leaving out such a requirement is especially interesting for Nichols et al., because the goal of the Personal Unified Controller (PUC) system is to facilitate

Requirement	Description
R1	support reuse of existing Web applications
R2	support federated access environments, give access to the runtime UI model to multiple nodes
R3	allow modification of the the handling of input data in the access environment
R4	allow modifications of output and structure of the UI in the access environment
R5	reduce user-perceived latency, especially in mobile post-desktop access environments
R6	support a functionally complete set of management and configuration operations, suitable for federated post-desktop environments

Table 2.3.: Overview of the requirements for improving the Web access system.

interaction in environments containing a lot of different appliances. Therefore, choosing which appliance to control or commissioning new appliances in the environment seems an obvious problem that should have been tackled by the system.

By contrast, Web applications include such mechanisms and are key to their widespread adoption. Therefore, this feature should be present in future access systems for interactive applications, where it needs to be designed in a way to match the specifics of post-desktop access environments.

2.6 Chapter Summary

In summary, we state that an access system meeting our objective should meet the six requirements shown in Table 2.3. The requirements in this thesis show considerable overlap with the requirements used by other researchers, as can be seen from the comparison in Table 2.2 and Table 2.1. This indicates that this set of requirements is reasonable and valid. The requirements of [TZV03] are completely covered, meaning that any access system suitable for interactive applications in post-desktop environments could also be used for universal access. Both, Trewin et al. and Nichols et al., do not consider management and configuration operations, especially their nature in post-desktop environments. However, these are important aspects of the very successful Web access system.

The analysis also shows that many of the requirements of Trewin et al. and Nichols et al. are already covered by the existing Web access system, corroborating our hypothesis that the Web should rather be extended than replaced in a clean-slate approach. Neither Trewin et al. nor Nichols et al. deal with applications distributed over the network. Therefore,

the problem of roundtrip latency does not affect the interaction and no means to counter this latency are needed.

The requirement R6 is also not covered by the existing sets of requirements. Although the requirements for Nichols et al. cover distribution of the UI to multiple devices in a federated access environment, they do not foresee any possibility for the user to control this.



Chapter 3

State of the Art

This chapter reviews state-of-the-art approaches and compares them to the requirements derived in the previous chapter. Existing approaches can be classified according to the taxonomy shown in Figure 3.1.

The existing approaches can be loosely separated into two categories. Approaches in the first category investigate how application development, especially UIDLs, needs to be improved to support post-desktop access environments. The second category comprises approaches that focus on the construction of post-desktop access environments.

Approaches in the first category can be differentiated further into approaches focusing on the remolding of UIs and those focusing on the redistribution of the UI. Approaches falling in the former category mostly rely on models during UI development and runtime. They will be discussed in Section 3.1. Approaches for redistribution of UIs typically maintain an abstract representation of the UI at runtime. This abstract representation can be used by the access system to distribute the UI to the devices in the current access environment. We discuss these approaches in Section 3.2.

Approaches from the second main category support the construction of new post-desktop access environments. This category can be further separated into approaches that focus on using existing applications in these access environments and approaches that assume that application development happens after the access environment has been set up. The

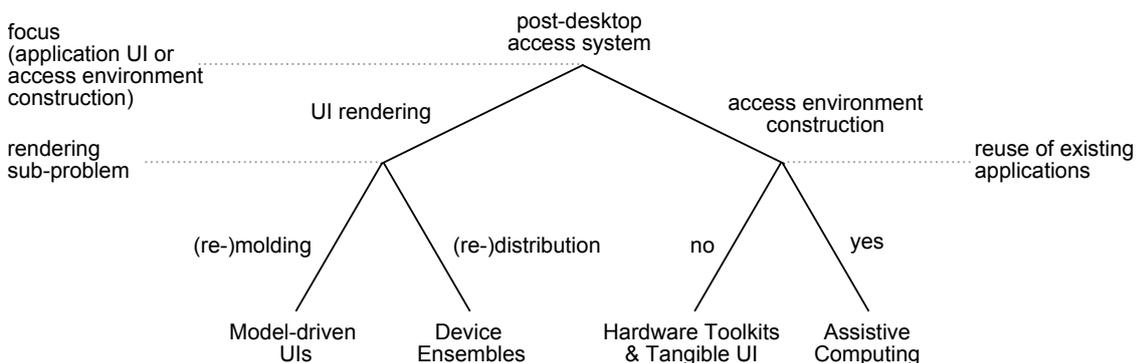


Figure 3.1.: Taxonomy for systems from related work.

former approaches are mainly researched in the context of assistive computing (discussed in Section 3.4), while the latter come from research on tangible interfaces and interfaces comprising novel hardware components (discussed in Section 3.3). In Section 3.5, we discuss some extensions to the Web access system that do not fit into the categorization above.

3.1 Model Driven User Interface Development

The proliferation of mobile devices, led to a broad variety of access and operating environments for applications. Maintaining different versions of the application UI for each target device quickly becomes tedious. This problem is amplified in post-desktop access environments, as the different types of mobile devices are just a subset to the set of all potential post-desktop access environments.

MDUID approaches address this problem by introducing an abstract UIDL. The developer specifies the UI at the Abstract User Interface level. The CUI for the target device is then generated with the help of tools.

3.1.1 Early Model-Driven Approaches

Representative examples for the first generation of MDUID approaches are the Teresa [MPS03] and ARTStudio [CCT01] system. The behavior of Teresa and ARTStudio is comparable to a compiler with different back-ends. The generated FUI directly uses available interaction capabilities in an access environment, just as the code generated by a compiler back-end directly runs on the target platform. This means the application developer can easily generate FUIs for different access environments from a single AUI-level description, as long as the target access environment is supported by the transformation machinery. There are other systems that work in the same way but use different source and target models.

Both systems support the Cameleon levels [BDB⁺04] introduced in Section 2.2.3.1. The automatic compilation is accompanied by extensive options to introduce special behavior for certain targets, which are then maintained even when the Abstract User Interface description is changed. The resulting application including the FUI is copied to the device where it is intended to run and execute. How this deployment is done is beyond the scope of the systems. Requirement R6 is partly addressed by ARTStudio, as the system provides a a UI for configuring the UI to different devices at runtime. ARTStudio does not, however, provide mechanisms for starting or stopping applications in the access environment.

Another drawback of early model-driven approaches is that once the FUI is generated, it can no longer adapt to changes in the access environment. Thus, neither requirement R3 nor R4 is met. To adapt the output presented to the user or the processing of input to

a novel type of access environment, a formal description of the novel access environment type is required. In addition, implementations specific to this access environment for all transformation steps from AUI to FUI are required before the access environment is supported. Requirements R3 and R4 are thus not addressed, as the user cannot provide such new transformations without running the whole transformation process again. As the FUI is intended to run on a single device, requirement R2 is also not addressed within these systems.

3.1.2 UI Plasticity and Models at Runtime

To overcome the limitations of the relatively static runtime structure of the early MDUID approaches, the concept of UI *plasticity* was proposed in [CCT⁺02]. Plasticity has been defined as the ability of an interactive application to provide a usable UI, even if it is used in an access environment different from the one for which it was originally designed. UIs that show plasticity can adapt to a range of access environments whose parameters are inside a plasticity threshold from the originally intended access system for which they were designed. The findings of [NCM07] show that it is indeed possible to automatically adapt the UI of an application to a different screen size that is not too much different from the screen size the UI was designed for. However, automatic adaptation to more complex access environments is problematic [MHP00].

To realize plasticity in the context of the MDUID paradigm, a model of the UI must be available at runtime, so that the interactive application or the user can select and apply transformations that apply to the access environment at hand.

Cameleon-RT

The Cameleon runtime architecture, called Cameleon-RT, proposed in [BDB⁺04] supports plasticity. The conceptual model of the Cameleon-RT runtime architecture has been successfully applied in the I-AM system [Bal08]. The system supports accessing interactive applications in multi-monitor environments, i.e., access environments comprising multiple screens and a single pointing input resource, e.g., a computer mouse. Figure 3.4 shows an example of a UI generated by this system. The requirement R2 is partly addressed by the I-AM system, as the screens are dynamically discovered from the environment. However, as the resources are only screens, requirements R4 and R3 are not addressed. Requirement R6 is partly addressed by the meta UI described in [SCCF08] which is shown in figure 3.2. However, the meta UI does not provide means for accessing, starting and stopping applications.

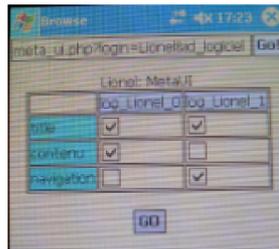


Figure 3.2.: Meta UI for splitting a Web application to different devices from [SCCF08]

Mapache

Another system using models and model transformations at runtime is Mapache proposed in [BPM09]. The application developer can easily specify new refinements and transformation rules for a particular access environment by using a lightweight development environment [BPM09]. These new transformations are then applied to the original AUI model. The Mapache system relies on the MundoCore middleware [AKM07] for connecting multiple devices in the access environment. Thus, requirement R2 is addressed.

However, the structure of the UI cannot be modified at runtime in arbitrary ways, e.g., one interactor cannot be implemented in a way to change the output shown on another interactor. Thus, implementing multitouch input is only supported if the application developer uses the correct interactors, as it is impossible to introduce additional mouse cursors on a screen. In conclusion, requirement R3 is not addressed and requirement R4 is not addressed completely.

The Mapache system provides a configuration UI where the user can influence the remolding. However, this UI does not allow the user to start or stop applications; requirement R6 is thus not fully addressed.

MASP

The MASP environment [BLFA08, FBSA08] follows an approach similar to Mapache (see figure 3.3 for an example UI). The system foresees a predefined set of input types (gesture input, natural language input, character input, and pointing input) and output types (signal output, graphic output, natural language output). Again, the rendering of one device cannot change the output on another device. The rendering is done in a purely hierarchical fashion. Therefore, the system does not fulfill requirements R3 and R4 .

A meta-user interface for MASP, addressing requirement R6, is presented in [RBA09]. It enables the user to deploy applications into the access environment. Also, the connection between the access environment and the application can be configured in the meta-UI by

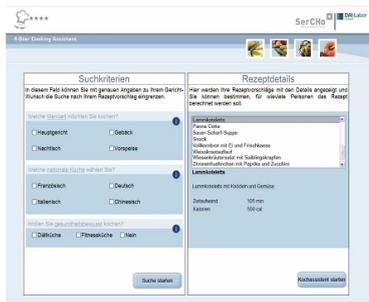


Figure 3.3.: GUI generated by the MASP environment from a AUI.

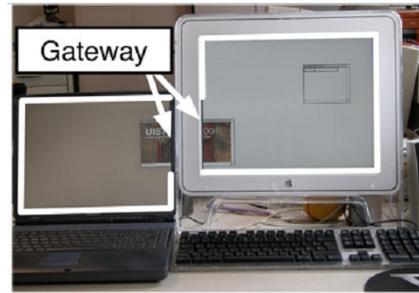


Figure 3.4.: Instance of a *plastic* UI as generated by the I-AM system.

turning a predefined set of three modalities on and off. The meta-user interface also allows the user to split the UI to multiple devices.

COMETS

The COMET system [DCC08] integrates MDUID with interaction toolkits. Unlike other systems using an MDUID COMET can be used to realize interesting post-desktop UIs, e.g., based on OpenGL rendering. In the COMET approach a UI is defined as a graph of COMETS. Each COMET has three facets, called *logical consistency*, *logical model* and *physical model*. The logical consistency facet specifies the task or set of tasks the user can perform with this COMET. In [DCC08] the example of ‘change to next slide’, ‘change to previous slide’ is used. The logical model implements the tasks specified in the logical consistency facet. It appears that the logical model is a class implementing the interface specified by the logical consistency. Finally, each logical model is bound to one or more physical models, whereby the physical models differ by the platform they require. For example, a logical model could have distinct physical models for Java and HTML. COMET then provide certain advanced mechanisms to the developer to specify the relationship between different COMETS in the graph governing how update and input events propagate in the graph of COMETS.

COMET comes with a large set of physical model implementations, including HTML, Java and OpenGL. However, it is not described in [DCC08] how the end-user could add new physical models to an existing logical model. Although, as the physical models are associated with a single logical model, modification of the output spanning multiple COMETS is not supported. Therefore, the system does not fulfill requirements R3 and R4.

COMET does not specify a meta UI and rather relies on the meta UI provided by the platform underlying the different physical models, e.g., it relies on the window manager for the Java physical model. Implicitly the user can choose the physical model she wants to use by accessing the application with different devices, however, she cannot do this explicitly in a meta UI. Requirement R6 is thus not satisfied.

The COMET system does not deal with latency, as all parts of the application are considered to be available locally in the network.

Although not explicitly discussed in [DCC08], COMET can deal with federated access environments, e.g., rendering parts of a slide show presentation application UI on a PDA, while other parts are rendered on a table top computer. Thanks to the graph-based structure, different COMETs can be assigned to different devices. It is however not clear from the paper whether just the rendering, i.e., the physical model is actually distributed to the device or the whole COMET.

End-user Scripting for Web-based Systems

As discussed earlier, the Web access system is similar to many model-based approaches; accordingly in this section we discuss some systems that make use of the DOM of the HTML page as a model at runtime.

Interactive web applications also share some drawbacks with MDUID approaches. Namely, developing a browser, which is the *target platform* for the most specific PSM in the web infrastructure, is difficult. Existing Web browser do not support adaptation to changing access environments. For example, the iPhone browser presents HTML interfaces using exclusively the resources provided by the iPhone device itself. Adapting it to the preferences of the user or making use of other interaction devices nearby, like a large screen in the vicinity of the user, is not supported. Thus requirements R2, R3 and R4 are not supported.

These limitations are partially addressed by end-user scripting frameworks for the Web, such as [GRE09, BWR⁺05, LHML08]. They enable the user to use the rich HTML Application Programmer Interface (API) for modifying the output of the Web application in the browser at runtime. This can be used to introduce support for characteristics of the access environment at hand. The output can be modified on a per User Interface Description (UID) element basis, e.g. by iterating over all HTML elements of a certain type, but also across different elements, e.g., by adding new HTML elements. Such modifications can be controlled through the meta UI of the browser by the end-user. Requirement R4 is thus satisfied. However, the processing of input data cannot be changed, thus failing to address requirement IN. The meta UI of existing Web browsers does not let the user access and configure the access environment, thus requirement R6 is not met.

MARIA

For completeness, we also mention the MARIA system [PSS09]. MARIA builds upon the Teresa approach by maintaining models of the UI at runtime. However, the focus in the use of these models to enable more lively HTML GUIs, using AJAX features, as opposed to

simple form-based Web interfaces. It does not address any further requirements introduced in Chapter 2 compared to the Teresa system.

3.2 Access Systems for Multiple Devices

The focus of the systems discussed above was on using models for describing and representing UIs at designtime and runtime. More specifically, these models are used for remolding the UI. The distribution of the UI to different devices in the access environment can also benefit from such high-level information. However, the problem of distributing the UI to multiple interaction devices connected via a network has been approached without using models from the MDUID approaches for representing the UI. The systems we discuss in this section emphasize requirement R2, and rely on a middleware for connecting the different devices in the access environment, e.g., MundoCore in [BM05a] and UPnP [UPN] in [VC04a].

3.2.1 Homogeneous Device and Multi-Monitor Access Systems

One particular instance of federated post-desktop access environments are multi-monitor access environments. Such access environments comprise multiple displays that are connected to the access system via a networking middleware. One or more input devices are multiplexed through the same middleware to provide input capabilities [BF05,BF07]. Multi-monitor capabilities are also offered by the I-AM system.

In this particular class of federated post-desktop environments, distribution of the UI to the different output devices can be done on the pixel level instead of the semantic level [BDB⁺04], i.e., without any high-level knowledge on the UI.

Like multi-monitor middleware for output, middleware for flexible input redirecting, e.g., PointRight [JHWS02], allows users to control single applications through multiple pointing input devices. CPNMouse [BLMA⁺01] even supports an arbitrary number of pointing devices, regardless of whether the underlying operating system supports this feature.

The access system provides the abstraction of a single large screen for the access environment. No adaptation of the output takes place. Therefore, these solutions only apply support for classic desktop GUIs interfaces; requirements R3 and R4 are not supported by these systems. The support for requirement R6 is also restricted to the functionality already known from desktop access environments.

3.2.2 Access Systems for Inhomogeneous Devices

Multi-monitor access systems support the distribution of a UI to multiple homogeneous interaction resources, i.e., multiple screens. Several other approaches have been conceived to address the problem of distributing the UI among *inhomogeneous* devices. They make use of abstract information about the UI structure, e.g., derived from the UIDL, to guide the distribution process. Interaction resources receive a part of the UIDL and render it for the device at hand.

Federated Devices

The concept of *federated devices* [BM05a] proposes to split an XForm [xfo09] automatically across multiple devices according to predefined patterns. The devices to which the interface is distributed are discovered automatically, taking the location of the user into account. Patterns include rules for separating, e.g., the control part and the display part of a slide-show application. The control part is rendered on a Personal Digital Assistant (PDA) carried by the user, and the slides are shown on a large wall-mounted display. If several large displays are available, the most suitable one is chosen based on the location information.

Changing the output to the user in a way spanning multiple devices is not possible. Parts of the UI can only be moved or replicated on different devices; the rendering of the part cannot be changed in a way that affects multiple devices at once. The processing of input data cannot be changed either. It is fixed in each interaction device. Thus, requirement R3 is not satisfied and requirement R4 only to a limited degree (no manipulation spanning multiple UIDL subtrees).

Pebbles

The Pebbles project [NMH⁺03] developed a system to use a hand-held device for remotely controlling appliances and applications running on other devices, e.g., the Personal Computer (PC). The requirements underlying this system were discussed in Section 2.5. The Pebbles system is highly optimized for remote controlling, i.e., the UI part rendered on the hand-held device contains a simplified version of a complex application or an exact mirror of the appliance UI. Complex interactions between the different devices, e.g., altering the output of the original UI once a hand-held is present, are not supported by the system. Requirement R4 is not addressed completely. The output can only be adapted on a per-element basis by changing the rendering function of a device. As with federated devices, Pebbles does not address requirement IN, as the processing of input data is fixed within each device.

Pebbles also supports another kind of federation, where the interface to multiple appliances dynamically discovered from the environment is *joined* into a single consistent UI [NMH⁺03]. The same is done in ICrafter [PLF⁺01]. This is exactly the opposite of federated access environments, where multiple devices and resources are used to interact with a single UI.

XWeb

The aforementioned systems relied on system-driven splitting of the UI. A different approach has been proposed for the XWeb system [OJN⁺00]. With the extensions described in [ONP01], the user can combine different devices into one access environment and define the distribution of the interface parts to these devices. The UI model used by XWeb is a hierarchical tree of data objects, which simplifies synchronizing the interface across different interaction resources but is not as rich as HTML. Furthermore, as a variant of HTTP, a client-server protocol, is used for communication protocol, modifications of the output are not possible from the application without a request from the client.

Interestingly, the meta-user interface is not a GUI as in [RBA09] or [Cou07], but a tangible meta-user interface. Communication between the different devices in the access environment is performed solely via the application data tree stored on the server. It is impossible to change the appearance of the UI on one device because another device becomes available. Therefore, requirements R3 and R4 are not satisfied.

XWeb specifically tries to leverage the principles of the Web access system, by modeling the interaction between the client devices and the server in a way closely following the HTTP protocol.

Dygimes

A similar approach is proposed by [VC04a] and earlier [CLC05], where a UI is distributed to multiple devices. The splitting of the UI to the different components in the operation environment is again automatically handled by the system. Devices are discovered using Universal Plug and Play (UPnP). Dygimes is important, as it combines model-driven approaches with federation over a network middleware. The system uses information from the task and concepts layer in the distribution, so that related UI elements go to the same device.

For the rendering, Dygimes uses a very flexible mechanism based on constraint solving [BBS01, LTVC06], making output adaptation for GUIs easy. Still, influencing the rendering on one device as part of the processing of input on another device is not possible; requirement R3 is thus not satisfied.

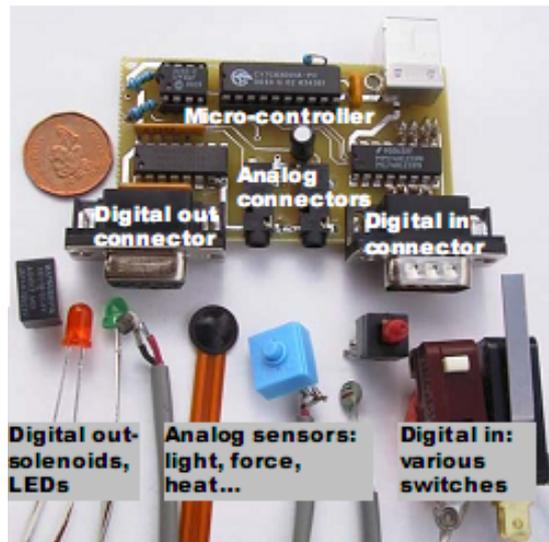


Figure 3.5.: Examples of Phidgets hardware

3.2.3 Discussion

All UI-splitting approaches discussed in this section distribute the output to the user on a per-UIDL-element basis. Integrating different input sources and altering the processing of input data is not supported, limiting the possibilities of creating novel types of access environments. For example, XWeb distributes different parts of the data tree to different devices, where they are rendered according to the device's rendering engine and input is handled by the input resources statically attached to the rendering device. Input patterns spanning multiple devices, for example, controlling a cursor pointing on one device from another device are not foreseen.

XWeb is the only system that provides a solution to the problem of deploying applications, and it reuses and extends the approach of the Web access system for this. However, it is unclear how an application is initially accessed to store its URL. Presumably this is done in the richer graphical meta UI which is just mentioned and not described. The obvious solution would be to use an approach similar to the address bar in Web browsers.

3.3 Toolkits for Post-Desktop Hardware

Systems supporting device federations address one aspect in which post-desktop environments differ from desktop environments, i.e., the way they are constructed. However, desktop and post-desktop access environments differ in another aspect: post-desktop access environments contain a host of different interaction resources that are not found in desk-

top access environments. Examples of such resources include context sensors, which have been already discussed in the AUGUR case study. In practice, accessing this non-standard hardware is difficult.

This is not only, as the cliché goes, a ‘small matter of programming’.

[KLLL04]. Therefore several systems emerged that allow application developers to use these hardware resources in an easy way.

3.3.1 Sensor Hardware Abstraction

The first step in supporting novel in- and output hardware is the access to the hardware on the driver level. Implementing system drivers for novel hardware is difficult, which is the essence of the above quote. This can be addressed by providing frameworks and ready-made hardware components that lend themselves for easier integration with access systems. We discuss several such approaches in this section.

Phidgets

One of the most popular systems are Phidgets (shorthand for ‘physical widget’), introduced in [GF01]. Figure 3.5) shows several Phidgets hardware devices. The Phidgets library wraps the low-level communication with these hardware devices, and thus makes it convenient to use them from applications. However, for this to happen, the application must be written from the start with the target access environment in mind. As a result, requirements R3 and R4 are not fulfilled with such an approach. Phidget hardware sensors can be automatically discovered when they are connected to a Phidgets system. However, they are not connected via a communication middleware, but through a wired connection. Thus, Phidgets are not fit to create a federated access environment and do not address requirement R2.

Phidgets can be used with existing applications, using screen scraping or ‘snarfing’ [MMBE00, GB02]. This approach only allows changes in the processing of input data but does not allow modifications of the structure of the UI. Thus requirement R4 is not supported.

iStuff

Another toolkit for abstracting from hardware devices is iStuff [BMRB07]. Using the PatchPanel [BSF04] and the EventHeap middleware [JF02], the iStuff devices can be mapped to existing applications in the iRos system [PJKF03], thereby supporting federated access environments. Hardware devices are connected to EventHeap using a

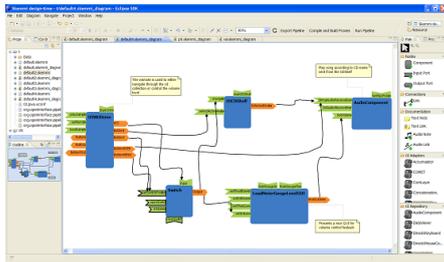


Figure 3.6.: OpenInterface operation environment configuration tool.

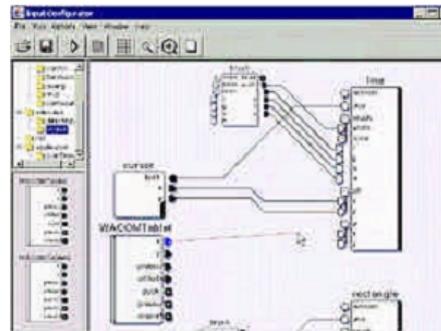


Figure 3.7.: ICON operation environment configuration tool.

proxy approach, where the proprietary messages from the hardware are translated into EventHeap tuples.

Application UIs and interaction devices are completely decoupled — they only exchange tuples via the EventHeap. Thus, there is no meaningful way to alter and adapt the output produced by an application, and the reaction to the tuples is completely at the discretion of the application. There is no way to modify the output in a structured way; requirement R4 is not addressed. Requirement R3 could be addressed by rewriting EventHeap tuples. The system does not support a meta UI.

3.3.2 Input Data Processing

The toolkits discussed above target the integration of single hardware components. Especially for supporting novel input components such approaches can be of great help. However, the raw low-level data provided by these systems is rarely useful for interacting with applications. On top of the access to the hardware, a basic framework for processing the sensor input is needed to make sense out of the data and to turn them into means for meaningfully interacting with applications. We discuss frameworks supporting the processing of input data in this section.

Papier-Mache

Papier-mache [KLLL04] supports the creation of tangible UIs. There, the level of abstraction is on a level above the one provided by Phidgets. The developer can specify which type of sensor hardware should be used, e.g., RFID or computer-vision-based sensing. The Papier-mache platform automatically discovers a suitable sensor (which could be a Phidgets sensor) and connects to it. This can be a local sensor or one from a federated

access environment. From this point on, Papier-mache generates events representing the addition, updating, and removal of objects from a sensor's view.

The developers of Papier-mache analyzed several existing applications using tangible interfaces and conducted interviews with developers of such interfaces as well. One result of this analysis was that an event-based model is more suitable for processing input in tangible computing access environments compared to the widget-based model employed for GUIs.

Combining the output to the user with the processing of input in one widget hinders development of tangible interfaces. On the other hand, the simple events provided by papier-mache suffice to build many interesting tangible applications. For example the marble answering machine of Durrell Bishop [IU97] can be easily implemented. However, the UI is still tightly coupled to the access environment for which it was designed. The papier-mache toolkit does not support modifications of the output or the use of input devices that was not foreseen during development of the application. Requirements R3 and R4 are thus not satisfied.

Pipeline Architecture Systems

The papier-mache toolkit restricted itself to binary events for the presence / absence of a tangible object. More interesting applications can be built when more fine-grained events are provided, e.g., the position of an object in addition to its presence.

Generating events that are meaningful for an application requires processing of the low-level input data. Often, processing steps can be reused for multiple applications and purposes. For example, a low-pass filter is a general-purpose processing step useful for smoothing input signals from any sensor.

There are various commercial [Nat] and academic tools which support the creation of such processing pipelines [pur]. Several of these tools have been specifically designed to support the development of UIs for post-desktop access environments.

OpenInterface

The OpenInterface [LAAVM09] platform is pipeline-based and comprises a range of components useful for developing UIs [ope]. It is not based on a networking middleware, and thus the components have to be deployed in the same node using configuration scripts. Federated access environments are not supported.

Requirement R6 is supported, as far as the configuration of the access environment and its connection to the application are concerned. OpenInterface provides an editing environment [LAAVM09], which although mainly targeting developers can also be used by the end-user (see Figure 3.6).

Like all approaches based on event pipelines, defining and manipulating output in the application UI is difficult. OpenInterface does not contain a rendering component for any UIDL. Thus requirement R4 is not satisfied.

Existing applications cannot be easily reused with OpenInterface. They need to be re-developed from scratch, or at least a custom proxy OpenInterface component must be developed.

ICON

The ICON [DF04] input configurator toolkit follows a similar, pipeline-based approach. Unlike the OpenInterface platform, its focus is not on providing a development environment for new applications, but rather to provide a means for connecting non-standard input resources to existing java GUI applications. However, the output of these applications cannot be modified by means of the toolkit. Again, the focus of ICON is on desktop applications, and thus federated access environments are not supported.

Like OpenInterface, ICON provides an editor (see Figure 3.7), which can be used by the business user to address the configuration of applications and the access environment, which is one part of Requirement R6.

Letras

For the sake of completeness, we mention Letras [HSSM10], which is a pipeline-based framework for supporting pen-and-paper post-desktop access environments. Unlike the other pipeline-based approaches it supports federation, as the different pipeline stages can be distributed in the network. However, the scope of Letras is restricted to pen-and-paper applications, so it supports neither modifications to the processing of input nor output modification.

Summary

Hardware toolkits focus on support for the application developer aiding in the development of new applications. Their intent is similar to that of model-driven UI systems, but instead of a declarative UIDL they favor a more procedural, toolkit-style UIDL produced by the hardware abstractions.

This approach seems to be most valid for input, as support for output is treated superficially and is limited to low-bandwidth output such as ambient lights, Light Emmitting Diodes (LEDs), or beeping. This is because the systems use a pipelined event-processing approach which makes it difficult to describe and model persistent, structural output UI elements. This is not a drawback for tangible interaction, as there the output is provided by the physical things making up the application UI.

3.4 Assistive Computing

Assistive computing can be defined as the ‘application of computing and information technology in solving relevant disability problems’ [SIG05]. Theoretical considerations on how assistive computing techniques can help to provide access to interactive applications in a given operation environment are presented in [WM03].

Means for adapting the processing of input data are researched within the context of assistive computing, for example, changing the input from a pointing to a scanning technique [Mac09] which better suits one-switch devices. The special devices used in an access environment are not foreseen when the application is developed. Therefore, assistive computing systems are necessarily open-adaptive from the point of view of the application developer. Assistive computing provides techniques for changing the output of interactive applications as well as for changing their processing of input data.

3.4.1 Non Web-based Systems

Screen magnifiers are an example of output modification which is agnostic to the interactive application at hand and therefore can be introduced in an open adaptive system. However, often better usability of the UI can be achieved if knowledge about the interactive application can be used, e.g., about the location of important buttons on the screen and so forth. This is especially true for adaptation of the processing of input.

Here, two different approaches are used by assistive computing. The first one is to rely on a dedicated assistive computing API [Mic], allowing programmatic control of the UI of an application. This approach is, e.g., used in the [MNWM04] project. However, these APIs do not provide means to modify the output.

SUPPLE

Although adaptations are performed automatically by the assistive computing system at runtime, setting up assistive computing systems requires complex configuration and customization procedures. A notable exception is the SUPPLE++ system [GW04]. It adapts the UI to the characteristics of a single device comprising in- and output capabilities *and* the capabilities of a user (see 3.8 for an example). SUPPLE++ uses an optimization-based algorithm to choose a concrete rendering for each element in the UIDL, e.g., one of the possibilities of drop down-list, radio buttons, and menu for a choice task, and layouts the widgets on the screen, e.g., increasing font and widget sizes to help with bad eyesight. The optimization algorithm minimizes the user’s *cost* of interacting with the interactive application, based on recorded or modeled user traces. With the ARNAULD [GWW08] system, user-specific cost functions can be automatically learned by observing the user

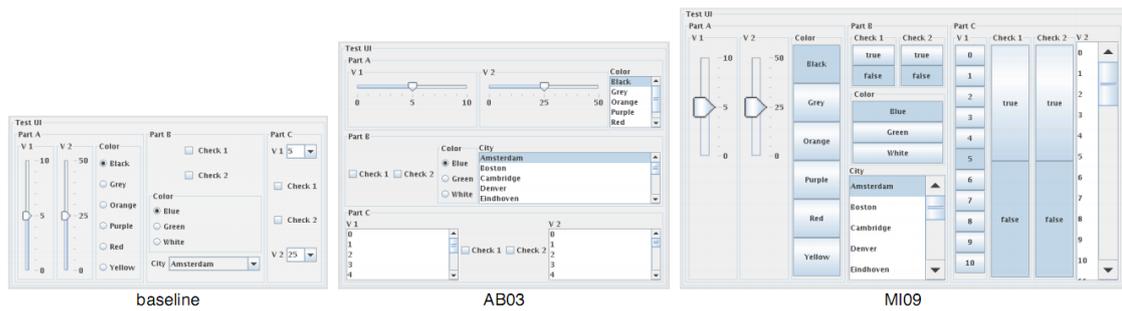


Figure 3.8.: Three variants of a GUI generated by SUPPLE.

performing some standard tasks. In other approaches, the global cost function for all widgets can be specified manually by the user, expressing preference for certain pairs of widgets [YNK09]. As this approach is based on optimization of observable variables (e.g., time on task), it applies even to novel interaction techniques. Thus, the processing of input data is automatically changed based on the *cost* function.

SUPPLE does not rely on an existing API for assistive computing but requires applications to be written in the SUPPLE UIDL, which is available to the SUPPLE system at runtime, i.e., it is used as a model at runtime and could support requirement R4. A technical limitation of the current SUPPLE system is the UIDL employed, which is not sufficiently rich and only supports basic hierarchical organization of UIDL elements. A more sophisticated language such as User Interface Markup Language (UIML) could be used, however. For example, [LVC06] has shown, that constraint-based rendering and layouting can be applied to user interfaces described in UIML.

Requirement R2 is not supported by the SUPPLE system, as the available interaction devices must be statically configured into the system, e.g., all parameters of the cost functions and the supported widget sets must be specified in advance.

IAT

Another approach is to synthesize mouse and keyboard events, i.e., to use the Operating System (OS) window handles as a runtime model. This approach can be used with all UIs written for a certain operating system, e.g., Windows. However, the possibilities of runtime modification of in- and output are more limited. If this knowledge is not provided by the developer explicitly, it can be extracted automatically. One way to gather such knowledge is to rely on the fact that the UI of most interactive applications is implemented using a GUI toolkit, such as Swing or Qt. Thus, one starting point for customizing GUIs to the needs of a disabled person is to change the toolkit code at runtime, e.g., modifying the classloader of the Java VM or using the reflection capabilities of the Java VM as proposed in [CHML06]. However, such modifications are only applied at the level of individual UIDL elements, addressing requirement R4 only partly. Requirement R3 can be satisfied,

as the examples in [CHML06] show, where e.g., one-switch devices are used for input. As an assistive computing system, IAT does not focus on federated access environments and thus does not address requirement R2.

3.4.2 Web-Based Systems

The rich output-modification possibilities make them suited for implementing assistive computing features in the browser [BL07]. However, existing end-user scripting frameworks offer no support for changing the processing of input data. Other assistive computing browser extensions enable support for different input techniques, such as voice [HSL08] or simplified key-based access [SLLH08]. These are special cases of assistive computing approaches, which will be discussed in the next section. The modified processing of input data is statically configured and still confined to the resources provided by the device on which the browser runs.

Summary

Results from assistive computing show how novel types of access environments can be used in combination with existing interactive applications in an open-adaptive way, i.e., without support from the application developer. Requirement R3 is fulfilled, e.g., in the SUPPLE system. The output-adaptation capabilities of assistive computing systems are less powerful than the output-modification APIs provided by the Web interface system, or they make use of the Web access plus end-user scripting frameworks to provide these capabilities.

In assistive computing, changes in the access environment occur less frequently than in post-desktop environments, therefore meta UIs are not considered, and requirements R2 and R6 are not addressed.

3.5 Extensions of the Web Access System

In this section, we discuss two approaches for extending the Web access system with support for post-desktop access environments.

Delivery Context

There are approaches for adding context processing capabilities to the Web access system. The question arises whether these existing approaches that give Web applications access to their context could be used for handling input data from post-desktop access environments.

The W3C has proposed an API allowing a Web application to gather information about its current environment [TJH10]. The general idea of this approach is to expose the delivery

context of a Web page as a DOM model. This DOM model can be used just like the one representing the rendered Web page in the browser; however, it represents the environment. This DOM model should mainly contain information about static properties of the current environment, such as screen size and computing capabilities of the device the browser runs on. This is a rather narrow understanding of what constitutes a Web applications access environment, which does not reflect the diversity of parameters found in post-desktop access environments. By the time the standard was proposed, no implementation was available. It is therefore unclear how web applications could actually use this facility to get access to input data provided by devices in the environment, which would be necessary to implement input-processing interaction strategies.

Ubiquitous Interactor

The Ubiquitous Interactor system [NBW05] interprets a model of the user interface at runtime. The interpretation can be changed by deploying new *interaction forms*. However, developing and deploying such interaction forms is difficult and cannot be done by the end-user for adapting to an operation environment. Also, interaction forms are only applied locally, to interpret a single element of the AUI model, limiting the support for novel forms of input and output which require access to multiple elements of the model at once.

Adaptive Web Browser

The adaptive Web browser and server described in [HI01] adapts Web pages to different contexts inside the browser and on the server. The server can reduce the resolution of images inside Web pages if the network connection is slow. The browser can adapt the appearance of Web pages to arbitrary context. How this adaptation is performed, however, and how the context is gathered, are not described in [HI01].

3.6 Discussion

In this section, we summarize the findings of the previous sections and compare existing approaches to the requirements defined in Chapter 2. We see that none of the existing approaches addresses the requirements to complete satisfaction. However, we can identify an opportunity for Web-based systems to target.

3.6.1 Overview

Table 3.1 presents an overview of how the approaches from the related work support the requirements. The meaning of the symbols in the table is as follows. A \circ means the

requirement is not supported (note that this does not mean it is impossible to fulfill this requirement with the approach, for example by implementing a new Web browser for each access environment arbitrary adaptations can be performed); a ● means the requirement is supported and can be performed by an end-user with some technical knowledge.

	R2	R3	R4	R5	R6
<i>Model-Driven User Interface Development</i>					
ARTStudio [CCT01]	○	○	○	○	(●) ¹
Teresa [LVM ⁺ 05]	○	○	○	○	○
Maria [PSS09]	○	○	○	○	○
Mapache [BPM09]	●	(○) ²	(○) ³	○	(○) ¹
MASP [BLFA08, FBSA08, RBA09]	●	(○) ²	○	○	●
COMET [DCC08]	●	○	○	○	○
I-AM [BDB ⁺ 04]	●	○	○	○	(○) ¹
<i>Device Ensembles</i>					
Federated Devices [BM05a]	●	○	(○) ³	○	○
Pebbles [NMH ⁺ 03]	●	○	(○) ³	○	○
Dygimes [VC04a]	●	○	(○) ³	○	○
DynamoAID [CLC05]	●	○	(○) ³	○	○
Plastic UIs [SCCF08]	●	○	(○) ³	○	○
XWeb [OJN ⁺ 00, ONP01]	●	○	(○) ³	○	●
<i>Hardware Toolkits</i>					
Phidgets [GF01]	○	○	○	○	○
Papier-Mache [KLLL04]	○	○	○	○	○
iStuff [BSF04, JF02, BMRB07]	●	●	○	○	○
ICON [DF04]	○	●	○	○	(○) ¹
OpenInterface [LAAVM09]	○	●	○	○	(○) ¹

¹ no application life-cycle control

² input handled only on a single device

³ only on a per-UIDL-element / subtree basis

	R2	R3	R4	R5	R6
<i>Web Interfaces and Extensions</i>					
Greasemonkey [GRE09]	○	○	●	○	(○) ¹
Chickenfoot [BWR ⁺ 05]	○	○	●	○	(○) ¹
CoScripter [LHML08]	○	○	●	○	(○) ¹
AccessMonkey [BL07]	○	○	●	○	○
Adaptive Web Browser [HI01]	○	○	●	○	○
WebSpeak [HSL08]	○	○	○	○	○
KeySurf [SLLH08]	○	○	○	○	○
UBI [NBW05]	○	○	○	○	○
<i>Assistive Computing</i>					
IAT [CHML06]	○	●	(○) ³	○	○
Supple [GW04, GWW08]	○	●	(○) ⁴	○	○
Yanagida 2009 [YNK09]	○	●	○	○	○

Table 3.1.: Overview of the support for the different modification scenarios in the related work.

The latency experienced by the user is not addressed in any of the state of the art systems. This is surprising, since many of them rely on network communication for propagating changes in the system. This may introduce large delays, which disrupt the flow of interaction.

3.6.2 Addressing the Limitations

Output Modification

Model-driven approaches have been advocated as one solution to UIs in post-desktop computing, yet they have still not caught on [MHP00]. They focus on the needs of the developer. By providing powerful abstractions, the developer no longer has to care about differences between the access environments of an interactive application. Supporting a novel type of access environment requires considerable effort, as it needs to be incorporated into all stages of the transformation machinery.

Regarding the capabilities of adapting the output to an access environment at hand, these systems provide the most extensive capabilities. The complete structure, up to the level of dialog flow between different screens, can be modified at runtime. However, the adaptations

⁴ not with its current UIDL

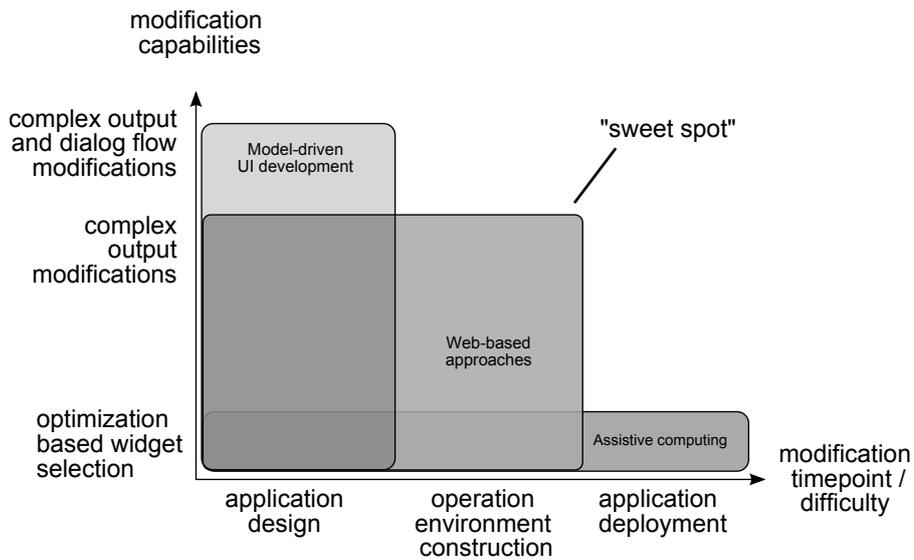


Figure 3.9.: Comparison of state of the art with respect to support for output modifications (Requirement R4). Web-based approaches, in particular approaches using end-user scripting hit the sweet spot in the trade-off between modification capabilities and modification complexity.

have to be specified at application design time. This limitation is overcome by newer approaches, which use the models at runtime. The Web as access system is very similar to these approaches, as the HTML DOM can be used as a model.

Systems developed in the context of assistive computing, such as Supple, Arnauld and IAT, can work completely autonomously at runtime. They use a mathematical model to make optimal choices for different widgets. The downside of these tools is that they do not use a model of the UI at runtime, which prevents them from performing more complex output modifications.

Having such a model of the UI available enables more complex output adaptations. Some assistive computing systems for Web applications use the DOM as a model. These systems support handicapped users with novel interaction techniques and, even more importantly facilitate the development and deployment of new interaction techniques.

Adopting the approach of end-user scripting tools for the Web provides for powerful output adaptation capabilities (e.g., new UIDL elements can be introduced or removed at runtime) at the price of requiring some programming. However, as we will discuss in Section 4.2, this drawback can be mitigated effectively. Therefore, end-user scripting tools demarcate a sweet spot, as illustrated in Figure 3.9.

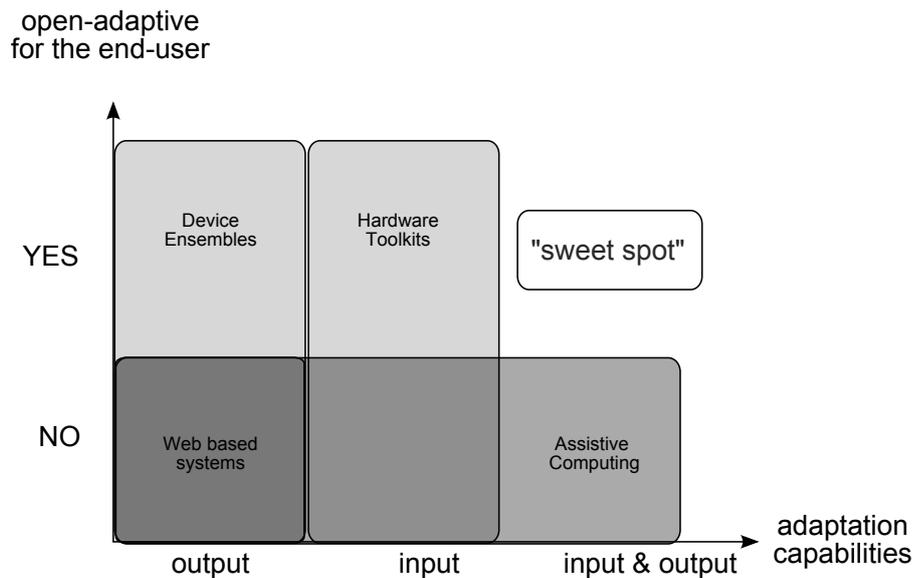


Figure 3.10.: State of the art with respect to support for open adaptiveness and simultaneous modification of output and input processing. No current approach provides the ability to modify the processing of input as well as output combined with the capability to let the end-user introduce such modifications in an open-adaptive way. This sweet spot in the upper right quadrant of the diagram is currently

Simultaneous Modification of Output and Input Processing

Only assistive computing systems are capable of simultaneous adaptation of the output (although they are limited as described above) as well as the processing of input. This is absolutely necessary to support input techniques, such as recognition-based interfaces [MHA00, MHA07]. However, assistive computing approaches require extensive setup and configuration and thus do not allow the end-user to change the access environment easily. Model-driven systems, including those using models at runtime, and systems supporting device ensembles, focus on the adaptation of output and on distributing output to different devices. They do not consider adaptation of the processing of input. Building upon the Web access system provides the required output modification capabilities. However, such systems still lack the ability to adapt the processing of input. Likewise, hardware toolkits also only handle the processing of input data. Approaches for supporting device ensembles and hardware toolkits often allow the user to introduce new types of resources, i.e., they are open-adaptive. Assistive computing approaches are usually not open-adaptive; they require the assistive computing system developer to add support for new types of access environment types. Figure 3.10 shows a lack of support for simultaneous output and input processing modification in an open adaptive way.

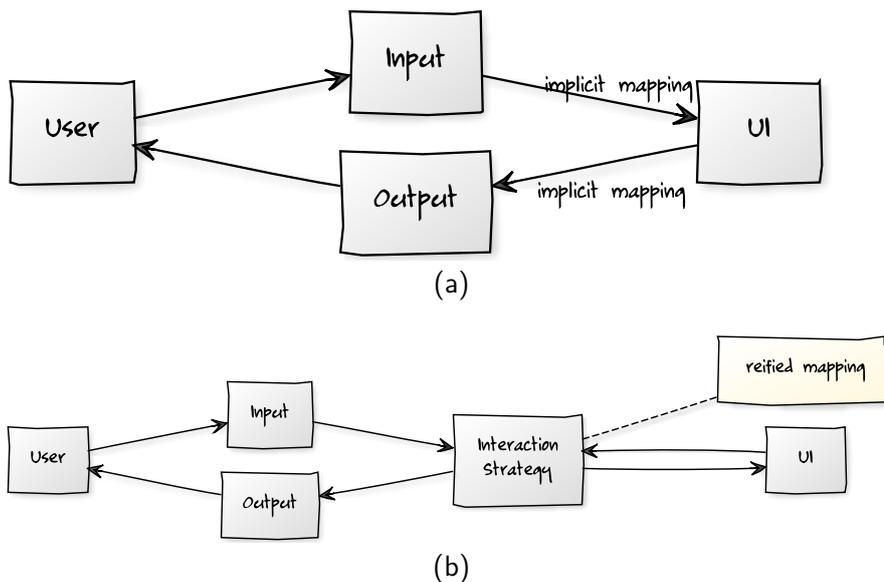


Figure 3.11.: Reification makes the mapping between the input and output data and the UI explicit (3.11b). Without reification, the mapping is implicitly defined (3.11a).

Reification

Flexibility and open-adaptive remodeling can be achieved by reifying the mapping between the input data from the access environment and the application UI, and between the application UI output and the access environment, as illustrated in Figure 3.11.

Representative examples of systems using reifications are *Garnet*, one of the first systems using reification, and *OpenInterface*, the most recent and most advanced system using reified mappings. *Instrumental Interaction* is a theoretical framework making heavy use of reified mappings for describing and analyzing interactive systems.

Garnet Interactors One of the earliest examples of reified mappings were *interactors* in the Garnet system [Mye90]. Interactors encapsulate the mapping from low-level keyboard and mouse input events to behavior of a set of graphical elements. To this end, a protocol for manipulating graphical UI objects is part of the interactors framework. Myers argues in [Mye90] that six types of interactors are sufficient to handle all input from mouse and keyboard for any application. For example, the *new point interactor* is used when the user needs to specify one or more points with the mouse. These points can be used as the corner points for creating a new rectangle in a graphics editor. A major idea of interactors is that one interactor instance can control the mapping between input and many graphical output objects.

OpenInterface Interaction Techniques The most recent and most advanced example of reified mappings are the *interaction technique* components in the OpenInterface frame-

System	Structured Output Modification	Input Not Restricted to Mouse & Keyboard
OpenInterface	X	✓
Interactors	✓	X

Table 3.2.: Comparison of different approaches for reifying the mapping between interaction services and UI. Instrumental Interaction, as a theoretical framework, does not provide any implementation.

work [LAAVM09]. In OpenInterface several components are composed into a pipeline structure, where pipelines end in components representing application UIs. Interaction technique components are a certain type of component that can be employed in the pipeline in between interaction resource components and application components. In contrast to interactors, OpenInterface does not specify a protocol for modifying application and UI output objects. Instead, it must be checked manually whether two components can be connected in a meaningful way. Unlike interactors, OpenInterface components are not restricted to mouse and keyboard input. For example, OpenInterface components for voice recognition are available. In principle, the OpenInterface approach allows the user to customize the processing of input to his or her preferences and habits and the interaction services at hand. However, the lack of a clearly defined protocol for manipulating applications and UI output objects limits the possibilities for modifying the output to the user in interaction technique components.

Instrumental Interaction The theoretic framework of instrumental interaction provides conceptual foundations for the reification approach. In [Bea00] the mappings are called *interaction instruments*. Interaction instruments can cover the whole range from GUI widgets that couple processing of input and output to the user, akin to Garnet interactors, to more generic interaction instruments, such as ‘indirect pointing with a mouse’, akin to OpenInterface components. In [Bea00] interaction instruments are described as being provided with interactive applications, but they could also be provided by the end-user as part of the access environment. In general, the goal of the instrumental interaction framework is to understand UIs, not to implement them.

Table 3.2 shows the capabilities of the existing systems. The systems either support structured output modification or their input handling capabilities are not restricted to mouse and keyboard. No system supports both features.

	Type of meta UI	
	digital	physical
MASP [BLFA08]	✓	✗
ReWiRe [VLC08]	✓	✗
Plastic UIs [SCCF08]	✓	✗
iStuff [BMRB07, JF02, BSF04]	✗	✗
OpenInterface [LAAVM09]	✓	✗
ICON [DF04]	✓	✗
XWeb [OJN ⁺ 00, ONP01]	✗	✓

Table 3.3.: Types of meta UIs used in the related work.

Meta UI

As Table 3.3 shows, a non-digital meta UI has only been considered in XWeb, although the meta UI of XWeb is very minimalistic; for example, it does not allow a user to start a new application. A more sophisticated, digital version is mentioned in [OJN⁺00] but not explained in detail. The relationship between the digital and physical version is not described either.

3.7 Chapter Summary

In this chapter, we have reviewed the state-of-the-art approaches for using applications in post-desktop access environments. None of the existing approaches addresses all of the requirements from the previous Chapter. As suggested by the results from the previous Chapter, the Web access system provides a solid foundation for addressing the limitations with respect to output modification requirements. However, adding capabilities for simultaneous modification of output and input processing to the Web access system is a major challenge. Furthermore, the existing systems provide meta UI capabilities, but these only come as a by-product. The functionality of these does not suffice for addressing requirement R6. The timing behavior and latency of interactive systems has not been addressed by any of the state-of-the-art approaches; thus none addresses requirement R5.



Chapter 4

The Post-desktop Web

This chapter presents a novel approach for using interactive applications in post-desktop access environments, called *MundoWeb*. The approach builds upon the technology of Web applications, which is extended in three aspects. The three extensions also reflect the three main contributions of this thesis:

- a novel approach for adapting interaction with Web applications to post-desktop access environments (realized and implemented in the *MundoMonkey* Firefox extension),
- a model for managing Web applications in post-desktop access environments and post-desktop access environments themselves (realized and implemented in the *Mine-Manager* system), and
- an intelligent caching and prefetching mechanism to reduce the latency perceived by the user when interacting with Web applications (realized and implemented in the *MundoProxy*).

The combination of all three extensions overcomes the limitations identified in the previous chapter and addresses the requirements in Table 2.3.

This chapter explains the overall approach proposed in this thesis using a CRM system as a running example. Section 4.1 describes how the three extensions integrate into the existing access system for Web applications. The following three sections introduce the extensions one by one, focusing on the interplay between the existing Web access system and the proposed extensions. Section 4.2 explains how the handling of input to and output from Web applications is extended in order to support post-desktop access environments, taking into account the specific nature of those access environments as discussed in Section 2.1.2. Section 4.3 presents a model for managing Web applications in post-desktop access environments *and* for managing these access environments themselves. Finally, section 4.4 shows how user-perceived latency is reduced to accommodate post-desktop access environments with a low-bandwidth, high-latency network connection, e.g., mobile environments. Finally, Section 4.5 summarizes the main results of this chapter.

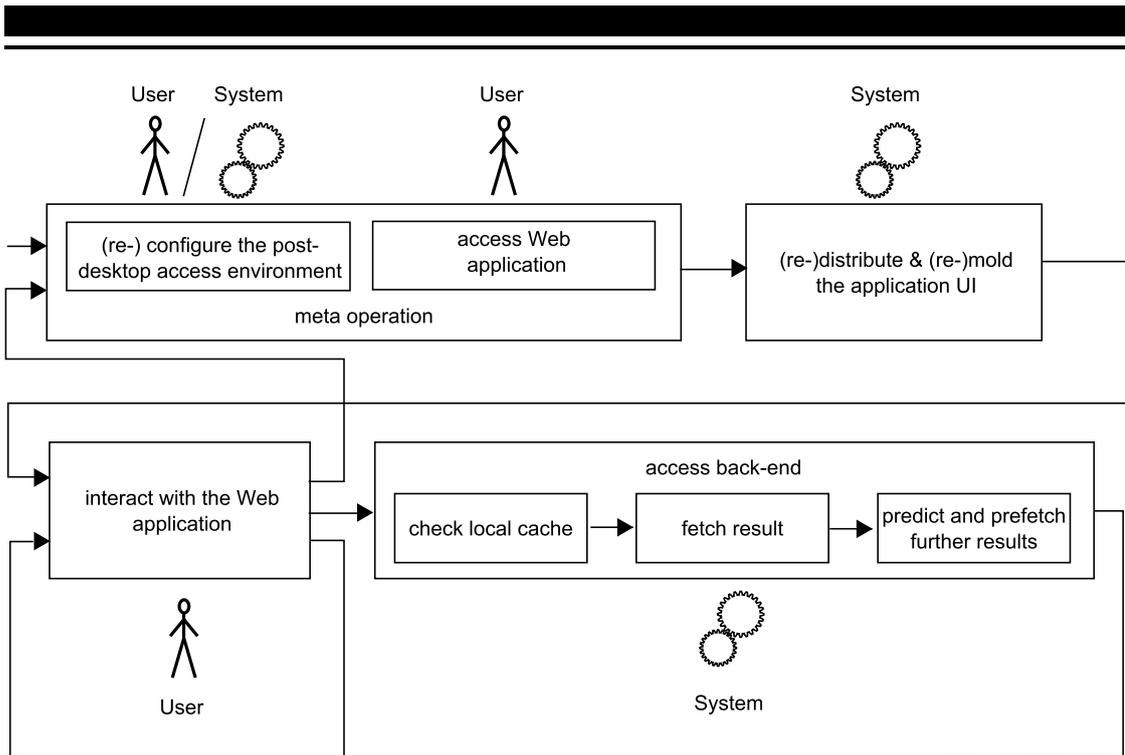


Figure 4.1.: Overview of the steps the user performs to access and interact with a CRM Web application in a post-desktop access environment using the Web access system with the proposed extensions. The meta operations are supported by the new model for managing Web applications in post-desktop access environments. Re-distribution/remolding and interacting with the Web application are supported by a novel approach for adapting interaction with Web applications to post-desktop access environments. Finally, access to the back-end is improved by the intelligent caching and prefetching mechanism.

4.1 Extending the Web Access System

As an example for illustrating the interplay between the components of the proposed approach we consider a CRM application. Such applications need to be accessed by sales representatives in the field on a mobile device, i.e., in a non-desktop environment. Today, the Web application is used via the mobile phone as sole access device. Ideally, the user should be able to utilize other options provided by the access environment to facilitate the interaction, such as a full-sized keyboard, a larger screen or even voice input.

Figure 4.1 shows the temporal order in which the extensions are applied for accessing and interacting with the CRM Web application in a post-desktop access environment.

-
-
- The sales representative configures the access environment. In this step, the interaction resources that will later be used for interacting with Web applications are selected. For example, in the car, voice input from in-car microphones is an option, in the office the sales representative might prefer to use a touchscreen or large keyboard. This step is optional, as sensible choices for the interaction resources used can be used as system defaults.
 - Next, the sales representative accesses the Web application, e.g., by pointing the browser to the URL of the CRM application. A user can access multiple Web applications with one configuration of the access environment. The first two steps are supported by the *MineManager* system.
 - Once a Web page of the CRM application has been downloaded, the system needs to connect the interaction resources to it. This could mean setting up voice control or classical point-and-click control, depending on the chosen access environment configuration.
 - The sales representative then interacts with the CRM application, e.g., looking up customer records, etc. Multiple interactions take place with the same setup. The connection between Web application and the interaction of the user is performed by the *MundoMonkey* system.
 - Most Web applications, and especially business applications like the CRM system, use a multi-tier architecture. The Web application accesses various services in the back-end, e.g., a customer database or a product repository. Round trips from the UI in the access environment to these services causes latency in the interaction. This is a threat to the usability of Web applications. To reduce this latency, *MundoProxy* uses a cache to answer these requests quickly.
 - Only if the cache does not contain the requested data, the services in the back-end are consulted. To keep the cache up to date, *MundoProxy* relies on prefetching in idle times.

Figure 4.2 shows where the three proposed extensions fit into the Web access system architecture. Two concepts, the management of Web applications and access environments and the framework for interacting with Web applications in post-desktop environments, are used within the access environment. They extend and augment the Web browser but do not interfere with the communication between Web browser and Web server, which is still handled by HTTP. Thus, these two concepts can be used to facilitate access to arbitrary Web applications in post-desktop environments. The concepts for reducing the user-perceived latency can be applied for multi-tier Web applications. In the evaluation we concentrate on SOA-backed Web applications. Many business applications fall into this category, and therefore SOA-backed applications are an important sub-class of all Web applications that deserves special attention.

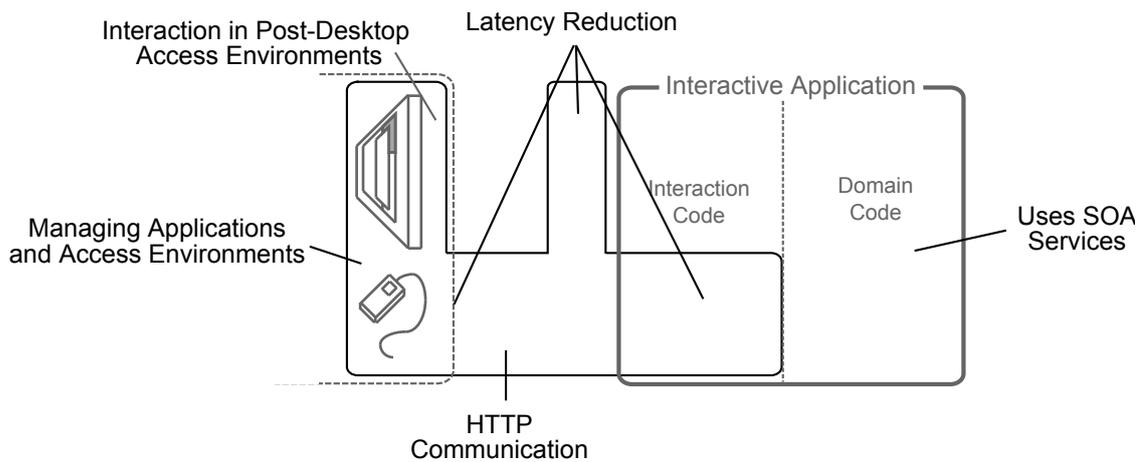


Figure 4.2.: Integration of the proposed concepts in the Web access system (see Figure 2.5)

The above example of the CRM application introduced the three extensions from the viewpoint of the user. We want to complement this view with a more conceptual view, presenting an overview of the three proposed concepts and their relation to the requirements from Table 2.3.

Managing Web Applications in Post-Desktop Environments Managing and configuring the Web application and the access environment comprises any actions the user must perform before she can interact with the target application, or that she must perform to adapt the interaction with the application to changes in the post-desktop access environment. The model proposed in this thesis combines the necessary features to accomplish these operations, thereby addressing Requirement R6 in Table 2.3.

Before the user can use the CRM application in an environment, she must do two things. She must start, and eventually deploy, the application, e.g., by typing the URL of the CRM application into the browser address bar or through a shortcut. Second, the access environment must be configured to user needs, e.g., connecting a voice recognition device to the CRM application. The first operation is already supported by current mobile browsers, although the way this is done needs to be reconsidered taking the characteristics of post-desktop environments into account. The latter operations are not supported by existing systems.

Interacting with Web Applications in Post-Desktop Environments In the Web access system, the browser is the part that is responsible for rendering a Web page and processing the input. Thereby, we abstract away the difference between the actual browser application, i.e., the Internet Explorer or Firefox application, and the operating system it runs on because we treat both as a single entity in the Web access system (see Section 2.2).

Current browsers are designed for desktop access systems. The way the rendering and processing of input is performed is fixed and cannot be adapted or customized at runtime. For example, the way input events from the mouse are translated into cursor movements on the screen is hard coded in the operating system, and the way click events are generated on the Web page is hard coded in the browser's rendering engine. In order to support post-desktop access environments, especially federated post-desktop access environments, a more flexible approach is needed. The output to the user and the processing of input data must be adapted to the resources available in the access environment at hand (Requirements R3, R4 and R2 in Table 2.3).

In contrast, this thesis proposes a flexible approach, relying on reification of the hitherto static mappings, thereby giving the user and the system the option to choose between different mappings. A reification of such a mapping is called an *interaction strategy*. Reifying interaction strategies has been proposed before, e.g., in Garnet [Mye90]. The contribution of this thesis lies in the way this concept is applied. With the proposed approach, it can be used with *existing* Web applications, i.e., Web applications unaware of the reification process, as opposed to supporting reification in the application toolkit.

Reducing User-Perceived Latency The usability of Web applications is not only determined by the quality of the interaction strategies and their fit to the access environment at hand. It is also affected by the responsiveness of the Web application. The more latency the user observes between one of her actions and the response of the application, the less usable the application becomes. In the case of the CRM application there are two sources for this latency: processing time spent in the SOA back-end (enterprise applications are typically implemented on top of a SOA infrastructure) and the latency of the mobile network connection. Both add up to the total latency perceived by the user.

Although the network latency and processing time of the SOA services cannot be reduced, the total latency perceived by the user can be drastically reduced by caching and prefetching requests. However, many prefetching and caching approaches proposed in the literature cannot be used in mobile access environments, because they would quickly drain the battery of the mobile device. The approach proposed in this thesis uses novel techniques and adaptive prefetching algorithms, overcoming these particular problems of mobile access environments and thereby addressing Requirement R5 in Table 2.3.

The evaluation in Section 6.3 concentrates on reducing the user-perceived latency for Web applications with a SOA back-end, as such applications are found in practice.

In the following sections, we deviate from the more chronological order in which the extensions are applied above, as it is necessary to present the concepts used for interacting with applications (Section 4.2) before we can describe how they are managed and controlled in

Section 4.3. We conclude with a presentation of the mechanisms for latency reduction in Section 4.4.

4.2 Interacting with Web Applications in Post-Desktop Access Environments

During runtime, the role of the access system is to mediate between actions and perceptions of the user in the access environment on the one side, and the UI of the interactive application on the other (see Definition 6). To do so, the access system translates actions of the user, digitized by input devices, into digital events in the UI, and vice versa, translates events happening in the UI into updates of the output to the user. Thereby, the way this translation is performed heavily depends on the access environment at hand, e.g., handling events from a pointing input device requires a completely different mapping compared to a voice input device. In desktop computing it was possible to optimize and standardize for a single type of access environment.

The existing Web access system, especially the Web browser component thereof, is designed for desktop and mobile access environments. The mappings used to translate between interaction devices and application UIs are defined once and for all in the operating system and the Web browser application. The browser cannot be reconfigured to use another *type* of interaction resource easily. For some systems, even the employed interaction resource *instances* cannot be changed. For example, the touchscreen-based direct-pointing interaction on the iPhone is always performed with the built-in touchscreen. The limitations of this approach for post-desktop environments are obvious. For example,

- the best choice of employed mappings depends on the available interaction resources, the application at hand, and the user's preferences and habits. At least the first parameter varies considerably in post-desktop environments.
- relying on statically attached interaction resources obviates using federated access environments (see Definition 5).

In order to support arbitrary post-desktop access environments, a more flexible setup is needed (see Requirements R3, R4 and R2 in Table 2.3). Reification of the applied mappings, as described in Section 3.6.2 has been proposed in the literature to achieve the necessary runtime flexibility. However, the existing approaches are limited. They either do not support structured output modification, or they are only able to handle mouse and keyboard input. Utilizing the specific nature of Web applications, which provide a very rich runtime-model of their UI, we can achieve reified mappings that combine both advantages. This novel approach relies on the notions of *interaction resources* and *interaction strategies*, which we explain in the following.

4.2.1 Interaction Resource

The notion of an interaction device as proposed by Braun ([BM05a]) is not adequate to analyze post-desktop access environments. For example, voice recognition systems are based on microphones and are not perceived as a device by the user. Nevertheless, they provide input to an application. In our approach, the access system decomposes into *interaction resources*. Similar to [VC04b] we define:

Definition 9 (Interaction Resource)

An interaction resource is a component of an access environment that can be individually employed for user interaction by interactive applications.

An interaction resource is made available to the interactive application through the access system. A mouse driver together with the mouse device thus constitutes an interaction resource for the desktop computing access system. Interaction resources are not the same as interaction devices, as often the same hardware in an access environment can be used for multiple interaction resources, potentially even simultaneously. For example, a speech recognition resource is realized by a combination of microphones (hardware) and a speech recognition engine (software). The same microphone hardware can be used to generate non-speech input in a different interaction resource.

Atomic interaction resources, like a single push button, can be aggregated to more complex interaction resources. For example, a mouse is composed of at least three atomic interaction resources, the two buttons and the movement sensor. It is up to the developer of the mouse interaction resource to decide whether she wants to expose the buttons and the movement sensor as individual interaction resources to applications or to define the whole mouse as one integrated interaction resource.

To describe the direction of information exchange we take the point of view of the application, i.e., the application processes input generated by the user interacting with *input resources*; output to the user is generated by using *output resources*.

Interaction Resources in Federated Post-Desktop Access Environments

To enable interaction with Web applications in *federated* post-desktop environments, we require that all interaction resources are connected via a communication middleware. This means that input events from input resources are sent as messages in a middleware. Likewise, output events from the UI are sent to output resources via the middleware. Using a

middleware is a necessary step to support federated access environments (see Definition 5) in the access system. However, the construction of such environments requires additional support, which we will discuss in Section 4.3.

It is unreasonable to assume that an extensive set of interaction resources will become available for any particular middleware chosen for a research project. Therefore we use proxy implementations for existing interaction resources that internally use, e.g., Human Interface Device (HID) or any other standard protocol. This approach towards implementing interaction resources for a system has is commonly found in the literature, e.g., in [BMRB07,VLC08,RHC⁺02]. Wrappers for connecting several input resources with very different characteristics (pointing, voice recognition, pen input) to such a middleware have been implemented [Hra08,Zin08,Ste09b] for experimentation as part of this thesis.

4.2.2 Interaction Strategies

We refer to a reification of a mapping from interaction resources to the Web UI and back as an *interaction strategy*.

Definition 10 (Interaction Strategy)

An interaction strategy is an implemented mapping from UIs to interaction resources in an access environment.

An example for a very well-known interaction strategy from desktop browsers is "text entry with the keyboard". In this interaction strategy pressing buttons on the keyboard is mapped to changing the value attribute of form DOM nodes. In post-desktop computing an alternative interaction strategy could be "text entry by voice", where speech recognition is used to enter text.

As identified in Section 3.6.2, existing approaches are not able to combine output modification using a rich model with input processing from arbitrary sources. Executing interaction strategies within the browser component can overcome this limitation. The interaction strategy has complete access to the application UI DOM model. The DOM can be augmented so that interaction strategies are able to receive events from arbitrary interaction resources via the middleware.

Output Modification

Interaction strategies need to have a rich API for modifying the output to the user and the structure of the UI to fulfill requirement OUT in Table 2.3. This in turn requires a

rich representation of the UI at runtime. In this aspect, interaction strategies as proposed in this thesis differ from OpenInterface and Assistive Computing systems (see 3.4), where no such API is provided.

Interaction strategies use the HTML DOM as runtime representation. By manipulating the HTML DOM tree, arbitrary modifications of the output and the UI structure can be performed. For example, the developer can add new HTML elements, remove existing elements, change location of elements in the DOM tree and change property values of elements, such as their color. Still, simple UIs can be generated using HTML markup. Furthermore, by using the well known DOM API, business users can leverage on their existing knowledge of Web scripting and learn from examples in any Web page on the Internet.

Input Processing

Additionally, interaction strategies need to be able to process input events from interaction resources in the access environment to fulfill requirement IN (see Table 2.3). Asynchronous event handlers, such as for the DOM `onClick` event, are used pervasively in Web programming. Therefore, to remain compatible with the event based programming model widely employed in Web UIs, we introduce a novel API that enables user scripts to receive events from interaction resources, and send events to interaction resources. Business users can then use their knowledge of Web scripting and can use the full JavaScript programming language for implementing the processing of input data.

To summarize, the proposed interaction strategies progress beyond the state of the art in the following aspects

- we provide a rich output modification model while at the same time
- we provide simple APIs for the processing of input events following the event-based pattern known to many end-user developers for the Web.

4.2.3 Composition of Interaction Strategies

As described above, the concept for interaction strategies facilitates implementation of individual interaction strategies, because existing knowledge of Web application development can be reused. Thus, a large repository of interaction strategies is likely to be available for adapting interaction with a Web application to a post-desktop access environment. However, it is unreasonable to assume that a single integrated interaction strategy, which comprehensively addresses all interaction resources in the environment, is available for each post-desktop access environment. A far wider range of post-desktop access environments can be covered, if single interaction strategies can be composed to more complex strategies.

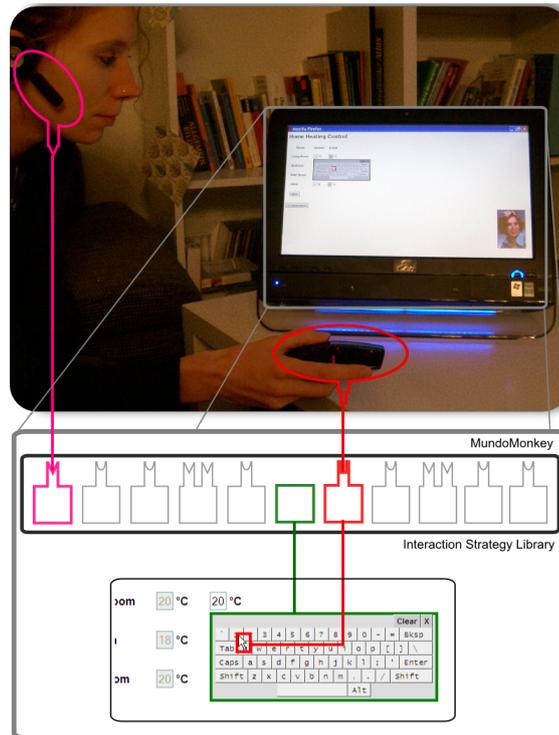


Figure 4.3.: The upper half of the picture shows the physical access environment in which a user interacts with a room-control Web application using a pointing resource and a speech input resource. Output is presented to the user on the TV screen. The lower half schematically shows how the interaction resources are connected to the UI with interaction strategies by MundoMonkey.

Strategy Library

A completely automatic approach wherein the system chooses interaction strategies directly from a global repository, is problematic. It would require a globally standardized description of how interaction resources can be handled by interaction strategies. Furthermore, user's preferences need to be taken into account when choosing strategies from a global repository. For example, one user prefers text entry with a pointing device using a strategy like Dasher [WBM00] while another user prefers an on-screen keyboard.

We propose to store a library of interaction strategies locally in the browser. The user of the browser instance decides which interaction strategies are put in the library. This turns each browser instance into a personalized configuration of the access system. The user can either be a business user that pre-configures a browser being used in a non-standard access environment for other employees or customers in the company, or an end-

user configuring her own browser. Figure 4.3 shows a sample strategy library containing 11 different strategies, represented by the different symbols in the lower half of the figure.

As an example scenario, we consider a sales agent who wants to use the voice input interaction strategy (described in Section 6.1.1) to interact with a CRM Web application using the voice-recognition capabilities of her car (which is connected to the middleware). If the interaction strategy is used for the first time, it must be deployed into the local strategy library. The interaction strategy is a piece of code which must be deployed into the strategy library of the sales agent's browser. It can be loaded either from a global repository or from another local source.

In our realization of the concept, deploying interaction strategies is done either by pointing the browser to the URL from which the strategy can be loaded or by directly storing the source code file of the interaction strategy in the strategy folder. After this has been done, the new interaction strategy is available and ready to use. Runtime management of the interaction strategy library is part of the *meta operations*, which we will discuss in Section 4.3.

Composition

As it would be cumbersome for the user to select desired interaction strategies manually, automatic composition of interaction strategies can be applied. In this way, the interaction strategies available in the local library are matched the interaction resources in the access environment. A composition is calculated that best fits to the interaction resources at hand. Figure 4.3 shows an example of such a configuration, where 3 (out of 11 available) interaction strategies have been selected and connected to interaction resources in the environment. We present an algorithm for calculating such compositions in Section 5.1

4.2.4 Resulting Extensions to the Web Access System

To realize these concepts, the Web browser component of the Web access systems needs to be extended. It is the only component of the Web access system that has access to the complete application UI as well as to all interaction resources in the access environment. Different types of extensions are possible. In the most extreme case, one can reduce the Web browser to just a host of the DOM and handle in- and output completely via interaction strategies. Another solution would be to utilize the built-in rendering capabilities of the browser, i.e., let the browser application render the DOM on a directly connected screen, as in desktop computing. In this case only additional output resources are connected using interaction strategies. The latter approach has been implemented and evaluated in the MundoMonkey system (see Sections 5.1 and 6.1) , as it is more efficient for the local screen and still provides the full flexibility of interaction strategies.

4.3 Meta Operations for Managing Web Applications and Post-Desktop Access Environments

Using the Web access system, deploying applications into the access environment is very easy. For example, Web applications can be deployed by typing the URL into the address bar of the browser. The process is so easy that Web applications are said to require *zero-installation*. This characteristic should be retained in an access system for post-desktop access environments (see Section 2.2). In post-desktop access environments, it is not enough to support easy deployment of applications into pre-fashioned access environments. Instead, end-users and business users should be enabled to construct access environments on their own and manage how applications are accessed in them (Requirement C&M Table 2.3).

In this section we address the question of which operations need to be provided by the access system so that users can manage and configure Web applications *and* the post-desktop environments in which they are accessed in. We call these operations *meta operations*, to distinguish them from base-level operations performed within applications by interacting with the application UI.

Definition 11 (Meta Operation)

Meta operations are

- *operations for managing applications, such as starting and stopping applications, as well as*
- *operations to configure and manage the interaction resources in the access environment and their connection to applications.*

This definition covers both operations found in existing desktop and mobile access environments as well as operations necessary for supporting post-desktop access environments.

4.3.1 Meta Operations for Managing Applications in Desktop and Mobile Access Environments

Meta operations are readily available in existing systems for desktop and mobile Web access, see Figure 4.4. Examples of meta operations provided by existing systems are the following.



Figure 4.4.: Meta operations in existing desktop browsers (a) and mobile browsers (b).

Desktop browser The desktop browser has meta operations allowing the user to

- access Web applications by typing the URL into the address bar
- store and retrieve bookmarks for frequently accessed URLs
- manage a history list of visited pages

Mobile browser The meta operations provided by mobile browsers closely resembles the meta operations provided by desktop browsers, e.g., they feature URL address bar and bookmarks.

4.3.2 Meta Operations for Managing Post-Desktop Access Environments

For using any interactive application in federated post-desktop access environments another class of meta operations is needed. As first observed by Coutaz, the user needs to control and configure the federated access environment itself [Cou07]. Coutaz coined the term *meta UI* for referring to a component providing such operations (cf. [Cou07]). In its narrow meaning, the term meta UI stands for ‘a point of control used for inspecting and manipulating the configuration of a post-desktop environment’ (cf. [VSL⁺09]). As Web access is one use case for federated post-desktop environments, we conclude that such operations need to be provided as well. The minimal set of operations for controlling the post-desktop environment is adding and removing interaction resources from the federated access environment. Also, the inspection of the environment through the user should be supported.

4.3.3 Meta Operations for Multitasking in Post-Desktop Environments

Whenever more than one application is used in parallel in an access environment, the user needs to manage the connection between the interaction resources in the access environment and the applications. In desktop environments, the window manager of the operating system provides operations for this purpose

- distributing the available screen space among applications
- assigning all input and output resources to an application by maximizing it.

The same operations are implemented a second time within desktop browsers, as browser tabs. It is interesting to note that a tighter integration of these meta operations with the Web browser has now been adopted by all browser implementations. This suggests that unifying the meta operations of the window manager and the browser makes the Web access system more usable.

On mobile devices, we also find this type of functionality. However, it is tuned towards the situation on mobile devices. For example, sharing screen space between different applications is not supported, as the screen is too small.

The realization of window manager meta operations for federated post-desktop environments must differ significantly from existing systems due to the nature these environments. Window managers of existing desktop systems are based on the assumption that the interaction with Web applications only happens by means of the interaction resources directly provided by the device running the browser. Thus, existing window managers do not foresee any meta operations for using interaction resources in the access environment, i.e., interaction resources provided by devices other than the one running the browser. Approaches for extending the window manager to multiple screens and computers, such as presented in [BDB⁺04] cannot be applied to types of interaction resources different from screens.

4.3.4 Meta Operations for Access System Customization

As discussed above, the user manages a library of interaction strategies in the browser component of the Web access system (see 4.2.3.1). Thus, corresponding management operations need to be part of an access system for post-desktop access environments. Again, similar functionality is already included in desktop browsers. At least the Firefox browser provides an easy-to-use extension mechanism, where users can install and deinstall extensions. With Greasemonkey [GRE09], the user can install and de-install user scripts. In our prototypical implementation, we rely on the functionality provided by the Firefox browser.



Meta Operation	Examples
Browser	
application access	address bar, bookmarks (desktop)
connecting output resources	browser tabs
Access System Customization	
add interaction strategy	Firefox extension mechanism in desktop environments
remove interaction strategy	Firefox extension mechanism in desktop environments
Meta UI	
adding interaction resources	n/a in desktop environments
removing interaction resources	n/a in desktop environments
Window Manager	
connecting output resources	maximize / minimize buttons, window resizing
connecting input resources	window focus & spatial mapping (desktop)

Table 4.1.: Complete meta functionality needed for accessing Web applications in post-desktop environments.

4.3.5 Resulting Extensions to the Web Access System

To the author’s knowledge, this is the first comprehensive analysis of meta operations for Web applications in post-desktop access environments. Its operations are a superset of the existing meta operations already included in mobile and desktop Web browsers. Table 4.1 contains a list of operations that are considered in the model.

In order to support these meta operations, a new system model is needed. It must comprise the notion of interaction resource and interaction strategy as introduced in Section 4.2. The operations supported by this model then need to be made available to the user in a meta UI, to fulfill requirement R6. Reconciling all operations of the model into a single meta UI avoids duplications, like the ones found between window managers and browser tabs on desktop systems. The meta UI and the underlying system model constitute necessary extensions to the existing Web access system to support post-desktop access environments.

4.4 Reducing User-Perceived Latency for Web Applications

According to [PdSB00], individual Web pages in a Web application can be seen as ‘one shot, higher-order messages sent from designers to users.’ Applying suitable interaction strategies reduces the effort for interacting with individual pages in a Web application after the Web page has been downloaded into the access environment. However, as most Web applications comprise a number of individual Web pages, a large degree of the perceived effort for interacting with the application is caused by the delay the user experiences while waiting for roundtrips from the browser to the server, and further to the application back-end [PK08]. If the application is designed and implemented in a naive way, the user perceives the invocation time of the back-end *and* the network time as latency every time a service in the back-end is requested. To ensure a good usability of Web applications in post-desktop environments, dealing with this latency is crucial.

Taking latency into account during the application design is not an option for interactive business applications. These applications are developed to support business processes, and these processes need to be adapted and tailored to business needs frequently. In the realtime enterprise these changes need to be performed by business users with minimal technical knowledge, instead of technical experts.

By adopting Web applications as technology, these business users are freed from having to tailor the interaction to the interaction resources in the access environment at hand, because this is automatically done by the interaction strategies in the user’s browser. If the business users needed to consider latency in the design of the application, this gain would be nullified.

Even if one could address latency issues during application design, the resulting application would still not be optimal for every user. Instead it would be only optimal for a hypothetical *average* user. In reality different users use applications in different ways. Ideally, the adaptation should thus adapt to the individual user automatically [HS08].

Therefore, to address these problems and fulfill requirement LATENCY 2.3, we propose to use caching and prefetching mechanisms. Using a cache is a common approach to reduce latency. With a cache, the latency can be reduced without additional development complexity. The cache stores all requests to the back-end so that they are available much faster should the request be made again. The effect of the cache can be further increased by relying on *prediction and prefetching*, i.e., adding entries to the cache without an explicit request.

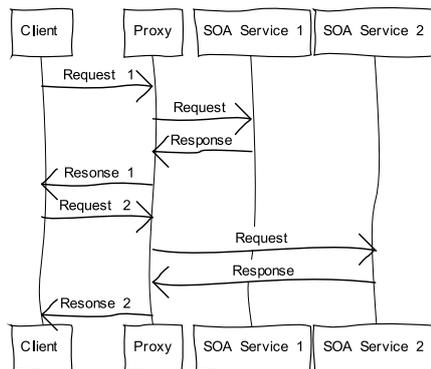


Figure 4.5.: Without piggybacking, the proxy returns only one response to the client for each request.

4.4.1 Probabilistic Prefetching

Simple caching alone reduces the user-perceived latency in case the user performs the same request more than once during a session. The rate of cache hits can be increased by proactively predicting future requests of the user and prefetching those into the cache *before* the user actually invokes them. In our evaluation, we use the FxL [HS07] algorithm for predicting the future requests of the user. This algorithm matches the recent history of service calls to the history of all observed service calls. The services that most often succeeded the current history in the past are chosen as predictions for the future request. For example, if the system observes the Web application performed the service requests A and B, and in the history requests to A and B were often succeeded by a request C, the request C would be predicted. Figure 4.6 shows how the prediction step leads to an additional cache hit compared to the non-prefetching baseline shown in 4.5.

Prediction and prefetching of results has a much larger effect if the effort for performing an operation is high, as in mobile post-desktop environments. The effect is much less noticeable in desktop access environments, where the effort for performing an operation is low. Even if the prediction fails the fraction p of all attempts, there is a net benefit of performing predictions, if $p(W + C) < C$. Here, W denotes the average overhead for a wrong prediction (which may, e.g., come in the form of additional latency while waiting for prefetching or from correcting actions that have to be taken to cancel the effects of a prefetched request) and C denotes the average effort for performing an operation in the application. We assume that the effort for performing an operation is 0, if a correct prediction has been made, i.e., it is returned instantly from the cache. This means the larger the value of C is, the larger can be the fraction of wrong predictions, before they become counterproductive. We leverage this fact and probabilistically predicted requests to the back-end, which then prefetches and feeds to the cache.

4.4.2 Resulting Extensions to the Existing Web Access System

Caching and prefetching can be introduced at various points in the access system. The cache is most effective if it is implemented *inside* the access environment, i.e., on the client browser. However, in the case of SOA-backed Web applications, the client does not directly access the SOA services in the back-end. Instead a proxy is used, as described in Chapter 3.

A proxy service is used for two reasons. The first reason is a technical limitation of current browsers. Existing browsers only permit requests to the server the Web application was initially loaded from because of the *one origin policy*. Thus, the proxy server has to act as an intermediate. Abiding by this restriction is arguably optional, as it is unclear whether this restriction will apply for future browser implementations. However, to evaluate the performance with realistic workloads and real backend services, it is necessary to conform to this restriction. A second reason for introducing the Web server as proxy between the access environment and the SOA back-end is not of technical nature but due to conceptual concerns. The setup with a proxy permits compression of complex SOA messages returned by the back-end services into a format more suitable to the client and the mobile network connection.

This proxy can also be augmented to predict and cache requests to the back-end. As an improvement to this basic setup, we propose to use a distributed cache, i.e., a local cache in the client browser as well as a cache at the proxy. This combines the advantages of the local cache in the browser with the ability to prefetch a larger amount of requests at the proxy. We propose a novel efficient mechanism geared towards the characteristics of mobile and other post-desktop access environments for synchronizing the two, which we explain in the next section.

4.4.3 Piggybacking Cache Synchronization

The cache on the proxy can easily be filled in idle times, e.g., as suggested in [EJM00]. There is no restriction in terms of bandwidth or battery runtime. The local cache at the client in the access environment, however, should not be filled in idle periods, because for mobile post-desktop access environments this would quickly drain the battery [Cao02], and we assume the storage space that can be allotted to the cache is restricted. The piggybacking approach solves these problems.

The normal protocol between the Web browser and the proxy server is that the server returns one response for each client request shown in Figure 4.5. Using piggybacking, we allow the server to reply with multiple responses to the original request of the client, i.e., the additional responses travel to the client piggybacked to the original responses, as shown

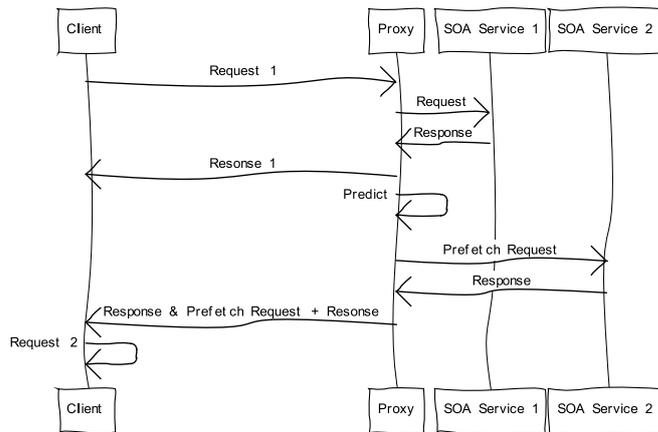


Figure 4.6.: With piggybacking, the proxy can send multiple responses to the client, potentially generating additional cache hits.

in Figure 4.6. At the client side, the additional responses are added to the local cache in the access environment. Thus, no roundtrip to the proxy and the backend is necessary, if the user invokes a request later on which can be answered from the cache. This reduces the user-perceived latency to nearly zero.

We also implemented some other trivial improvements over the naive setup.

- a service can be marked as not eligible for prefetching,
- the client informs the proxy of any requests that were retrieved from the cache, to keep the history on the proxy synchronized to the history of the client, and
- entries already present in the client cache are not sent to the client again.

4.5 Chapter Summary

We have described essential concepts for an improved access system for Web applications in post-desktop access environments. The proposed concepts extend the existing access system for Web applications in three ways:

- A method for flexibly customizing and adapting the Web browser to handle input from arbitrary interaction resources in the post-desktop environment is proposed.
- A set of meta operations, which is general enough to handle Web access in arbitrary post-desktop access environments, including federated access environments, is introduced. It is meant to replace and augment existing meta UI functionality provided by mobile and desktop browsers.

-
-
- Finally, a solution to the problem of high user-perceived latency in mobile access environments was presented.

The concepts in their entirety are the first approach to tackle all aspects of access to interactive Web applications in post-desktop environments, addressing the requirements in Table 2.3 in a coherent way. Existing approaches only focus on individual requirements, e.g.,

- supporting construction of the access environment in an automatic [VLC08, CLV⁺03] or manual way [NRCM06];
- improving usability by providing a UI automatically adapted to the access environment [GW04, SZG⁺03] or by allowing the user to customize the UI [DF04];
- or improving the usability by reducing latency through caching [ED05].

Also, in contrast to related work, where clean slate approaches are proposed, all three components making up the approach of this thesis have been carefully designed in a way permitting their implementation as conservative extensions to existing, widely used Web technologies. This way existing SOA-backed Web applications can be reused as much as possible. Therefore, it is possible to evaluate the resulting access system with real applications.

Chapter 5

Models, Designs, Algorithms

The previous chapter presented the overall *MundoWeb* approach and introduced its three components and their integration into the existing Web access system. This chapter now takes an inside look into the three components of *MundoWeb*, detailing the applied algorithms, underlying system models and UI designs. It presents the rationale for design decisions and discusses alternatives where applicable. The contents of this chapter form the basis for the implementation and evaluation of the concepts presented in the next chapter. The concepts for interacting with Web applications using the interaction strategies have been implemented in the *MundoMonkey* Firefox extension. The concepts for managing post-desktop access environments and Web application lead to the implementation of the *MineManager* meta UI. And, finally caching and probabilistic prefetching have been evaluated with *MundoProxy*. Figure 5.1 situates these components in the context of the existing Web access system.

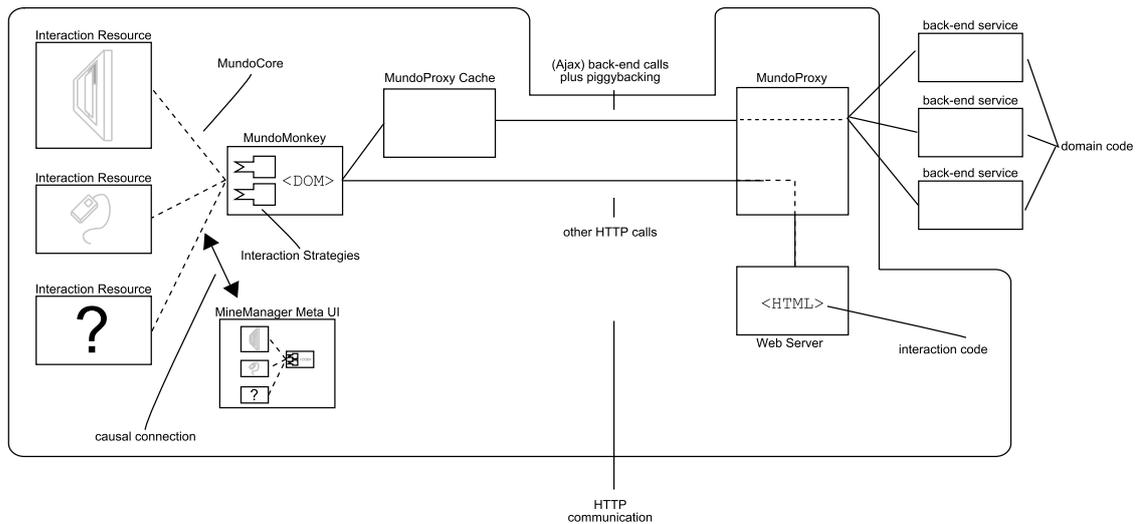


Figure 5.1.: MundoWeb overall architecture in the context of the Web access system (see Figure 2.5).

5.1 Interaction Strategies for Web Applications in Post-Desktop Access Environments

In this section, we describe the design of an extension to the Web browser, called MundoMonkey, supporting the use of interaction strategies as presented in Chapter 4. The extension enables the browser to use arbitrary interaction resources in (federated) post-desktop access environments.

First, we describe the design of the API for developing interaction strategies for MundoMonkey. It has been designed to reuse the well-known event-based programming paradigm pervasively found in Web applications. We illustrate the use of the API with a sample interaction strategy for connecting a Web page in the browser to two input resources in the access environment. We also discuss how interaction strategies can be used to realize other use cases, e.g., context-aware interaction. We then turn to the case where multiple interaction strategies are available at runtime and describe an algorithm for calculating the best combination of interaction strategies. Finally, we discuss options for realizing MundoMonkey on top of existing technologies for the Web access system. This leads to the conclusion that MundoMonkey needs to be implemented as a native extension to a Web browser instead of relying on an approach using only JavaScript and proxies.

5.1.1 Developing Interaction Strategies

The availability of a large number of interaction strategies for a wide range of interaction resources is crucial for the success of any system relying on reification. Therefore, interaction strategies should be easy to develop, and their programming model should leverage existing knowledge of Web development as much as possible.

Interaction strategies prescribe how changes in the UI of an interactive application are rendered to the output resources in the access environment and how input data from the access environment is processed and turned into events in the UI. Thus, suitable APIs for output and input need to be designed.

Output Modification API

Interaction strategies should be able to use the full DOM API for modifying the output to the user. This satisfies the requirement for structured output modification. User scripts [GRE09], i.e., JavaScript code that is injected in the browser and executed in the context of a Website, can leverage the rich output modification capabilities of the HTML DOM to manipulate the output to the user. The large number of available Greasemonkey scripts (more than 30,000) suggests that a business user should be capable of developing interaction strategies. Thus, the API is sufficiently easy to use.

Input Processing API

User scripts lack support for reacting to input events from interaction resources in the access environment. Therefore, we extend the DOM API as follows. Interaction strategies can add event listeners not only to DOM elements in the Web page but also to interaction resources in the access environment. To do this we provide access to the middleware connecting the federated access environment through the global variable `mc`. The name of the variable refers to the communication middleware MundoCore [AKM07], which is used in the implementation.

MundoCore is a channel-based publish/subscribe middleware. Interaction strategies can register an event listener to a channel. The event listener is subsequently called whenever the specified event in the interactive space occurs. Likewise, strategies can raise events in the access environment, e.g., to send a new recognition grammar to a speech recognizer in response to a change in the Web page. All published and received event data is encoded using the JavaScript Object Notation (JSON) [JSO] format. Serialization of JavaScript objects and de-serialization into JavaScript objects can be done very efficiently with this format. This API is very convenient for Web programmers, as it resembles the event-based programming style applied in most JavaScript programs on the Web. The following sample code shows an event handler which moves a cursor on a Web page.

```

mc.addEventListener(
    "pointer_move",
    function (evt) {
        //parsing the JSON coded event data
        var data = eval("(" + evt + ")");
        var curX = computeX(cursor);
        var curY = computeY(cursor);
        cursor.style.top = curY + data.dy;
        cursor.style.left = curX + data.dx;
    });
//Upon receiving a pointer_move event from the environment,
//the element representing the cursor is moved to another
//location using methods from the Web page \ac{DOM}.

```

Listing 5.1: Input processing

Sample Strategy

To turn the simple event handler above into a useful pointing interaction strategy, code for performing click events must be added. Simply performing the `click` event on the element at the coordinates of the cursor will not suffice, as the cursor is part of the DOM and thus would receive the `click` event at these coordinates. A solution is shown in the following listing.

```

mc.addEventListener(
    "pointer_click",
    function (evt) {
        var curX = computeX(cursor);
        var curY = computeY(cursor);
        cursor.style.display = "none";
        el = document.getElementById(
            curX, curY);
        cursor.style.display = "";
        performClickEvent(el);
    });
//The cursor is temporarily hidden before the click. This ensures
//that the original \ac{DOM} elements positioned underneath the
//cursor receive the click event.

```

Listing 5.2: Point & Click Strategy

In this example, fixed channels are subscribed (`pointer_move` and `pointer_click`). However, the channel names are normally provided by the MundoMonkey framework at run-

time through a management API that interaction strategies must implement. The channel names are taken from the description of the interaction resources obtained from Mine-Manager. This API is also used during the computation of a juxtaposition of interaction strategies, which we describe in Section 1.

Combining Input Processing and Output Modification

One of the advantages of interaction strategies compared to other approaches for reification is that interaction strategies can manipulate the output to the user and process input data from the environment at the same time. A very simple example strategy making use of this feature is a strategy for multitouch like interaction (MLI). The MLI strategy allows the user to control two independent cursors on the screen. MLI does not require that displaying two cursors is supported by the operating system. Instead, MLI adds two elements to the Web application DOM, playing the role of mouse cursors as shown in Figure 5.2. Therefore, this strategy is an example for testing whether it is possible to support novel forms of interaction as MundoMonkey strategies, although this has not been supported by the platform on which MundoMonkey runs. The position of the two cursor elements is continuously updated by attaching event listeners to the events from the pointing devices, using event listeners as in the code above.

In addition to the simple ‘click’ behavior implemented as shown in the code listings above, MLI implements a ‘pinch’ gesture: If the buttons on both pointing devices are pressed while their cursors are over the same UI element, this element can be enlarged or shrunk using a pinch gesture. Figure 5.2 shows how the strategy allows the user to zoom interface elements with this technique. MLI runs with any Web application. Displaying two cursors would be extremely difficult without having access to a high-level output representation, i.e., the DOM.

Context-Awareness and Interaction Strategies

One feature deemed essential for accessing and using applications in post-desktop environments is *context awareness* [CCDG05]. We argue that context awareness (at least for system use [MH09]) can be achieved by interaction strategies as well. A different concept, e.g., context strategies, is not necessary.

From a *technical point of view*, there is no difference between a source of contextual information and an input resource as defined above, and therefore contextual data can be fed into applications by means of input resources. There can be no crisp distinction whether data from an interaction resource represents input or context information. Context and input can be seen as two extremes of a continuous spectrum of user input from the system’s point of view.

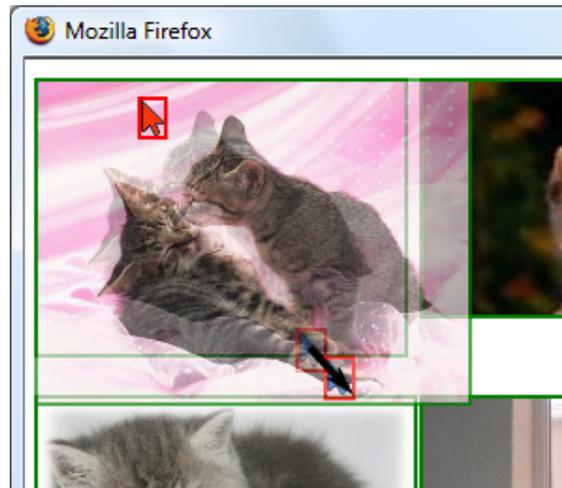


Figure 5.2.: Zooming a picture with the MLI strategy.

Usually, only *intentional* signals of human users, i.e., signals that are generated with the explicit goal of generating input to an application are considered as *input* from the user. However, *non-intentional* signals can be processed with the same machinery (sensor hardware, etc.) that is used for intentional signals. For example, the data from the mouse may be used as *input* for controlling a pointer. At the same time, the same data may be used to detect if the user is nervous by measuring the tremor in the mouse movements. The latter would then be considered *contextual* data.

Following the well-known definition for context from [Dey01], *all* user generated input (including, e.g., mouse movements) can be considered context as it is definitely relevant for the interaction between user and application. The definition of Mühlhäuser et al. [MG08] overcomes this problem by restricting context to *auxiliary information*. However, the decision of what is *auxiliary* can only be made at runtime. In one case, processing voice input may not be necessary for an application, as a pointing device and a keyboard are available. Accordingly, if the application uses voice input in such an access environment, this would be contextual data. If, however, the mouse and the keyboard device are removed, voice input becomes essential for using the application, and thus becomes normal input.

We conclude that *interaction resource* may also refer to input data sources which provide data that would be considered contextual data. We will use the terms *input resource*, *output resource* and *context resource* to further specify the nature of an *interaction resource* where this is necessary.

Management API

Interaction strategies not only need to interact with the resources in the access environment and the UI. They also need to interact with the MundoMonkey system itself. For

example, they must be started and stopped if interaction resources become available or unavailable. Another example is the setup of a wiring, i.e., coordinating interaction resources and interaction strategies to use the correct channel names from the middleware. When a strategy is activated for an interaction resource or a set of interaction resources, MundoMonkey sets the correct channel names.

MundoMonkey communicates with the *MineManager* (described in Section 5.2) to get a list with the descriptions of the currently available interaction resources, i.e., interaction resources that are connected to a Web application. The *MineManager* provides

- the type and
- the channel

used by the interaction resources. The *MineManager* meta-UI lets the user influence the interaction resources used.

5.1.2 Composing Interaction Strategies at Runtime

We expect that the user can choose from a large repository of strategies maintained by a community of end-users. Strategies for several interaction resources have been developed as part of this thesis (e.g., strategies for pointer, keyboard and voice input, for an on-screen keyboard¹ and an output strategy for enlarging the font size).

As an alternative, strategies could also be supplied by device vendors wishing to support novel interaction techniques with their devices. For example, the pen input strategy (described in Section 6.1.2) could have been developed by the pen manufacturer.

Interaction strategies can also be implemented with little effort by a business user. As an example, we consider the scenario of a shopping mall. Multiple users (customers) interact with an interactive Web application in the shopping window of a travel agency. To improve the interaction, the clerk of the travel agency could deploy a new or modify an existing interaction strategy to better adapt the UI to the situation in the mall. Similarly, a technically skilled user in a department may devise an interaction strategy for incorporating contextual data such as the current location gathered from sensors as input to an application, e.g., as proposed in the AUGUR system [HSM09].

Automatic Composition Algorithm

At runtime, MundoMonkey chooses which interaction strategies are best suited for the access environment at hand. The decision is based on the available interaction resources in the access environment and the user preferences, as shown in the Algorithm for **Calculating a Juxtaposition of Interaction Strategies** on page 94.

¹ based on the Greasemonkey on-screen keyboard www.greywyvern.com/code/javascript/keyboard

Algorithm: Calculating a juxtaposition of Interaction Strategies.

interaction strategy composition **Input:** installed strategies: availableStrategies

Input: resources from access environment: availableResources

//sort, first by number of handled resources, then by user preference

sort(availableStrategies)

//initially no resources are assigned to any strategies

foreach $r \in \text{availableResources}$ **do**

| usedResources[r] = false;

end

//try out all strategies one by one

foreach $\text{strat} \in \text{availableStrategies}$ **do**

| //try to assign interaction resources to this strategy

| //initially, no interaction resources are assigned to the current strategy

| **foreach** $r \in \text{availableResources}$ **do**

| | tempUsedResources[r] = false;

| **end**

| //counts the number of interaction resources assigned to this strategy

| assigned = 0;

| //resources needed by the strategy are numbered 1 .. strat.nResources

| **for** $i = 0; i < \text{strat.nResources}; i++$ **do**

| | //find first unassigned interaction resource with the correct type

| | **foreach** $r \in \text{availableResources}$ **do**

| | | if (!usedResources[r] \wedge !tempUsedResources[r] \wedge r.type ==

| | | strat.neededResourceType(i)) //assign the i-th interaction resource of the
| | | strategy

| | | //assignment will only be put into effect later

| | | tempUsedResources[r] = true; assigned++; strat.assignResource(i, r);

| | | break;

| | **end**

| **end**

| //could all interaction resource needs of the strategy be satisfied?

| **if** $\text{assigned} == \text{strat.nResources}$ **then**

| | //this strategy is ready and will be used

| | readyStrategies.push(strat);

| | //mark resources as used

| | **foreach** $r \in \text{tempUsedResource}$ **do**

| | | usedResources[r] = true;

| | **end**

| **end**

end

The interaction resources advertise their type, e.g., ‘mouse’ via the middleware. This information is used by MundoMonkey to match them to strategies as shown in 1. Every strategy specifies how many resources they can handle and which types these resources must have. MundoMonkey selects at most one strategy for handling the events from any interaction resource. Thereby, strategies which handle more interaction resources at once are selected first. The reason for this is that MundoMonkey assumes the input of two interaction resources processed in a single strategy is better integrated than in two separate strategies.

When MundoMonkey runs a strategy, it selects a free communication channel in the middleware and assigns this channel to the interaction resource or sensor and as well to the strategy. MundoMonkey thus dynamically creates a choreography of the resources in the access environment. The handling of events is performed by the strategies; however, MundoMonkey steps in whenever the choreography needs to be updated, e.g., because an interaction resource has left the environment, and replans the choreography. The current implementation uses simple string matching to determine whether interaction resources fit the requirements of a strategy, e.g. if a strategy requires the resource type ‘mouse’ the device has to report exactly as ‘mouse’.

Considering User Preferences

If multiple strategies are equally suited (e.g., handle the same number of resources), user preferences come into play. The user can provide a rating (integer value) for each strategy. This rating can be used to break ties. In the current implementation, there is no UI for setting these preferences. This could for example be implemented following the approach proposed in [YNK09], where the user only has to express preferences on pairs of interaction strategies.

Strategies that do not require any input from interaction resources, e.g., the onscreen keyboard, are treated specially. Whether such strategies are applied is only defined by the user preferences. These are stored on a per-application URL basis. These are added to the list *readyStrategies* at the end of Algorithm 1.

5.1.3 Implementation Design

The Greasemonkey Firefox extension can be used to provide output modification capabilities to interaction strategies. If interaction strategies are implemented as user scripts, they can access the DOM as required. However, user scripts lack support for reacting to input events from interaction resources in the access environment. The question arises how this limitation can be overcome, i.e., how JavaScript code executed in the context of a Web page can access a communication middleware.

Other systems [VLC08,AS07] rely on HTTP-based publish subscribe implementation, simulating pushing events onto the browser from the Web server with techniques such as long-polling [Rus06]. The connection to the communication middleware used in the access environment is performed by an HTTP server acting as a proxy. As latency and throughput are essential for efficiently connecting interaction resources in the access environment to the application UI in the browser, we compared the performance of a Firefox extension implementing a MundoCore node with the fastest AJAX-based publish subscribe implementation (Bayeux + CometD + Jetty + Dojo as described in [Rus06]). Thereby, the performance of the AJAX-based solution we tested is better than the performance of the other systems mentioned above, as many optimizations have been implemented.

We tested the performance of both approaches by implementing a simple echo server in JavaScript that subscribes to a “request” channel and on reception of a message publishes to a “reply” channel. A remote client program simulating an interaction resource in the access environment was then used to measure message roundtrip times and maximum throughput. Our measurements show that the Firefox extension significantly outperforms the AJAX-based implementation. Latency is lower by a factor of 3.2–53.5, and throughput is higher by a factor of 16:

	MundoCore Firefox Add-on	Ajax-based
Roundtrip time async (ms)	0.93	N/A
Roundtrip time sync ¹ (ms)	² 15.63	49.80
Throughput ³ (msgs/sec)	10,039.98	627.34

Based on these results, we chose to implement MundoMonkey as an extension, connecting the Web browser to the MundoCore environment. The MundoMonkey extension is based on the C version of MundoCore [Ait09].

¹ Execution synchronized with the UI thread of Firefox. Synchronization is required whenever changes to DOM are desired.

² Limited by maximum Firefox UI update rate (approx. 60 fps)

³ Large number of requests is pipelined until CPU load saturated at 100%

5.2 System Model for Augmenting the Web Browser with Meta Operations

In this section, we describe the design of the extensions for the Web browser adding support for the meta operations introduced in Section 4.3. First, we present a meta model for post-desktop access environments and a corresponding meta object protocol. The meta model serves as a canonical representation of federated post-desktop access environments. Thus, it can be used to *describe* federated post-desktop access environments.

Second, we describe the design of a meta UI, called *MineManager* that operates on instances of the meta model. The meta object protocol prescribes how instances conforming to the meta model may change at runtime. The meta model is thus used as an implementation framework for the meta UI, i.e., the meta UI acts as a view to the representation of the access environment. The meta object protocol establishes causal connections between the model of the federated access environment and the meta UI on the one side, and between the model of the access environment and the actual middleware used in the federated access environment on the other.

The *MineManager* meta UI is *proactive*, which means it performs configuration actions of the access environment without explicit and interactive user confirmation. We formally define policies for taking proactive actions that ensure the user stays in control in a technical sense. However, technical policies necessarily fall short of satisfying the user's desire for a *feeling* of control. This aspect is addressed as well in the design of *MineManager*.

5.2.1 Meta Model and Meta Object Protocol for Post-Desktop Access Environments

Users need to be able to control access environments and the applications they use via meta operations made available in a meta UI (Requirement C&M). Operations the user invokes in the meta UI need to be translated into changes in the configuration of the access environment, and likewise, changes in the configuration of the access environment need to be reflected in the management UI. This two-way synchronization ensures that the digital meta UI stays consistent with the physical state of the access environment.

We propose a meta model for access environments and a meta object protocol for manipulating instances conforming to the meta model to address this problem. The meta model provides a reference framework that can be used in meta UI implementations. The meta model and meta object protocol are rich enough that the operations required in 4.3 can easily be expressed in terms of the meta model. Yet, it is abstract enough that it can be easily instantiated with any implementation technology for access environments.

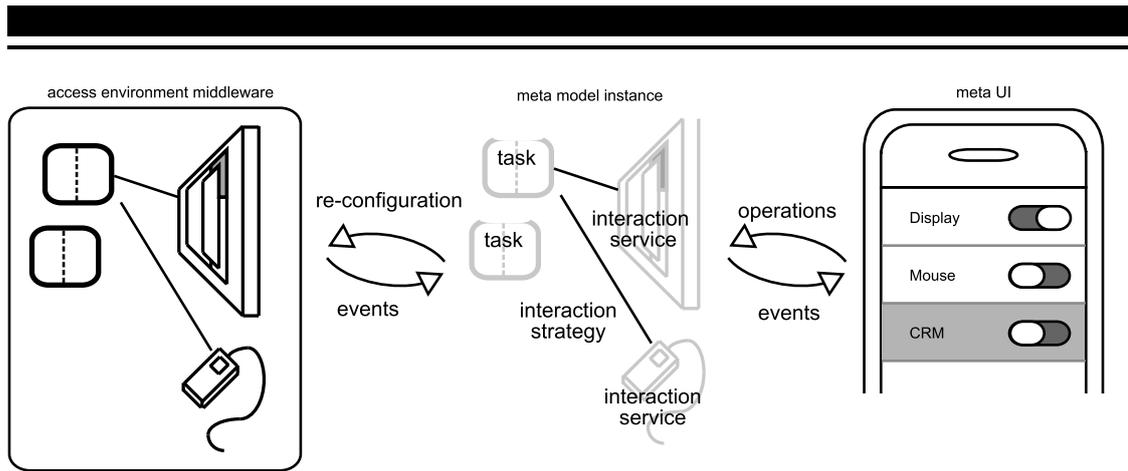


Figure 5.3.: The meta model serves as a canonical representation of the physical access environment. The meta UI links to this and is synchronized by reacting to events.

Figure 5.3 shows the application of the meta model. The representation of the access environment configuration in the digital meta UI is kept synchronized with the actual state in the federated access environment, implemented on top of the MundoCore middleware and the MundoMonkey extension for interaction strategies.

The general problem of synchronizing the digital meta UI and the state of the access environment exists independently from any implementation technology used for the meta UI, which could, for example, be a GUI or a voice UI. Utilizing the meta model as an implementation framework facilitates the development of views in different modalities to the same underlying model.

Meta Model Classes

In the following we will introduce STUD, a meta model for post-desktop access environments and an accompanying meta object protocol. A predecessor of this model was presented by the author as part of [VSL⁺09]. Service - Task - User - Device (STUD) foresees that four classes, shown in Figure 5.4, should be used to model post-desktop access environments.

Device: *Devices* are the hardware components of the access environment. A device provides at least computing and networking capabilities. Devices may also provide other hardware components, particularly hardware that can be used for interacting with the user, such as screens, accelerometers or microphones. Note that this notion of a device differs from the concept of *interaction device*. Devices merely refer to hardware resources present in the access environment available to the system, regardless of how these are perceived by the user.

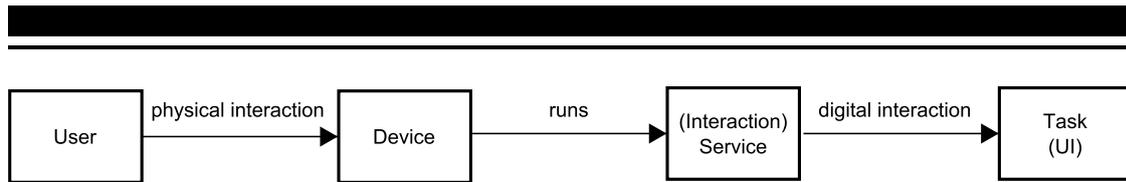


Figure 5.4.: Overview of the elements of the STUD meta model.

Service: A *service* is a functional component. The functionality can be directly related to the hardware of the device the service is executed on, e.g., a mouse service reads out the state of a hardware button on the mouse device and publishes the information to the environment. Such *interaction services* are of particular importance for the scenario of accessing Web applications in post-desktop environments. They directly correspond to interaction resources as introduced in Section 5.1. Other services represent actuators, e.g., for controlling the room heating; yet other services are purely implemented in software and only use the computing capabilities of devices, e.g., a database service. Services are used to model any interaction or context resources in the environment.

Task: *Tasks* represent the activities the end-user can execute to reach a goal. In this thesis, we only consider tasks implemented as interactive Web applications. Thus a task corresponds to a Web page opened with a browser in the access environment. However, in dynamic environments tasks may also emerge dynamically from the present services and devices.

User: A *user* is a human who works on the tasks in the environment using interaction services.

Federated Access Environment as Instance of this Meta Model

A typical access environment for Web applications thus contains the following entities.

- at least one browser service running on a device with comparably high computing power and Internet connection. This service hosts the DOM of Web pages and performs the HTTP communication with the server (background interaction). It may also provide interaction capabilities, although this is not required. In this thesis, when we use the term *Web browser* we refer to this combination of service and device in the access environment.
- a number of tasks, i.e., Web pages whose DOM representation is hosted by the Web browser
- a number of interaction resources. The capabilities of each of these resources is exposed to the environment as interaction service. Ideally, interaction resources are self-contained, i.e., directly run the interaction service. However, as this is not the case for many existing interaction devices, a proxy approach as in [BMRB07] can be used. The goal of this setup is to allow the flexible combination of the Web browser

with the interaction services in the environment, thereby creating a federated access environment.

- we only consider a single human *user* working on the tasks using interaction services in this environment.

All nodes forming a federated access environment (see Definition 5) are connected by a ubiquitous computing middleware. Such middleware replaces the hard-wired communication links that exist between the components in monolithic desktop or mobile access environments. Different types of middleware have been employed to this end, including tuple spaces [JF02], centralized message brokers [VLC07] and peer-to-peer based publish/-subscribe middleware [AKM07], which is used in the implementation and evaluation in this thesis.

Using STUD for Other Purposes than Web Access

Services other than interaction services do not need to be considered for the use case of accessing Web applications. Such other services could, e.g., be *room control services* for controlling the room's air conditioning. Although the STUD model is general enough to support such services, they may be not that important in practice. In many existing systems, such services are already exposed via a Web user interface and can be used from a browser [Goo10]. We do not see a reason why this approach could not be adopted in general, e.g., requiring that all services, be they local or remote, are exposed to the user via a Web front-end.

For modeling Web access environments, only a single user is considered. Tasks and services are not shared among users on the level of the post-desktop environment. Simultaneous access by multiple users has to be handled in the back-end applications. However, the same physical environment, e.g., a room, may host more than one personal access environment, and operations for distinguishing between the resources used by different users in the same physical environment are provided and explained below.

Meta Object Protocol for STUD

The classes of the meta model can be used to represent and describe the state of a post-desktop access environment. The user operations defined in Section 4.3 need to be implemented on top of this model. Likewise, events from the underlying physical access environment need to be defined, so that the meta UI can react to these. An overview of the supported events and operations can be found in Table 5.1. Details are provided in the following paragraphs.

STUD operation / event	Classes	Description
available / unavailable	S,T	A task or service becomes available or unavailable for association. This event is used in the meta UI to maintain a list of tasks and services which can be associated later on.
add / remove	S,T	Services and tasks can be added to and removed from the access environment. Adding can be initiated from the middleware, i.e., when a new service is discovered, or by the user, e.g., when a new application task is deployed.
Present / UnPresent	S,T	Services can be used for interacting with the application task. If one or more interaction services are used to present an application task, they become available as interaction resources for <i>MundoMonkey</i> interaction strategies. The inverse operation can be used to detach a service from a task.
remove connected	S,T	Similar to the remove event; this event occurs, however, when a service becomes unavailable that is currently used in an interaction strategy. In this case, <i>MundoMonkey</i> needs to re-adapt the interaction.
Associate / UnAssociate	S,T	Deploy an application task or interaction resource in the access environment. This operation takes the Uniform Resource Identifier (URI) of the corresponding Web application as parameter. In case an interaction service is associated into the access environment, a middleware-dependent URI identifying the service is used. This operation is always triggered by the meta UI.

Table 5.1.: Overview of the operations and events in the meta object protocol for STUD.

Operations

The meta UI is used to invoke several other operations, as described in Section 4.3. The STUD framework defines an extensive set of operations that can be performed on the entities in an access environment, which could be used to model the meta UI operations: *Present*, *Suspend*, *Resume*, *Migrate*, *Invite*, and *Share*. For the purpose of Web access in federated post-desktop environments, the *Present* operation is needed.

Present(T), Present($T,S+$): Connect a task T with one or more interaction services $S+$. If no target service is specified, the system has to decide automatically which services should be used for interacting with the task. To stop using any services for presenting a task, the *UnPresent* operation with the same parameters can be used.

This operation is used to model the ubiquitous window manager functionality of the meta UI, both for connecting input resources and output resources. It does not prescribe how the presentation is done, e.g., connecting a task with a pointing service has different semantics compared to connecting a task with a voice recognition service. The concrete implementation used is decided by the *interaction strategy* employed for the connection, as discussed in section 4.2.

Only single-user access environments are considered for Web access. Thus, inviting other users (*Invite* operation) and sharing (*Share/UnShare* operations) are not used, as they require another user as argument. However, it is necessary to distinguish between devices and services that belong to the personal access environment of one user and the services and devices generally available.

Instead of the *Share/UnShare* operations used in STUD, we introduce the *Associate / DeAssociate* operation that transfers a service, task or device that is available, e.g., discovered by the middleware, into the access environment of a user. For example, transferring a generally available task into the access environment corresponds to selecting an URL from the list of favorites in the browser. *Association* is thus used to model the *application access* operation of the meta UI.

Associate(URL): Add a *task* or *service* into the access environment identified by their URL. For *tasks*, Web addresses can be used as URLs. For *services* Web addresses could be used too, although, a middleware may use other identifiers for them.

For Web applications we can assume that all applications can directly be associated; they can be associated by entering the URL in an address bar of the browser. However, for interaction services this approach is not feasible. In post-desktop scenarios, interaction services may be available for association where it is not obvious to the user which URL should be used to refer to a certain service. Printing the URL on devices, e.g., as a bar code, so that it can be read by the user is one option. However, it would be much more usable if the meta UI directly shows available, but not yet associated services to the user. This desire leads to a fourth state in the life-cycle of a resource, which is called “available”. The life-cycle of a resource thus has four phases as depicted in Figure 5.5.

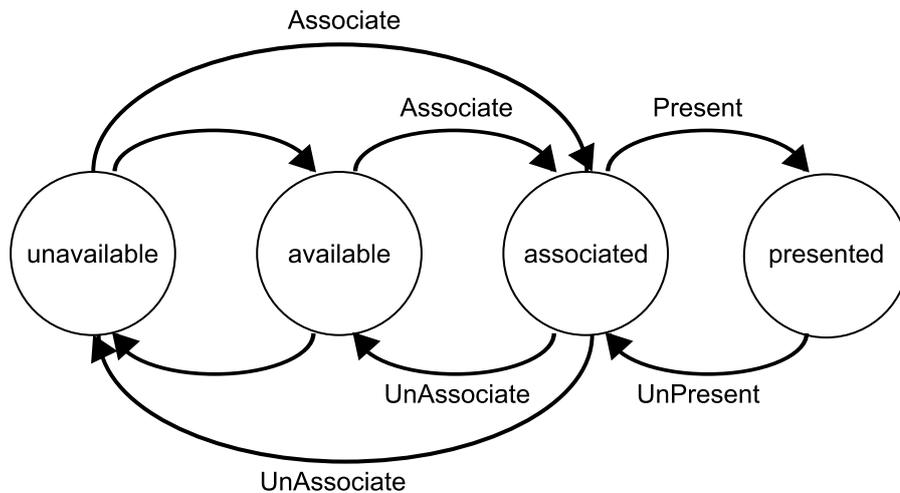


Figure 5.5.: Life-cycle of a service or task. Interaction services necessarily become available before they can be associated. If they are deassociated they become available again. Tasks may be directly associated from an unavailable state. If they are deassociated, they return to unavailable state.

Events

If operations are issued by the meta UI, the view can be adapted. However, the operations may fail, or the access environment may change due to actions outside of the meta UI. To synchronize the meta UI in those cases, the meta object protocol defines events that are signaled whenever a resource changes its state. These events have to be consistent with the physical state of the access environment.

This can be achieved by reacting to the following events in the meta UI generated through physical actions in the environment:

S, T_g+, S, T_g- : A component (*service, task*) becomes available or unavailable for *Association*. The index $_g$ stands for global, as these resources have not yet been associated with a user's access environment.

We allow this event also for *tasks*, as this can be used for maintaining a history, i.e., once accessed tasks become available for further use in an unassociated stage. Association together with these events is used to implement the true meta UI functionality in the Coutaz' sense of the meta UI.

$S, T+, S, T-$: A new component (*service, task*) is added to the access environment or an existing one is removed from the access environment.

In general, a component could be any of the four entities (user ($U+, U-$), device ($D+, D-$), service ($S+, S-$) or a task ($T+, T-$)). However, for the purpose of modeling post-desktop environments for accessing Web applications, ($U+, U-$) events do not oc-

cur. There is only one user in each environment, as the environment is a user's personal access environment. As the only relevant devices are devices providing interaction services, $D+/D-$ do not have to be considered explicitly as they are always accompanied by an $S+$ or $S-$ event. Note that the $(S+, S-)$ events are suitable for maintaining consistency with the physical environment. For example, if the user brings an interaction device into his access environment by moving into a room containing voice recognition equipment, an $S+$ event occurs. The events are used for notifying the views of any updates in the model, thereby allowing the user to inspect the current state of the environment in the digital meta UI. $T+$ and $T-$ events can be triggered by changes external to the meta UI, and thus they need to be reacted upon. By updating the digital meta UI according to these events it can act as a single point of control for the post-desktop environment. Reacting to these events ensures that the digital meta UI stays consistent with the physical environment. This helps to realize the concept of a meta UI reflecting the physical world as discussed in Section 4.3.

Most likely the meta UI will have to react differently if a service that is currently presenting a task goes away as compared to a service which is not used going away. Therefore, the basic events from above are refined, and two new events are added to the framework.

S, T_c- : A component (*service, task*) that was used as operand of a *Present* operation without a corresponding *UnPresent* is removed from the access environment. The index c stands for connected.

Other STUD Operations

Migrating services between devices (*Migrate*) is only meaningful for services that do not require the special hardware of a certain device. As this is always the case for interaction services, this operation does not have to be considered here. It would not make sense to migrate the pointing service from a touchpad device to a microphone. Furthermore, as all computation intensive services can be run in the SOA back-end, potentially in a cloud infrastructure, migration of non-interaction services for performance reasons should no longer be necessary.

The *Suspend(T)* and *Resume(T)* operations are already supported by existing Web browsers, e.g., by keeping the state of different sessions in tabs or windows. Only when the browser instance is terminated is the session state lost. Web browsers also provide different means for manually adding tasks to the environment, e.g., the address bar and bookmarks. Table 5.2 shows the complete mapping of the required meta functionality to the STUD operations.

Sample Scenario

To illustrate how the meta model is used, we consider again the example of the CRM application. Initially the sales representative uses this application on her mobile phone.



Meta Operation	STUD operations
Browser	
application access	<i>Associate</i> (URL of task)
pausing and resuming applications	<i>Suspend</i> (<i>T</i>) and <i>Resume</i> (<i>T</i>)
Meta UI	
adding interaction resources	<i>Associate</i> (URL of service)
removing interaction resources	<i>DeAssociate</i> (URL of task)
Window Manager	
connecting output resources	<i>Present</i> (<i>T</i> , <i>S+</i>), full screen <i>Present</i> (<i>T</i>)
connecting input resources	<i>Present</i> (<i>T</i> , <i>S+</i>), full screen <i>Present</i> (<i>T</i>)

Table 5.2.: Meta operations from Table 4.1. The differentiation between connecting input and output resources is handled by interaction strategies. They are treated as equal in the meta UI.

Thus, the access environment comprises just the input and output resources of the mobile phone, e.g., a screen and touch input. The interaction strategy setup matches this situation.

When the sales representative enters her car, which includes a voice input system, this is discovered by the middleware. An $S+_g$ event occurs as the new service becomes available. The meta UI can react to this event and update the list of available interaction resources. Here, we assume the meta UI also automatically associates the interaction service. To do so, an associate operation is executed, leading to several more events. In the case of the voice interaction resource, it becomes unavailable for association, as it does not make sense to share the voice recognition capabilities.

In a similar manner, there may be a car-specific application that only makes sense to use inside the car, e.g., the control of the car stereo. We assume that this is a Web application, available via a certain URL. The presence of this application and its URL is signaled to the meta UI with a $T+_g$ event.

The meta UI displays a list of all applications that are directly available for association, e.g., in a special bookmarking folder of the browser. When the user selects this application from the list, an *Associate* operation is performed. Usually, this is directly accompanied by a *Present* operation, for presenting the newly associated application task on all available interaction resources. The application does not become unavailable for further associations, as it can be used multiple times in parallel.

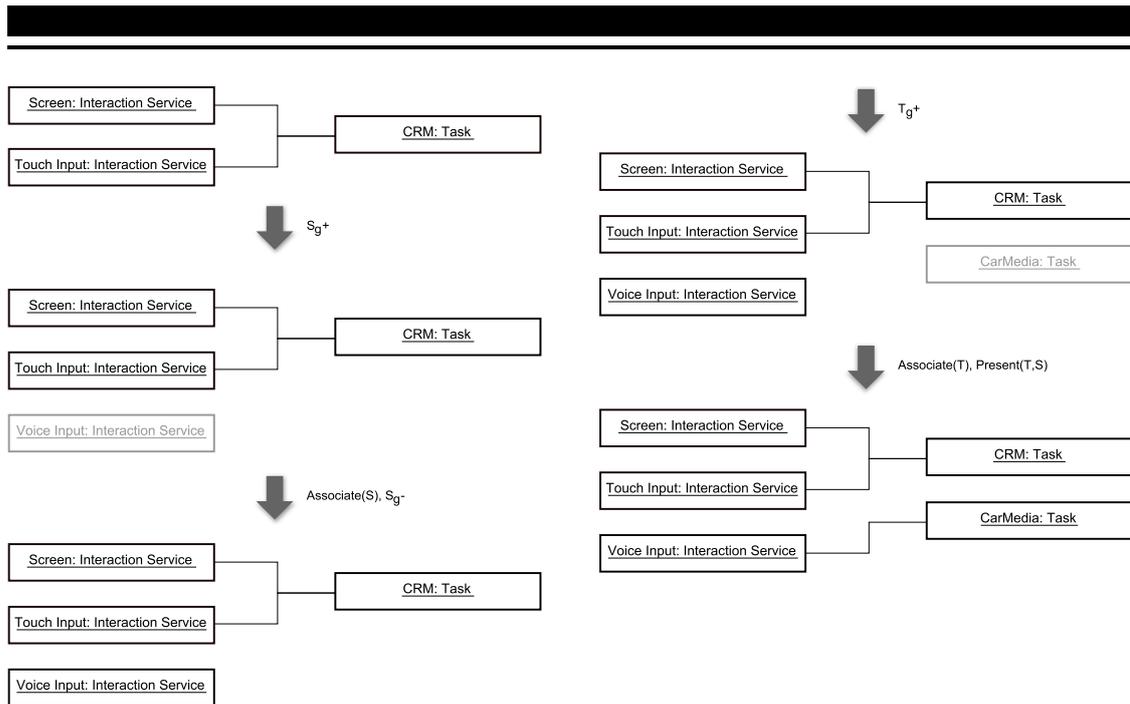


Figure 5.6.: Flow of events in the sample scenario.

In all cases, the state in the physical access environment and the state inside the *MundoMonkey* system (i.e., which interaction resources are connected to which application task) are reflected in the meta UI. Changes in the state cause changes in the meta UI and vice versa.

Figure 5.6 shows a graphical overview of how the scenario unfolds in a series of Unified Modeling Language (UML) object diagrams.

5.2.2 Proactive Meta UI

The events from the middleware are generated automatically and cause updates in the digital meta UI. However, operations such as $Present(T, S^+)$ have to be invoked by the user. As said in Section 4.3 this is problematic, as users dislike the overhead of a digital meta UI. They prefer to have some operations performed automatically. For example, if the user enters her office, any screens should be automatically associated, i.e., the *Associate* operation should be performed in a proactive way. Furthermore, if a new task is accessed it should directly be assigned to these screens via a *Present* operation. However, when the meta UI performs an operation proactively, i.e., without explicit confirmation by the end-user, we run the risk that this operation was unwanted and the user has to roll back the operation.

We formalize the concept of proactive operations by defining some preconditions that must hold before an operation is performed in a proactive way and some procedures that guarantee that these preconditions are met. This approach ensures that the user can trust the meta UI to act on his behalf in the technical sense.

Proactivity Properties

The operation Op on services or tasks P (parameter) may only be carried out in a proactive way if the following criteria are met.

1. P must find the user *trustworthy* to perform the operation Op by checking digitally accessible properties of the user (we write $(P, user) \in T_{Op}$).
2. The meta UI uses digitally accessible properties of P to judge whether P is qualifies for the operation Op ($(user, P) \in T_{Op}$).
3. The operation can only be performed when the user *intends* to do so ($(Op, P) \in I$).

Thus, the operation should be performed if these three properties hold:

$$(P, user) \in T_{Op} \wedge \tag{5.1}$$

$$(user, P) \in T_{Op} \wedge \tag{5.2}$$

$$(user, P) \in I \tag{5.3}$$

The procedure for establishing mutual T_{Op} may vary from use case to use case, e.g., the user needs to provide credit card data to P , and P must provide some certificate to the user and so on. This procedure can be automatically performed by a digital representation of the user and the task or service. In contrast, automatically detecting the user's intent is almost impossible. Still, establishing the three properties must be unobtrusive for the user and involve with minimal interaction, otherwise the meta UI would not be perceived as proactive. The following process should be supported in the meta UI to achieve this goal.

1. The user decides if $(user, P) \in I$. The I relation is ultimately defined by the user, even if this is done subconsciously most of the time. Explicitly modeling this step makes sure that the user is in control.
2. The user somehow signals this to her meta UI using an *authorization procedure* described in section 5.2.2.1. Thereby she authorizes the meta UI to perform the rest of this procedure automatically.
3. Meta UI and task or service perform a *qualification procedure* to establish mutual T_{Op} . *Qualification procedures* usually involve a user's providing of something (information or money) to prove trustworthy to the task or service. Conversely, the task or service may also provide something to prove trustworthy to the user. *Qualification procedures* are further described in section 5.2.2.1.

-
-
4. If the *qualification procedure* succeeds, the operation is performed.
 5. If the *qualification procedure* fails, e.g. if the task or service reject the association, this is signaled to the user.

Qualification Procedure

The role of the *qualification procedure* is to establish mutual T_{Op} . Several options for implementing this on the side of the service or task and the meta UI exist. Usually the service or task will require some proof for the trustworthiness from the user which is provided automatically by the meta UI. The meta UI is responsible for delivering this proof of trustworthiness only if it has been authorized to do so by the user via an authorization procedure described below. In turn, the meta UI may require additional proof of trustworthiness from the service or task.

Typical examples of qualification procedures used for Web access are cookies, provided by the browser (acting as a meta UI for the desktop), and certificates provided by the Web server. In this case, the qualification procedures only check digital properties. For some operations on services in post-desktop environments, other approaches can be used. For example, a qualification procedure may require the transmission of data via out-of-bounds channels, e.g., a *location-limited* IR channel [BSSW02].

Authorization Procedure

Authorization procedures are built into all interactive qualification procedures. For example, using some out-of-bounds channels requires user interaction, e.g., SyncTap [Rek04]. Thus, the qualification procedure ensures that the user authorizes the operation. The user would not perform the necessary steps for the authorization procedure unless $(user, P) \in I$ holds. Completely automating the process would require that digital properties could also be used to determine whether I exist.

This is problematic, as methods for determining that the user's intend to perform an operation are error-prone and will therefore lead to an unsatisfied user. As I cannot be detected directly, we rely on authorization procedures, i.e., actions that the user must perform and after which the meta UI may assume I for a task or service is given. Introducing the abstract concept of an authorization procedure opens the opportunity to provide several implementations ranging from interactive, for risky operations, to completely automatic for low-risk operations. Authorization procedures can be implemented in several ways, all supported by MineManager:

Static List For the association operation, the simplest form of authorization is a static list of task and services. This list could, for example, contain personal items of the user that should always be associated, like a wrist watch screen service. Whenever a task

or service from the list is encountered, the meta UI thus will automatically try to associate the task or service.

Authorizing Real World Actions Another option is to employ techniques like the already mentioned SyncTap [Rek04] or [HMS⁺01]. Association of a public screen could for example be performed when a button on the screen and a button on the meta UI device are simultaneously pressed. This procedure is slightly less convenient than automatically associating the service but provides more control to the user. On the other hand, it is still more intuitive compared to the third option.

Digital Interaction The last and most intrusive option for authorization is to use the digital meta UI, i.e., invoke the operation explicitly. Operating on the digital meta UI is often perceived as overhead by the user. However, sometimes digital interaction is preferred to physical interaction.

Discussion

The STUD meta model introduces the additional step of *association* compared to other models for managing post-desktop environments. By explicitly considering this step the resulting framework is more flexible compared to the existing frameworks, which do not have a state comparable to the *unavailable* state in the life-cycle of a resource (cf. Figure 5.5). Having such a state permits introduction of location based service selections, e.g., as later proposed by the EnvB [LV11] system, where all locally present resources are put into the unavailable state from where they can easily be deployed.

Of course, having such a fine-grained set of operations and events often causes unnecessary complexity for the user. For example, when a new task is associated, it will usually also be directly presented using some or all available interaction resources. By making the meta UI proactive, the additional complexity can be hidden from the user. The MineManager framework provides the means to do this.

5.3 Latency-Reduction Algorithms

Extending the Web browser with interaction strategies improves the interaction with a single Web page. However, the overall usability of the Web application is also determined by the latency the user perceives for server roundtrips, i.e., for switching from one Web page in the application to another or to access services in the application back-end. The processing time of the back-end services stays the same, regardless of whether the Web application is accessed from a desktop or post-desktop environment. The network latency and client overhead is, however, larger for many post-desktop access environments, compared to their desktop counterparts. The reason for this is that post-desktop environments have to rely on a mobile network connection and can only use the limited computing re-

sources of mobile devices. These factors add up and result in a comparably high latency perceived by the user, threatening the overall application usability.

In Section 4.4 we outlined a general approach for reducing the user-perceived latency through caching and prefetching. In this section, we present a concretization of these concepts for SOA-backed Web applications. For SOA-backed Web applications accessed in post-desktop environments, the user perceived latency comes from three sources:

- processing time of the back-end services
- network latency
- proxy and client overhead

We present algorithms for predicting and prefetching requests to a SOA back-end, concretizing the general approach for prefetching of back-end requests outlined in Section 4.4.1. Furthermore we present algorithms for synchronizing a cache on a mobile device with the cache on the proxy, relying on the piggybacking mechanism.

5.3.1 Proactive Prefetching of SOA Requests

In general, prefetching of SOA requests means requests to back-end SOA services are performed in the idle time of the user and stored in a cache. From there, they can be retrieved at low latency, avoiding the processing time taken by the back-end services. Depending on the location of the cache, the network latency and proxy overhead can also be avoided.

To determine the requests to prefetch we use *discrete sequence prediction algorithms* [LS94]. Using an automatic algorithm frees the application developer from having to deal with latency explicitly. As a side effect, using an automatic algorithm for calculating the prefetched requests permits adaptation of the prefetching to the individual user at hand, which outperforms statically preprogrammed models [HSM08].

To describe the algorithm we apply for predicting SOA requests we introduce the following symbols:

- q_i - The i th request from the user to the SOA back-end.
- r_i - The response from the SOA service in the back-end to q_i .
- s - The complete sequence of requests issued by the user and their responses (q_i, r_i) . This is a hypothetical construct, as it also includes future requests which have not yet been made.
- h - The observed history of s , i.e., the requests already issued by the user and their responses.
- a - A subsequence of adjacent request/response pairs of h .

- \circ - The sequence $a \circ (q_{j+1}, r_{j+1})$ is the concatenation of a and (q_{j+1}, r_{j+1})
- $A - Z$ - Uppercase letters denote a concrete request q_A yielding a concrete response r_A . For example, q_A could mean invocation of the operation \circ of service \mathfrak{s} with parameters \mathfrak{p} . The resulting response \mathfrak{r} would be written as r_A . Concrete sequences are coded by concatenating request/response pairs, e.g., ABA .
- $P_n(q|h)$ - The probability that request q is performed by the user in the next n requests. For example, $P_1(q_A|AA)$ is the probability that q_A is immediately issued again, after the user already has issued q_A twice in a row, each time resulting in the response r_A .

The problem the algorithm must solve can thus be described as follows:

Given an observed sequence of user requests and service responses h , return a set of predicted requests Q , such that $\forall q \in Q. P_n(q|h) > T$ for a defined threshold T .

This request can then be executed against the SOA back-end, and the result can be stored in a cache. Thus, the latency caused by the *processing time of the back-end services* is avoided, should the client actually make the request.

Prediction Algorithm

We use a variant of the FxL algorithm for predicting SOA requests. FxL has been co-developed by the author [HS07]. The FxL algorithm was used to predict input symbols a user enters into a UI. Several other similar algorithms could be used instead of FxL, e.g., ActiveLeZi [GC07] or the one presented in [JB02]. However, FxL has been shown superior to these algorithms for the purpose of predicting user actions [HS07].

FxL works as follows. A *frequency table* storing the number of occurrences of all subsequence up to a length l is maintained and updated after each observed request/response pair. We write $fr(a)$ to denote the frequency entry in the table for subsequence a . FxL also maintains the set of all observed requests Q and all responses R . For example, if $h = ABCAB$ and $l = 2$, the following entries are stored in the frequency table:

$fr(A)$	2
$fr(B)$	2
$fr(C)$	1
$fr(AB)$	2
$fr(BC)$	1
$fr(CA)$	1

Based on the frequency table FxL calculates *scores* for requests as follows:

$$score(q) = \sum_{i=1}^{l-1} i \sum_{r \in R} fr(a_i \circ (q, r))$$

Based on these scores, an estimated occurrence probability $p(q)$ for the request q is calculated by normalizing all scores.

$$p(q) = \frac{\text{score}(q)}{\sum_{q' \in Q} \text{score}(q')}$$

Finally, the set of actually predicted requests $PRED$ takes T into account.

$$PRED = \{q \in Q | p(q) > T\}$$

These predicted requests should then be considered for prefetching. Several enhancements to the FxL algorithm need to be introduced in order to account for the peculiarities of prefetching SOA requests, where some requests are not eligible for prefetching. We discuss these in the next section along with parametrization options for the algorithm. The modifications are

- Excluding services from prefetching
- Correcting for excluded services in the prediction
- Parametrization of the algorithm.

Excluding Requests from Prefetching

Prefetching should only be applied to uncritical requests, e.g., for read-only service operations. It could be dangerous if a request due to a prefetch operation would cause irreversible changes that are not desired by the user. Applying prefetching therefore requires an option to mark requests as non-eligible for prefetching. This can be done either in the description of individual services or within the application making use of the services.

Web Service Description The first option is to leave the decision whether prefetching is possible or not to the service in the SOA back-end. In architectures adhering to the Representational State Transfer (REST)-style [Fie00] it can be inferred from the HTTP method used, whether a request can be prefetched, e.g., calls using the HTTP `POST` or HTTP `DELETE` method are excluded from prefetching. Many business applications rely on Simple Object Access Protocol (SOAP) style services as opposed to REST-style services, where this cannot be done. One alternative would be to introduce additional flags in the service description, e.g., the Web Service Description Language (WSDL) file. Potentially, the decision whether a service call can be prefetched or not can be automatically derived from semantically richer descriptions, using formats such as Web Service Modeling Ontology (WSMO) [LPR05] or Web Service Description Language with Semantic Extensions (WSDL-S) [AFM⁺05], instead of plain WSDL.

Application In cases where the decision cannot be made based on the service description, e.g., because the respective flags are not provided by the services used, it has to be encoded within the application. This slightly increases the development effort for the application. The effect is, however, barely noticeable, as the developer needs to specify some information about the services used anyway, e.g., the URL of their WSDL files. Specifying whether a request is eligible for prefetching just results in setting a boolean flag.

From the viewpoint of the middleware in the proxy, it does not matter where the decision for marking a service call as non-prefetching has been made. Thus, in the experiments we performed for evaluating the performance of our approach we used the second option as it allowed use of existing SOAP services, without having to change their description.

Correcting History for Excluded Requests

Although services not eligible for prefetching are not prefetched, they still form part of the history of observed requests and can be used to calculate further predictions. However, in our experiments we learned that excluding these read-only requests from the history actually provided better prediction results. This is probably due to the fact that the order of read-only requests is relatively stable in an application. Prefetchable requests are used to get to desired information items, and thus are more likely to repeat, e.g., the user visits the same folder in an hierarchy over and over again. In contrast, read-only requests represent manipulation of objects, and these are more random and should be treated as noise.

Prediction Parameters

Prefetching can be controlled by means of several parameters.

maximum sequence length L The maximum sequence length l denotes the length of the history window that is used to construct the prediction model. Greater values may achieve better prediction quality, but they also lead to higher memory requirements and probably to overfitting as well. Our experiments suggest that $l = 4$ yields good results, which is consistent to the findings of [HS07].

minimum confidence C The minimum confidence T value denotes a threshold for the prediction confidence. If the probability for a request is below this threshold, it will not be considered for prefetching, even if it is the best ranked request. This avoids requesting unnecessary data in cases where the next request cannot be predicted precisely.

maximum number of prefetched requests N The maximum number of prefetched requests M limits the number of requests that are processed for prefetching. Even if the estimated probability of more than M requests surpasses the threshold T , only

the best M are considered for prefetching. The limitation of this value can save processing time on the proxy and most importantly also limits the amount of data in the cache.

Algorithm: Serving Client Requests (simple)

Input: client request q

```
idle = false;
if  $q \in \text{proxy-cache}$  then
  |  $r = \text{proxy-cache}[q]$ ;
else
  |  $r = \text{perform-request}(q)$ ;
end
send( $r$ );
 $\text{proxy\_cache}[q] = r$ ;
 $h = h \circ (q,r)$ ;
idle = true;
start Prefetching Algorithm in new thread;
```

Prefetching Requests in the Proxy Cache

The *proxy cache* resides on the proxy server where the complete idle time can be used to prefill this cache, as battery runtime is not an issue. Whenever a request from the client arrives at the proxy, the proxy first consults the cache. If the request cannot be answered from the cache, the service in the SOA back-end is called. The result is immediately sent to the client, as shown in Algorithm **Serving Client Requests (simple)**.

After this has happened, the proxy enters the *prefetching phase*. The last request and response are appended to the history of observed requests h . The rest of the prefetching is performed in a new thread, so that new client requests can be handled immediately. The prefetching follows Algorithm **Prefetching**. Based on this history, a set of predictions p is calculated, as described in Section 5.3.1.1. The predicted requests not already present in the cache are prefetched and added to the cache.

5.3.2 Piggybacking and Distributed Cache

When implementing a cache for SOA requests for Web applications, the question arises where this cache should be placed. Implementing the cache on the proxy allows avoidance of the latency overhead caused by the processing time of the back-end services. The latency caused by the processing on the proxy and the network latency are still perceived, even for requests that can be answered from the cache. If the cache is implemented locally at

Algorithm: Prefetching

Data: history of observed requests / response pairs h ; idle flag

```
 $p = \text{predict}(h)$  while  $idle \wedge p \text{ not empty}$  do  
   $q = \text{pop}(p)$ ;  
  if  $q \notin \text{proxy-cache}$  then  
     $r = \text{perform-request}(q)$ ;  
     $\text{proxy-cache}[q] = r$ ;  
  end  
end
```

the client, within the post-desktop access environment, these two sources of latency can be avoided.

However, directly adding all prefetched requests into a cache located at the client requires that the network connection between client and proxy is kept open during the idle periods. At least for mobile post-desktop environments, this would quickly drain the battery of the mobile device.

To get the combined advantage of a *proxy cache* and *client cache*, without compromising the battery runtime in mobile post-desktop environments, MundoProxy uses a distributed cache, where the *proxy cache* and *client cache* are synchronized using the piggybacking protocol.

Piggybacking Prefetched Results to the Client Cache

Piggybacking is used to transmit contents from the *proxy cache* to the *client cache*. To explain the piggybacking protocol, we consider the example in which the user performs an *original request* q_A which cannot be answered from the *client cache*. The proxy server invokes the back-end service and returns r_A to the client. The idea of piggybacking is as follows. Instead of returning only the *original response* r_A to the client, the proxy piggybacks *additional data* consisting of prefetched requests and responses, e.g., (q_B, r_B) . At the client, this additional data is added to the *client cache*, potentially generating more *client cache* hits.

Correcting for Client Cache Hits

One major problem of history based prediction is that the algorithm suffers from the additional cache hits it generates. Once a request is prefetched and stored in the *client cache*, the proxy server does not know whether it is used or not. This means that the request history observed at the proxy does not necessarily reflect the real request sequence of the user. To overcome this problem, we follow the approach of [FCLJ99] and keep a

Algorithm: Sending Requests on the Client

Input: user request q
Data: client-cache, list of cache hits C

if $q \in \text{client-cache}$ **then**
 $r = \text{client-cache}[q]$;
 $C = h \circ (q,r)$;
else
 $r = \text{send-request-to-proxy}(q,C)$;
 $C = \text{null}$;
end

separate sequence history on the client to record cache hits and transfer this history to the proxy with the next request. Maintaining a list of cache hits and communicating them to the server implies additional complexity and data transfer, but to a very small extent which is far outweighed by the improved prediction quality achieved.

The client uses the Algorithm for **Handling Proxy Responses at the Client** to perform a request. The cache hit history is sent to the server with the original request. Sending the request to the proxy is an asynchronous operation in the client, thus the client application does not block. The response is handled by Algorithm **Handling Proxy Responses at the Client**.

Algorithm: Handling Proxy Responses at the Client

Input: proxy response r, P
Data: client-cache, user request q

ui-update(r);
client-cache[q] = r ;
foreach $(q,r) \in P$ **do**
 client-cache[q] = r ;
end

To account for these two improvements, the algorithm for handling client requests at the proxy is changed as shown in Algorithm **Serving Client Requests**. The predicate *piggyback – more* is used to stop piggybacking more data, e.g., if a given total message size is reached, a certain number of request/response pairs have been sent to the client, or the *proxy cache* only contains entries predicted below a certain threshold T_{client} , which can be different from the T used in the FxL prediction algorithm.

Algorithm: Serving Client Requests

Input: client request and client cache hits q, C

```
idle = false;
if  $q \in \text{proxy-cache}$  then
  |  $r = \text{proxy-cache}[q]$ ;
else
  |  $r = \text{perform-request}(q)$ ;
end
while  $\text{piggyback-more} \wedge \text{proxy-cache not empty}$  do
  |  $p = \text{select entry from proxy-cache}$ ;
  |  $\text{remove } p \text{ from proxy-cache}$ ;
  |  $P = P \circ p$ ;
end
 $\text{send}(r,P)$ ;
foreach  $c \in C$  do
  |  $h = h \circ c$ ;
end
 $h = h \circ (q,r)$ ;
idle = true;
start Prefetching Algorithm in new thread;
```

Piggybacking Temporal Behavior

The reason why piggybacking data is more efficient than using an additional request in the idle time for transmitting *additional data* to the client is the following. Setting up the radio connection on the mobile device comes with a relatively large overhead. This starts with the radio chip tuning to the base band and ends with the Web browser instantiating an XMLHttpRequest object. By using piggybacking, we increase the exploitation of the connection that has to be opened anyway due to the original request of the user. Table 5.3 shows typical values for the delay (and the bandwidth) of mobile networks.

In the most simple implementation of piggybacking, which was used for the integration with ‘AjaxWeaver’ described in Section 6.3, the response to the *original request* was sent together with the *additional data* as one chunk of data, as indicated in the algorithm above with one single `send` statement for transmitting r and P to the client. Likewise, the client only continued processing the results when both, the *original response* and the *additional data* had been received.

This approach introduces a rather large prefetching overhead, because the response to the *original request* is only made available to the user after the prefetching has been

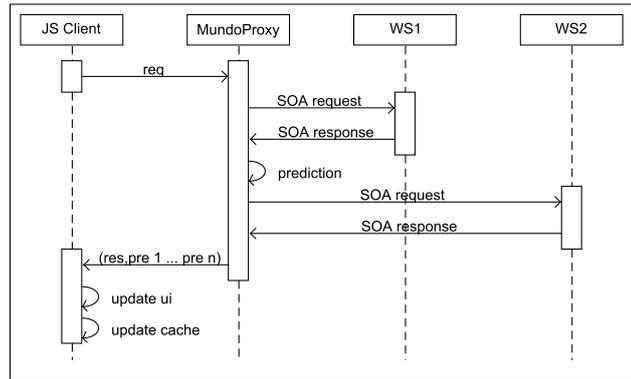


Figure 5.7.: Sequential implementation of piggybacking

completed. Still, an overall net benefit is possible if the additionally generated *client cache* hits outweigh the prefetching overhead, as shown in the sequence diagram in Figure 5.7.

The overhead of this is small if P is small in comparison to the network bandwidth. However, in reality, P can be quite big, so that transmitting P to the client induces a noticeable delay. To avoid this, the proxy transmits the *original result* first, as soon as it is available, followed by a boundary symbol. On the receiving side, the client continuously parses the response to check whether the boundary symbol has already been received. If this is the case, the update of the application is immediately performed. In the background the proxy may send further piggybacked data to the client, which is parsed and added to the cache. Figure 5.8 shows a sequence diagram of the improved behavior.

As the same amount of data is sent from the proxy to the client, besides very few control characters marking the boundaries between the results to the original request and the prefetched requests, the total duration of the network connection is not longer in this implementation than in the previous implementation of piggybacking. The latency overhead of prefetching is almost completely removed, and not noticeable compared to the normal variance of request times.

Preset	min delay [ms]	max delay [ms]	Upstream [kbit/s]	Downstream [kbit/s]
GPRS	150	550	40.0	80.0
EDGE	80	400	118.4	236.8
UMTS	35	200	128.0	1920.0
HSDPA	35	200	348.0	14,400.0
local	0	0	∞	∞

Table 5.3.: Network characteristics according to [Goo09]

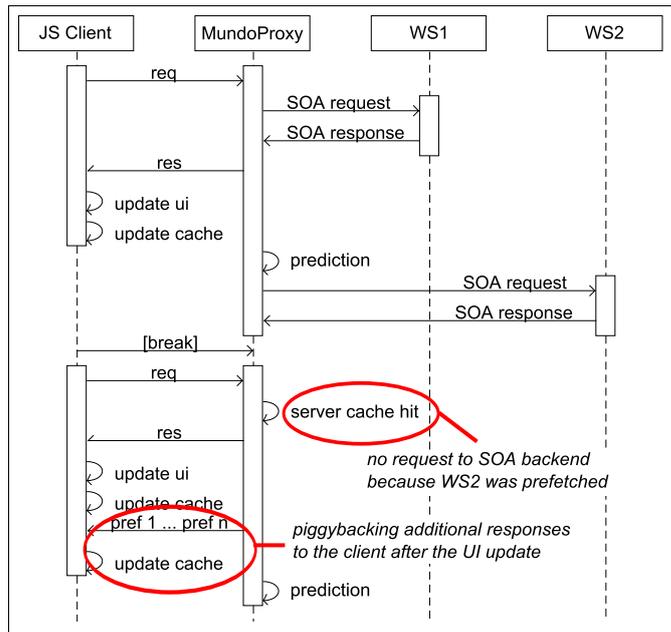


Figure 5.8.: Prefetching is performed in idle periods on the proxy, the client continuously parses the response text and the UI update is performed before the piggybacked results are added to the cache.

5.4 Chapter Summary

We have shown that it is possible to address requirements R2, R3 and R4 for a single use case. To evaluate the MundoMonkey approach in more depth, we have to show that is indeed possible to implement a large number of interaction strategies in a generic way, i.e., able to work with any Web application. Otherwise, installation of Web-user-interface-specific strategies would be needed, which would impose severe scalability problems considering the huge number of existing Web applications. Furthermore, we have to show that it is possible to support a wide range of input devices and interaction techniques by using MundoMonkey strategies.

Thereby the focus will be on interaction techniques that cannot be realized by synthesizing mouse and keyboard events, e.g., interaction techniques that require the combination of output modification and input handling, as these cannot be easily realized with existing systems. We already presented one example of such an interaction strategy, the MLI strategy.

We presented the STUD framework consisting of a meta-model and meta object protocol for handling meta interaction. The framework focuses on enabling Web access for Web access in post-desktop environments. However, it contains useful elements which can be

reused for other applications in post-desktop environments. STUD supports the understanding of post-desktop environments by providing a conceptual framework. And, even more important, it supports the implementation of meta UIs for Web access in post desktop environments that are proactive and causally connected to the physical state of the environment.

We presented algorithms for automatically prefetching SOA service calls in a SOA backed Web application. We adopted the FxL algorithm for this purpose as the results in [HS07] show that the algorithm performs well in similar settings. Furthermore, we presented an architecture and algorithms for a distributed cache. The focus has been on an energy efficient implementation of cache synchronization which is of high importance in mobile access environments. Piggybacking permits utilization of idle time for prefetching, yet it conforms to the energy requirements of mobile devices.

The system has the potential to reduce the latency perceived by the user while interacting with an SOA-backed Web applications. It is agnostic to the nature of the services used and also to the application; as such, it can potentially be integrated into any access system for SOA-backed Web applications. Compared to other approaches for prefetching and caching, our approach does not require any manual configuration of the prefetching algorithm (see Table 5.4). The evaluation presented in Section 6.3 will show that our system performs well compared to a gold-standard, and that prefetching and prediction generate considerable benefit.

	automatic prefetching	energy-efficient utilization of idle times
[EJM00]	✓	✗
[Cao02]	✗	✓
[ED05]	✗	✓
MundoWeb	✓	✓

Table 5.4.: Comparison with other prefetching and caching approaches.

Chapter 6

Implementation and Evaluation

Evaluating research on user interface systems is inherently difficult [Ols07]. There is no standardized task set or workload which can be used to directly compare one system against another. Comparison of research systems is further complicated by the fact that although described in the literature, the system implementations are not easily available. In this thesis, we take a pragmatic approach, performing quantitative evaluation of various aspects and subsystems where possible, e.g., MineManager GUI and the performance gain of caching, while using qualitative techniques for other aspects, e.g., case studies for the implementation of MundoMonkey strategies.

The chapter is organized as follows. Section 6.1 presents three case studies of interaction strategies implemented with MundoMonkey. These case studies illustrate the flexibility of the MundoMonkey approach and show that requirements R2, R3 and R4 are addressed. Section 6.2 reports on the implementation of the MineManager framework for a ubiquitous computing middleware and on user experiments conducted with this implementation. Finally, Section 6.3 reports on the evaluation of the caching and prefetching algorithms. These were performed on two synthetic datasets, as real datasets are not available. The second dataset, however, achieves a level of realism which has not been achieved in the related work. Section 6.4 summarizes the evaluation results of all three components.

6.1 MundoMonkey Evaluation

The MundoMonkey system was implemented as a Firefox extension, according to the design considerations explained in Section 5.1. We evaluated whether it is actually feasible to implement interaction strategies for accessing Web applications from various post-desktop access environments.

The author and others applied MundoMonkey for implementing interaction strategies for various post-desktop access environments. The target access environments used in these case studies were selected so that all features of MundoMonkey could be tested, although they were also partly motivated by user needs. Thereby, the goal of these case studies was to clarify whether requirements R3 and R4 have been met, i.e., whether it is feasible to implement, based on MundoMonkey,

- interaction strategies for various input, output and context resources in an application-independent way, and
- if these can be automatically combined at runtime as proposed in Section 5.1.

6.1.1 Voice Input Strategy Case Study

As the first case study for MundoMonkey, we present an interaction strategy for voice input. Thereby, the voice input device connected to the Firefox browser is the Talking Assistant [AKM04]. The Talking Assistant is a small device that is always carried by the user. The idea is that the user in a mall can use the voice recognition capabilities of the Talking Assistant with a browser in an interactive shopping window. If somebody wearing a Talking Assistant approaches the shopping window, the Talking Assistant is connected to the browser and can control the Web page loaded in the browser via voice commands. Such a browser could, e.g., be deployed in a shopping mall.

The problem solved by the voice interaction strategy is to provide a recognition grammar for the Web page at hand to the Talking Assistant. This has to be done without any specific knowledge about the Web application, as otherwise a new strategy for every interface would be required.

The goal of this case study is to

- implement a usable interaction strategy in an application independent fashion, proving that it is technically feasible to implement interaction strategies independent of the application
- use input from interaction resources in a federated access environment, showing that it is technically feasible to support federated access environments with MundoCore and MundoMonkey



Figure 6.1.: Sample page used in the user study. The character link could be activated by saying "Click on character, please".

- provide an agreeable usability for a moderate effort, showing that it makes sense to try to perform adaptation by interaction strategies

Report

Thanks to the access to the DOM of the Web interface in the browser, the strategy can be implemented in a Web-interface-agnostic way. The recognition grammar provides phrases for controlling every interactive element of the Web page, extracted from the DOM tree. The content of the element is padded with fixed phrases for making the interaction more natural. To access a link on the page, the user can e.g. say "click on <link text>, please". For interactive form fields the fieldname is not so easy to extract. We used existing algorithms for determining the labels of interactive elements on the Web page, [LHML08, HM09]. Once the page is loaded and processed, the resulting grammar is sent to the Talking Assistant via the MundoMonkey Extension. Figure 6.2 shows an example for a rule generated for a link with the label "character", more complicated rules may also involve variables, i.e. inline dictation, whose values are passed back to the strategy as parameters. The grammar is specified in the MS SAPI5.3 format¹.

Once the grammar has been sent to the Talking Assistant, the reactive part of the strategy handles all recognized utterances. To do so, it stores a specific callback function for every grammar rule. The callback functions are generated while processing the page and creating the grammar. For example, matching of the rule in figure 6.2 results in following the "character" link in the Web page. Our voice interaction strategy supports different actions for the different HTML form element types and HTML links.

¹ [http://msdn.microsoft.com/en-us/library/ms723632\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms723632(VS.85).aspx)

```
<rule id="command4">
  <O><O>please</O>
  <L>
    <P>select</P>
    <P>click<O>on</O></P>
  </L>
</O>
<P>character</P><O>please</O>
</rule>
```

Figure 6.2.: One rule of the grammar generated for the sample page.

User Study

To evaluate whether our voice interaction strategy provided usability comparable to other state-of-the-art voice user interface techniques that do not benefit from the flexibility of MundoCore and MundoMonkey, we compared it against the built-in voice control of the Internet Explorer in Windows Vista.

The study was conducted using a within subject design. Participants were members of our department and students ($n = 10$). Every participant completed a task with the Firefox browser, augmented with our voice input strategy extension and the Vista voice control for Internet Explorer. The Talking Assistant also used the Vista Speech Recognizer, so we reduced the difference to just the mapping of speech recognition results to actions in the Web page. This procedure allowed us to test our interaction strategy against the built-in Internet Explorer strategy. See Figures 6.4 and 6.3 for an overview of the setup in both conditions. Although we tested only one Web user interface in the user study, the strategy works with other Websites, e.g., the one of E-Bay or of online travel-agencies. As the example shows, a strategy can be efficiently implemented without tailoring it to a specific Website.

The task performed by the participants was to gather information from Wikipedia articles, which could be easily replaced by the contents of a shopping catalog in the mall scenario. Participants had to scroll down twice (they were told to "Find the information on the bottom of the page"). Then they were instructed to select a certain link ("follow the third link in the list") and then select a link of their choice ("follow any link on this page that interests you"). After the task, participants filled out the SUS usability questionnaire [Bro96]. The order of conditions was counterbalanced to control for learning effects. We found the ratings of our strategy ($M = 62.5, SD = 17.16$) were significantly higher than the rating for the Internet Explorer voice Interaction ($M = 51.5, SD = 18.33$) using a dependent samples t-test ($t(9) = 2.45, p < .05$), see figure 6.5. The effect size was medium to large with $Cohens'd = 6.4$.

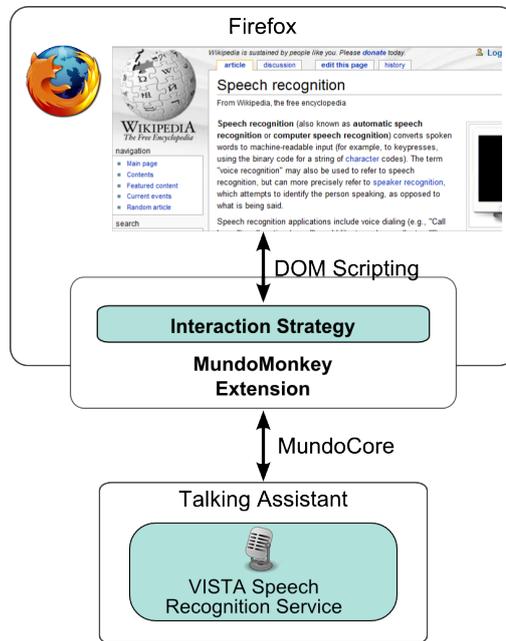


Figure 6.3.: Setup in the voice-interaction-strategy condition used in the study



Figure 6.4.: Setup for the built-in Internet Explorer strategy condition used in the study

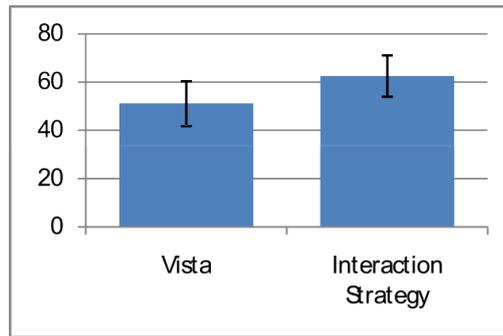


Figure 6.5.: Our voice-interaction strategy achieved a significantly higher SUS usability score compared to the built-in Internet Explorer strategy.

Results

The MundoMonkey voice strategy achieved a higher SUS score than the conventional Internet Explorer strategy. However, accurately measuring and comparing the usability of the two strategies was not the main goal of this experiment. Rather, we wanted to prove that one can implement strategies of comparable usability to commercial systems with MundoMonkey. Therefore we did not use a larger sample size or obtain detailed quantitative data.

The reason why the voice strategy of MundoMonkey achieved a higher SUS score is probably the higher recognition rate of the Vista Speech Recognizer when using a small recognition grammar for the Website at hand (as in the voice-interaction strategy) compared to the dictation grammar used in the Internet Explorer condition.

Implementing voice interaction as *interaction strategy* within MundoMokey has several advantages over embedding it in the operating system or the browser, as e.g. done in Windows Vista. The recognition is done on the Talking Assistant, which exclusively is used by a single user. This allows the voice-recognition engine of the Talking Assistant to be highly customized for a single speaker, greatly improving recognition performance. Further, the flexibility of MundoCore allows different Talking Assistants to dynamically associate with the browser and interact with the Web application, as required in a mall which is populated by many users.

6.1.2 Pen-Input-Strategy Case Study

In several instances MundoMonkey was used to combine the Web browser with input from a digital pen using the Anoto technology [ANO]. One example where such an input modality benefits the user is a online guestbook, where the user can leave painted notes for other users, as shown in 6.6.

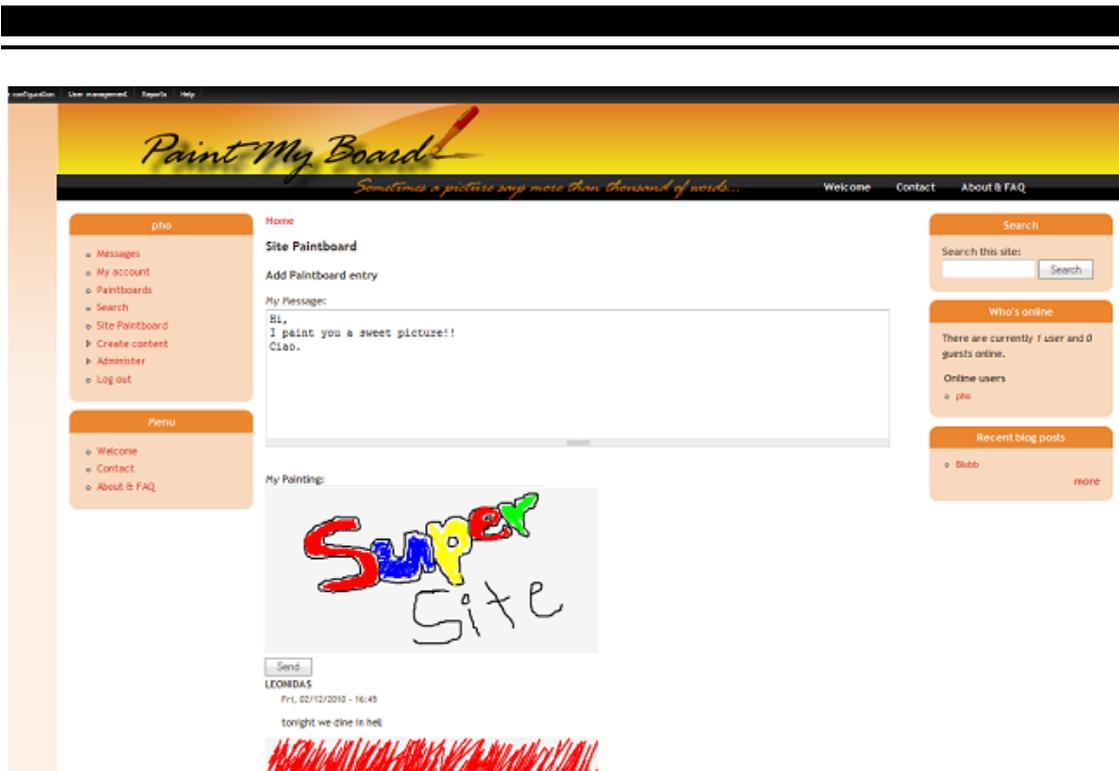


Figure 6.6.: Screenshot of the message board application using pen input.

Report

Pen Interaction Resource: To utilize an Anoto pen as input resource with MundoMonkey, a suitable interaction resource must be implemented. The interaction resource is a MundoCore service wrapping the low-level communication with the pen and emitting MundoCore events. For the experiment, the Nokia SU-1B pen was used. It connects to a computer via Bluetooth. The interaction resource MundoCore service sends events of the following format shown in Table 6.1. The implementation makes use of the Letras platform for pen & paper computing [HSSM10].

field	description
x	float — absolute x coordinate in the Anoto pattern space
y	float — absolute y coordinate in the Anoto pattern space
state	enum — PEN_DOWN: pen just put on paper, PEN_MOVE: pen moving on paper, PEN_UP: pen lifted from paper

Table 6.1.: Format of the events generated by the pen interaction resource.



Figure 6.7.: User interacting with a Web application in the browser using a pen.

Interaction Strategy: The interaction strategy developed for the scenario maps interactive regions in the Web page to regions on the Anoto paper. As the paper coordinates used differ from user to user, the strategy cannot be used without tailoring. Automatically figuring out the correct Anoto coordinates to use would require a global infrastructure keeping track of which coordinates are assigned to which user, e.g., as proposed in [WNS07]. In contrast, MundoMonkey interaction strategies are a much more lightweight solution, allowing each organization to handle the problem itself by adapting the interaction strategy.

The strategy purposefully does not work with existing HTML pages, e.g., enabling pen input for all `<input type="image">` fields. Instead it uses a novel tag called `<painter>`. This is to show how interaction strategies can be developed for specific Web sites or groups of Websites. Still, the strategy is independent of any particular Web page and works together with any Web page using the newly defined tag.

Results

Pen input provides for a more natural interaction in many cases [Ste09a]. Figure 6.7 shows a user interacting with the system. Although the example uses the example of a guestbook application, pen input could also be used to great benefit in business applications, where graphical information is exchanged, e.g., in collaborative business process modelling.

MundoMonkey proved to be a lightweight tool to enable this new interaction modality for the immense amount of existing Web applications. In [Ste09a], an even more general

interaction strategy was implemented that draws on the pen as unified interaction resource for physical paper documents and virtual Web documents.

6.1.3 Federated Access Environment Scenario Case Study

An example scenario illustrating the application of MundoMonkey is a Web browser running on an interactive TV set in the living room, as shown in Figure 6.8. Different members of the family want to interact with Web applications on the interactive TV in different ways using different interaction resources, thereby dynamically changing the interactive space around the browser.

First, Sally wants to flick through the family’s photo album. She uses a mid-air pointing device, e.g., a gyro-mouse or WiiRemote to interact with the browser. This is automatically detected by MundoMonkey, which allows Sally to control a mouse cursor and use the buttons to ‘click’ UI elements. When Sally switches to the home heating control Web application, she can use an onscreen keyboard to set the room temperature to the desired values, as no physical keyboard is detected.

Later, John walks into the living room and wants to adjust the temperature settings. He uses the mouse Sally left on the living room table to navigate to the temperature setting he wants to change. As he is left-handed, MundoMonkey switches the mapping of buttons compared to Sally’s setup. Because John is also somewhat short-sighted, the font-size in the UI increases when he is using the application. John is also wearing a headset with an integrated speech recognition service. So, instead of using an onscreen keyboard, John can control the temperature settings with voice commands.

Report

Three interaction strategies were used in the scenario: the multi-cursor pointing strategy already described in Section 5.1, that was adapted so it also works with one single pointing device, the voice interaction strategy described above and the on-screen keyboard strategy. The used interaction resources were the Talking Assistant as in 6.1.1. The pointing interaction resource was implemented as a MundoCore proxy for the Logitech MX Air mouse, which was connected to a PC via Universal Serial Bus (USB). This PC communicated to the machine running the Firefox browser with the interaction strategies using MundoCore.

The browser had the three strategies stored in the local library. However, MundoMonkey was extended to read in the user preferences from a special *user service*. Thereby, the user service provided a rating for each strategy in the local library. It also sends a picture of the user that is used to show which user controls the screen, as can be seen in the lower right corner in Figure 6.8.



Figure 6.8.: Example for a federated access environment in a living room.

The connection between the user service and MundoMonkey was established on a first-come, first-serve basis, i.e., the first user service in the MundoCore network connects to the browser. Other options that were not implemented, but have been used in other context, would be the usage of location data, which is very well supported by MundoCore [BM05b].

Results

The Web application with which the system was tested was designed to resemble the functionality that one would expect from a home control application. Among other things, it contains a Web page for controlling the heating and airconditioning of different rooms, a task that has also been used in [CCT⁺03]. With MundoMonkey the adaptation of the interaction with the Web application is adapted to a group of devices, thereby supporting federation of devices. This is in contrast to [CCT⁺03], where adaptation is only performed to one target device. Further, *plastic user interfaces* do not provide means to take user preferences into account, if they have not been foreseen by the application.

6.2 MineManager Implementation and Evaluation

The evaluation of the MineManager framework addresses two issues:

-
-
- first, we evaluate whether the MineManager framework leads to easy-to-use and functionally complete meta UIs for post-desktop access environments
 - second, we claimed that the framework is easy to instantiate for any ubiquitous computing middleware.

To evaluate the second aspect, we implemented the framework for the MundoCore ubiquitous computing middleware. The framework was independently applied to a second ubiquitous computing middleware, ReWiRe [VLC08]. We report on the MundoCore implementation in a case study. Being able to implement the framework for two different middlewares suggests that the framework itself can indeed be implemented easily.

To evaluate the former aspect, we designed and implemented an application based on the MineManager framework for the iPhone. We then conducted user tests with this application. We obtained quantitative measurements to judge the usability of the resulting interface. To distinguish between effects that can be attributed to the concrete implementation and the effects that can be attributed to the *concepts* provided by the framework, we obtained further qualitative data.

The rest of this section is organised as follows. First we report experiences from implementing the framework in the MundoCore middleware. Then, we describe the design and implementation of the MineManager iPhone application and present the quantitative and qualitative results of our user experiments. We conclude with a short summary of our findings regarding the MineManager framework.

6.2.1 Implementation of the MineManager Framework for a Ubiquitous Computing Middleware

MundoCore [AKM07] is a flexible communication middleware that supports a mix of communication styles (publish-subscribe, distributed object computing, and streaming) and which runs on different hard- and software platforms (C++, Java, x86, ARM). It has been specifically designed to support ubiquitous computing applications.

The basic abstraction MundoCore provides for a node in a ubiquitous computing environment is the *MundoCore service*. A *MundoCore service* is a process that communicates with other Mundo services by exchanging MundoCore messages. To implement the MineManager framework, we provided base classes for the four entities in the MineManager frameworks that extend the *MundoCore service* class. The user is represented by a singleton instance of the user service, which is called a minimal (digital) entity Minimal Entity (ME). This service can be executed on the personal device of the user.

This requires that all interaction devices are wrapped with a MundoCore service acting as a proxy, following the same approach as [BMRB07]. We implemented such services for standard input devices, i.e., mouse and keyboard, as well as for some advanced devices, i.e.,

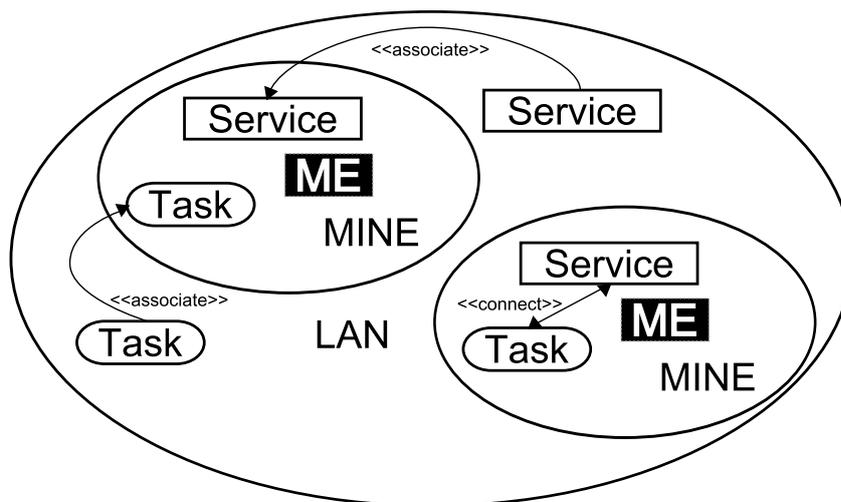


Figure 6.9.: In Mundo a pervasive environment is dynamically constructed by end-users. Users can *associate* resources into the MINE.

voice recognition and the Wii Remote. As output devices a Scalable Vector Graphic (SVG) canvas is available and a text-to-speech engine.

Only a dummy implementation of the tasks class exists directly in MundoCore; however, the Firefox Extension developed for MundoMonkey (described in Section 4.2) can be used to connect web pages from a real browser.

On top of the basic communication layer MundoCore supports a service discovery mechanism. The ME service starts a discovery querying for all available interaction services in the environment. Furthermore, MundoCore provides a special field, called *zone*, which can be used to restrict the visibility of services, i.e., services are only found by discovery, if their zone matches the zone specified in the query. This is used to implement the *association* operation. Instead of a *URL*, MundoCore uses a *GUID* to identify a service.

We defined two zones representing publicly available MundoCore services (zone *LAN*) and associated MundoCore services (zone *MINE* - Mundo Integrated Network Environment) as shown in Figure 6.9. MundoCore provides events whenever a MundoCore service, which can be an interaction service or a task, becomes available or unavailable in the *LAN* zone. These events are used to generate S, T_g^+ or S, T_g^- events. In the same manner, S, T^+ or S, T^- events are generated when a service enters or leaves the *MINE* zone. The interface for task and service contain a method for association, which just changes the zone of the MundoCore service.



Figure 6.10.: MineExplorer on the iPhone

6.2.2 User Experiments

We implemented an application based on the MineManager framework for the iPhone, called MineExplorer (see Figure 6.10). To evaluate the MineExplorer application, we conducted user tests. We let users perform a series of tasks with the prototype and recorded measures such as number of errors and time on task. We also used the System Usability Scale (SUS) [Bro96] questionnaire to estimate the usability of the prototype. Further, we asked a number of more open questions and observed participants in solving the task, trying to evaluate whether the concepts of the MineManager framework are understood by users.

Method

Participants

We recruited 20 participants for our user study. All participants were students or employees of the TU Darmstadt. All participants stated that they use computers on a daily basis, 17 of the participants were studying or have studied computer science, 3 studied other subjects. The age of participants ranged from 25 to 40 years. The sample included 19 males and one female. In summary, the sample represents users that have a lot of experience using computers.

Imagine you are in a public building and as always you carry your iPhone with you. Now, you receive an e-mail you want to reply to. As the e-mail is somewhat lengthy and typing with the keyboard built into the iPhone is somewhat difficult, you want to use a larger screen and a normal keyboard. Luckily, there is a larger screen and a keyboard publicly available. Your task is now to associate the keyboard and the screen and to connect these two devices to the e-mail application.

start configuration		
<i>Available</i>	<i>Associated</i>	<i>Connected</i>
(Keyboard) (Display 1)	(E-Mail Application)	-
goal configuration		
<i>Available</i>	<i>Associated</i>	<i>Connected</i>
-	(E-Mail Application) (Keyboard) (Display 1)	(E-Mail Application , Keyboard) (E-Mail Application , Display 1)

Figure 6.11.: One task from the user test.

Apparatus

One part of the study consisted of tasks the participant had to perform. Each task required the participant to turn a start configuration of the access environment into a goal configuration. A sample task is shown in Figure 6.11; the complete list of tasks can be found in Appendix A.

The tasks were performed by the participants using the MineExplorer running on an iPhone device. The post-desktop access environment was simulated by the experimenter on a laptop computer running a Java program. This program was also used to log the user's actions in the prototype, e.g., to compute the time on task.

Furthermore, participants had to answer a questionnaire. The items in the questionnaire comprised the SUS [Bro96] questions, allowing us to obtain a measure of usability. The questionnaire also contained items aiming at the participant's understanding of the underlying concepts. The questionnaire can also be found in Appendix A.

Procedure

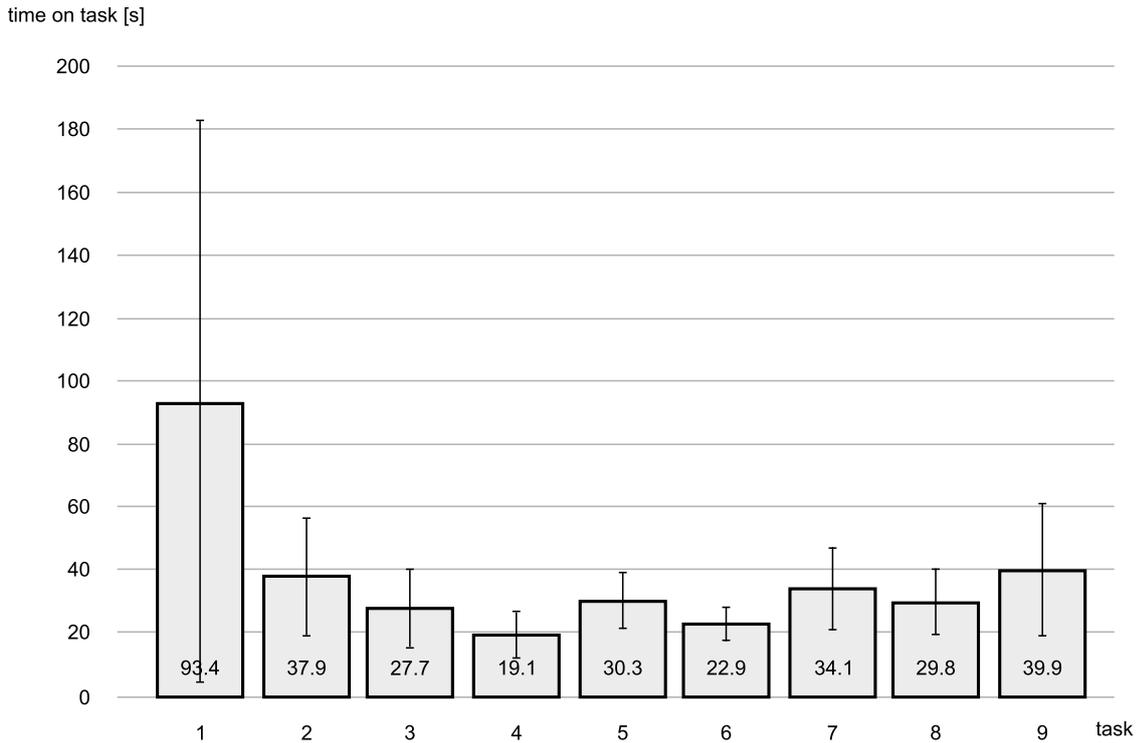


Figure 6.12.: Time on task for the different tasks from the user test.

To begin with, the participants were briefly introduced to the purpose of the study. Thereby, the experimenter introduced the futuristic setting, i.e., participants should imagine input and output devices were publicly available, e.g., provided by the university.

Then they were given the iPhone with the application running and performed the tasks one by one. They were not given the opportunity to test the MineExplorer application first, as we were interested in seeing how fast participants could grasp the GUI. The experimenter surveyed the actions of the user in the monitoring application and also simulated the scenario, i.e., making devices appear and disappear. After completing the tasks, the participants answered the questionnaire. Finally, participants were given the opportunity to express any additional feedback or comments.

Results

We split the presentation of results into two parts. In the first part, we present the results on the usability of the MineExplorer application. In the second part, we present the results from the questions and our observations regarding the suitability of the concepts.

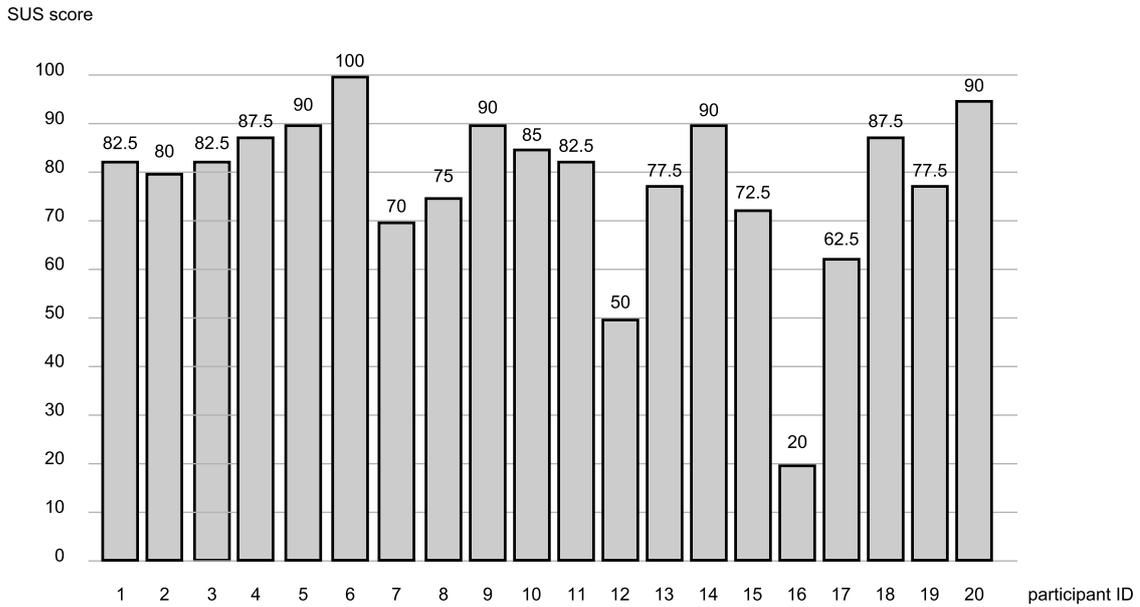


Figure 6.13.: SUS scores for the 20 participants.

Usability

Figure 6.12 shows the time participants took to solve each task. On average, participants took 93.4 seconds to solve the first task. As we did not allow users to try out the system beforehand, this time almost completely reflects the time necessary to get accustomed to the GUI. The standard deviation for the first task was 88.9 seconds. This means there is a relatively large variance across subjects. Some are able to solve the task very quickly, while others need much longer.

However, the times of task 3 ($M = 27.7s$) and task 4 ($M = 19.05$) show that all participants are able to learn how to use the system quickly. Solving task 3 required identical steps as task 1, task 4 could be solved identically to task 2. Users took a third of the time for task 3 compared to task 1, and half of the time for task 4 compared to task 2.

In task 5, the user was confronted with a larger number of services. However, this had only a minor effect on the time on task, participants taking only about 3s longer to solve task 5 compared to task 3, which was identical. The times for task 6 are not much longer than for the other tasks, showing that users are able to solve unforeseen problems with the MineExplorer GUI.

Figure 6.13 shows the SUS score calculated from the questionnaire responses of the participants ($Md > 80$, $M = 77.9$, $SD = 17.75$). Thus, the SUS score of the MineExplorer consistently lies in the top 25%.

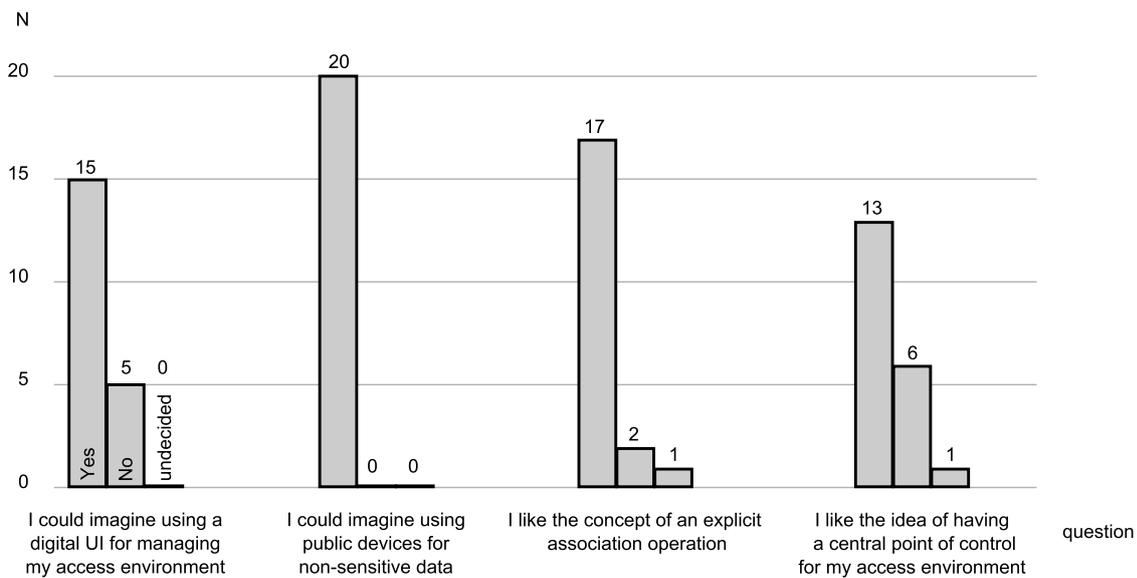


Figure 6.14.: Questions regarding the intelligibility of the underlying concepts.

Concepts

In addition to the usability analysis, participants were asked questions about the MineManager concepts. The majority ($N = 15$) of participants could imagine using a digital UI for managing their access environment. All participants stated they would use publicly available devices for interaction, in case only non-sensitive data is involved. However, all participants refused to use public devices for sensitive transactions, e.g., performing online banking.

In the answers to the questionnaire 17 participants stated they liked the concept of an explicit *association* operation; only 2 said that they do not think this operation is important, as use implies trust.

Finally, 13 participants stated they liked the idea of having a central point of control for their access environment. However, 6 participants had problems with the idea of having a digital UI as the single point of control.

Discussion

These results show that the concepts in the MineManager framework indeed lead to usable UIs. The separation into available and associated services and devices was accepted by users. Making this distinction explicit, e.g., by using two separate list views as in MineExplorer, makes it easy for users to keep track of their personal access environment.

Establishing connections between interaction services and applications by using a connection operation also made sense to users. However, the implementation of the MineManager concepts we used in the MineExplorer was criticized for being unintuitive: some users tried to connect interaction services and tasks using a drag&drop gesture, others first selected multiple interaction services and wanted to connect them all at once. This led to the introduction of the $Present(T, S+)$ operation into the model, which makes it clear that such an operation should be supported as an atomic operation by a GUI. Once users understood how connections were performed in the MineExplorer GUI, they could easily perform the tasks in the experiment. Some users even reported that they enjoyed performing these tasks with the iPhone, which highlights the importance of the *personal* device.

A serious drawback in the implementation of the MineExplorer interface was the representation of the active connections. This became apparent in task 9, where users mostly dismissed the message generated by MineManager in response to the S_c- event. For these reasons, users asked for an overview of all connections. In afterthought, such an overview should have been included in MineExplorer from the beginning, and could have been easily implemented by following the *Present* operations.

An important point limitation of the current MineExplorer design is the number of tasks and services that can be comfortably managed with the GUI. Therefore, we explicitly asked the users how many items they thought they would be able to handle with the prototype. On average users answered that they could handle 8 tasks and devices. This should be enough for handling the devices in most post-desktop access environments. However, if one thinks of the number of tasks that the user might want to manage, this may be actually too little.

6.3 Evaluation of MundoProxy

The evaluation of the MundoProxy was done with regard to two main claims

- Can MundoProxy be integrated into an infrastructure for SOA-backed Web applications without causing additional overhead for the developer, and
- can MundoProxy reduce the latency perceived by the user without compromising battery runtime in mobile access environments

To answer the first question, we integrated the MundoProxy into “AjaxWeaver” an enterprise SOA system and evaluated the effects of the changes for the developer. This was the case. We then went on and used this system to estimate the effects on latency. These estimations suggest that caching and prefetching indeed reduce the latency perceived by the user. To quantify these improvements, we conducted two further experiments.

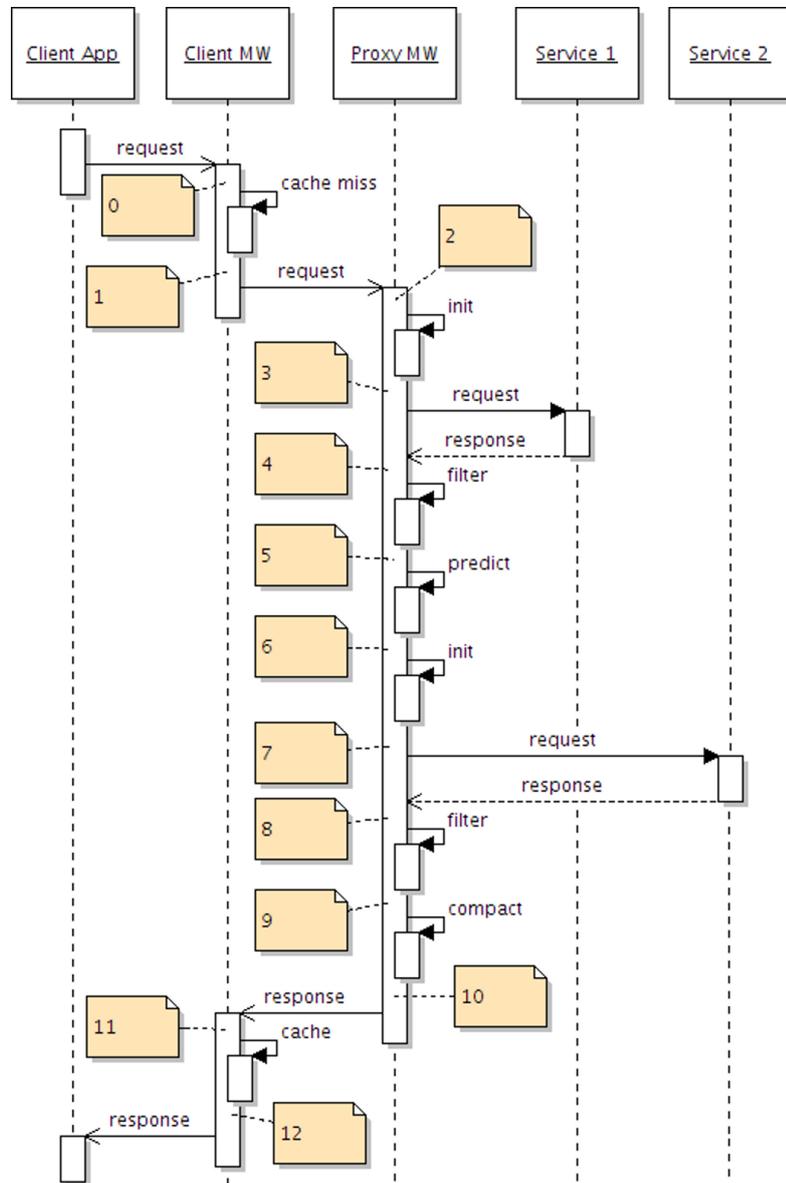


Figure 6.15.: This diagram shows how a request is handled at the MundoProxy and at which checkpoints we took time measurements.

6.3.1 Integration into an Enterprise SOA

To show the practical value of the MundoProxy, we implemented it on top of the existing “AjaxWeaver” framework in cooperation with SAP. “AjaxWeaver” is an end-to-end infrastructure for composing and consuming mobile AJAX applications that are connected to SOAP-based Web services [HWDB08]. In addition to compatibility with many different AJAX-enabled browsers, “AjaxWeaver” provides client runtime and middleware components to facilitate SOA consumption on mobile device browsers. Thus, post-desktop access environments are supported by means of mobile Internet-enabled devices. “AjaxWeaver” provides simple methods to reduce SOAP messages and to mitigate the network latency via increased client-side processing.

The main parts of the “AjaxWeaver” platform are the client side library, the proxy, and the actual applications, described as XML documents that are interpreted at runtime by the client and the proxy. The client part of the platform runs on all major mobile browsers (e.g., Google Android, Opera Mobile, Nokia S60 browser, iPhone Safari and Pocket Internet Explorer).

When an “AjaxWeaver” application is accessed, the client library downloads the application description. The description is interpreted and rendered as HTML UI by the JavaScript library inside the browser. The user is now able to use this application, e.g., a CRM system, to make further requests to the enterprise SOA, e.g., to query customer information. All further requests are routed through the library, where the caching is performed. Requests to the proxy are performed using AJAX techniques.

The only change the implementation introduced for the developer of an SOA-backed Web application was a single check-box to mark a service as eligible for prefetching, as discussed in Section 5.3.1.2. This means the first goal, i.e., not introducing additional complexity for the developer, was reached.

6.3.2 Quantifying Latency Reduction in AjaxWeaver

We measured the reduction of user-perceived latency from our enhancement in “AjaxWeaver” [GSH⁺09]. Suitable real-world workloads for the evaluation are not available, as Enterprise Information Software (EIS) and corresponding access logs are mostly treated as confidential data by any company. Therefore, we evaluated our enhancements in an artificial scenario based on services provided by the *Enterprise Service Workplace* in the *SAP Community Network*. After a free user registration, these services are publicly available for testing purposes. For this evaluation, three of the provided Enterprise Resource Planning (ERP) services were used. The first one is a customer query, which was used to implement a keyword search over all customers. It returns customer names along with their IDs. The second service returns customer data and contact information

for the specified customer ID, while the third service returns a list of sales arrangements for a specified customer ID. For the evaluation, we used these services in the following way, which resembles a typical use case:

1. Search for the term “chemical.”
2. View the details of one specified customer and close.
3. View the details of another specified customer.
4. View her sales arrangements and close.
5. Search for the term “espresso.”
6. Search for the term “turner.”
7. View the details of one specified customer.
8. View her sales arrangements and close.
9. Search for the term “hollywood.”
10. View the details of a random customer in the list.
11. View her sales arrangements and close.

During each request we recorded timestamps at the 12 checkpoints shown in Figure 6.15. Using a fixed scenario does not allow us to evaluate the performance of the sequence prediction algorithm. Thus, the effect of our improvements will be different for realistic scenarios. However, the evaluation shows that the overhead of prefetching is so marginal, that our approach should be beneficial even with very modest success rates of the sequence prediction algorithm.

MundoProxy Implementation Used

The existing framework prevented parallelization of the prefetching as described in Section 5.3. Instead, prefetching had to be performed in the thread which answered the client requests, thereby using the **Simplified algorithm for serving client requests**. This algorithm performs predictions in the same thread in which the original request is served. This way, only the latency caused by setting up the mobile network connection is avoided.

Every request to the enterprise SOA is processed as shown in Figure 6.15. In the following description, the timepoints T_i refer to the time at the corresponding checkpoint i in Figure 6.15, the intervals t_i refer to the timespan $T_i - T_{i-1}$.

First, the client application forwards the request to the JavaScript library on the client (T_0). Next, the library decides whether this request can be fulfilled from the client-side cache (T_1). If so, it proceeds at T_{12} by returning the cached result to the application, where the UI is updated. If the request cannot be answered from the local cache, the request is forwarded to the proxy (t_2).

Algorithm: Simplified algorithm for serving client requests.

Input: client request and client cache hits q, C

```
if  $q \in proxy-cache$  then
|  $r = proxy-cache[q]$ ;
else
|  $r = perform-request(q)$ ;
end
 $p = predict(h)$ 
 $send(r,p)$ ;
 $h = h \circ (q,r)$ ;
```

The proxy is implemented as follows. The request from the client is received at T_2 . Default parameters left out by the client to save network bandwidth are substituted, and a correct URL for the target service is generated during the `init` step. After this, the request is forwarded to the target service at T_3 . We assume this service can be reached via the fast corporate LAN. The service's response arrives at T_4 . It is then filtered at the proxy. In this step, e.g., data fields that are not used at the client are filtered out, again to save bandwidth. This step finishes at T_5 .

With prefetching enabled, the prefetching algorithm is used to determine whether another service request should be automatically executed in advance. If this is the case, another cycle of request initialization, server request and result filtering is performed between T_5 and T_9 . Without prefetching, the steps between T_5 and T_9 are skipped.

Finally, the reply to the client is composed. This reply is available at T_{10} . It consists of the response to the original request and, if prefetching was used, piggybacks the data for the prefetched requests. A condensed text format is used for this reply instead of XML/SOAP, again to save network bandwidth. After generating the reply at the proxy, it is transferred to the client. Finally, the JavaScript library at the client unpacks the results, caches them locally so that they are available for future requests, and updates the application.

Method

An evaluation of the entire system for each interesting combination of clients, networks and services proved to be very complex. Therefore the system evaluation was split up into the following parts:

- Measurement of the middleware overhead caused by the proxy (t_p)
- Measurement of the middleware overhead caused by the client (t_c)
- Measurement of the overhead caused by the backend services (t_b)
- Estimation of the network delay (t_n)

#	w/o prefetching, time measured in ms						
	t_3	t_5	t_6	t_7	t_9	t_{10}	t_p
1	24.46	20.63	0	0	0	1.21	46.3
2	40.29	41.51	0	0	0	0.25	82.05
3	19.88	40.47	0	0	0	0.25	60.6
4	25.07	20.44	0	0	0	0.34	45.85
5	11.25	10.83	0	0	0	0.27	22.35
6	26.92	29.52	0	0	0	1.95	58.39
7	19.87	38.71	0	0	0	0.23	58.81
8	25.59	19.37	0	0	0	0.36	45.32
9	11.44	12.36	0	0	0	0.5	24.3
10	36.2	47.84	0	0	0	0.28	84.32
11	12.23	1.48	0	0	0	0.17	13.88

Table 6.2.: Proxy overhead measured w/o prefetching.

- Calculation of the total delay experienced by the user and speedup caused by the enhancements

Proxy Overhead

For this part of the evaluation the proxy server was run on a MacBook with an Intel Core2Duo 2,4 GHz processor and 2 GB of RAM. The backend services were substituted by dummy services running on the same machine. We ran a script in the Firefox 3.0.5 browser on the same machine that automatically went through the 11 steps of the scenario 100 times. Thus, all communication was performed via the loopback network device. The average of the measured times can be found in Tables 6.3 and 6.2. The first column denotes the step in the scenario from above. All times are average times from 100 evaluations given in milliseconds. Table 6.2 contains the values with prefetching disabled. Note the zeros in columns four, five and six. These steps are skipped without prefetching. Table 6.3 contains the corresponding values with prefetching enabled. Note the zero rows four and six, and the near-zero row eight. In these cases, the request has been predicted in the previous step and was prefetched. Accordingly, the client does not request it from the proxy, but retrieves it from the cache. Row eight was not always predicted as a prefetch, as the online learning algorithm had to adapt to the scenario.

Client Overhead

To measure the effect of our prefetching enhancements at the client, we used an Apple iPod Touch with firmware 2.2. We accessed the proxy via WLAN and ran the client part of the middleware in the built-in Safari browser. We again ran a test script, performing 100 runs

with prefetching, time measured in ms							
#	t_3	t_5	t_6	t_7	t_9	t_{10}	t_p
1	25.69	19.57	0.79	18.95	29.03	0.97	95
2	0	0	0	0	0	0	0
3	16.19	34.2	0.43	11.57	16.07	0.39	78.85
4	0	0	0	0	0	0	0
5	12.32	10.45	0.72	11.41	17.99	1.59	54.48
6	0	0	0	0	0	0	0
7	18.05	34.03	0.43	11.25	12.54	0.43	76.73
8	0.57	0.11	0.01	0	0	0	0.69
9	11.92	15.04	0.56	0	0	1.2	28.72
10	39.65	48.71	1.07	0	0	0.28	89.71
11	9.88	1.33	0.43	0	0	0.08	11.72

Table 6.3.: Proxy overhead measured with prefetching.

through the scenario with and without prefetching enabled. The results for the measured client overhead t_1 and t_{12} with and without prefetching are shown in Table 6.4. The first column denotes the step in the scenario from above. All times are average times from 100 evaluations given in milliseconds. The left half of the table gives the values with prefetching disabled. The right half of the table give the corresponding times with prefetching enabled. Note the relatively large difference in lines five and six. With prefetching, the request no. 6 is already sent and filled in the cache at step no. 5. Consequently, the time needed for caching in step six is much shorter with prefetching enabled, as no response from the server is parsed.

As one can see, only a very small fraction of overall *user-perceived latency* is due to the client library. This is the case although even the baseline performs client-side caching, thus emphasizing the efficiency gain provided by simple client-side caching. The additional overhead caused by our prefetching enhancements at the client are nearly non-measurable.

Backend Services

The timing of the backend services of course depends on the nature of the corporate LAN and the actual services used. For the services used in our scenario we measured 16 *ms* as the average time for a single service request. While the overhead without t_b will be used to calculate the speedup achieved in the middleware, the total, including t_b serves as a hint to what the speedup will be for the user in an application scenario. Note that if a prefetch occurred, t_b is $2 * 16$ *ms*, as the time for accessing a backend service has to be added twice, once for the original request and a second time for the prefetched request.

#	with prefetching time measured in ms		w/o prefetching time measured in ms	
	t_1	t_{12}	t_1	t_{12}
1	1.08	18.07	1.09	18.19
2	1.15	2.70	1.43	0.16
3	1.21	2.99	1.28	5.75
4	1.24	2.74	1.46	0.15
5	1.27	2.51	1.62	35.74
6	1.20	20.10	1.44	0.19
7	3.76	2.91	1.24	6.19
8	1.24	3.39	1.38	0.21
9	1.20	10.76	1.22	10.58
10	1.24	2.73	1.19	2.71
11	1.19	1.43	1.29	1.55

Table 6.4.: Client overhead for caching measured with and w/o prefetching.

Network Delay

To simulate different wireless networks, the network characteristics shown in Table 5.3 were used to calculate different response times based on the results of the locally measured values. To do this, we measured the amount of data that had to be transferred between the proxy and the client for each of the 11 requests from the scenario. Given the network delay for an empty packet t_{d0} , the upstream data rate r_{up} and the downstream data rate r_{down} , the overall network delay for a request with size s_{req} and a reply with size s_{repl} is then calculated using the following equation:

$$t_n = t_{d0} + s_{req} \frac{1}{r_{up}} + s_{repl} \frac{1}{r_{down}}$$

Results

To determine the average *user-perceived latency* caused by the middleware for the various network settings, we added the values measured for the client and the proxy overhead to the network delay, computed from the network characteristics. We thus were able to compute the values for four typical mobile network conditions. To calculate the total *user-perceived latency*, we further added the time spent in the backend services. Figure 6.16 shows how the latency decomposes for requests three and four from the scenario for the UMTS network condition. The figure highlights the difference between non-prefetching behavior - roughly equal latency for both requests - and prefetching behavior - slightly increased latency for request three where the prefetching is performed and no noticeable latency for request four that can be fulfilled from the client cache.

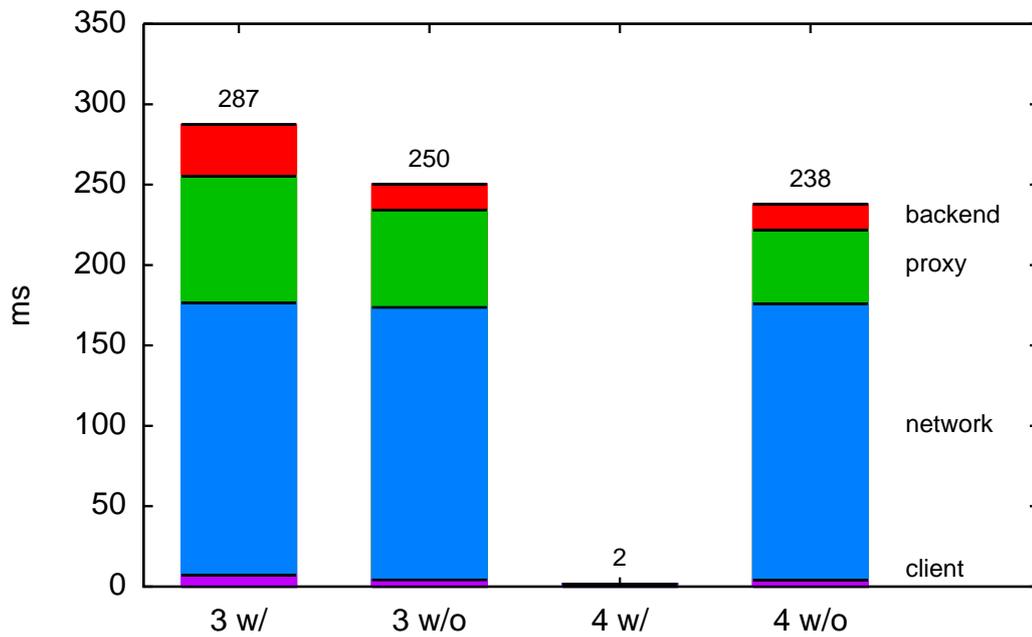


Figure 6.16.: Time comparison for two requests with and without prefetching. Depicted are the UMTS average times for requests no. 3 and 4 of the scenario.

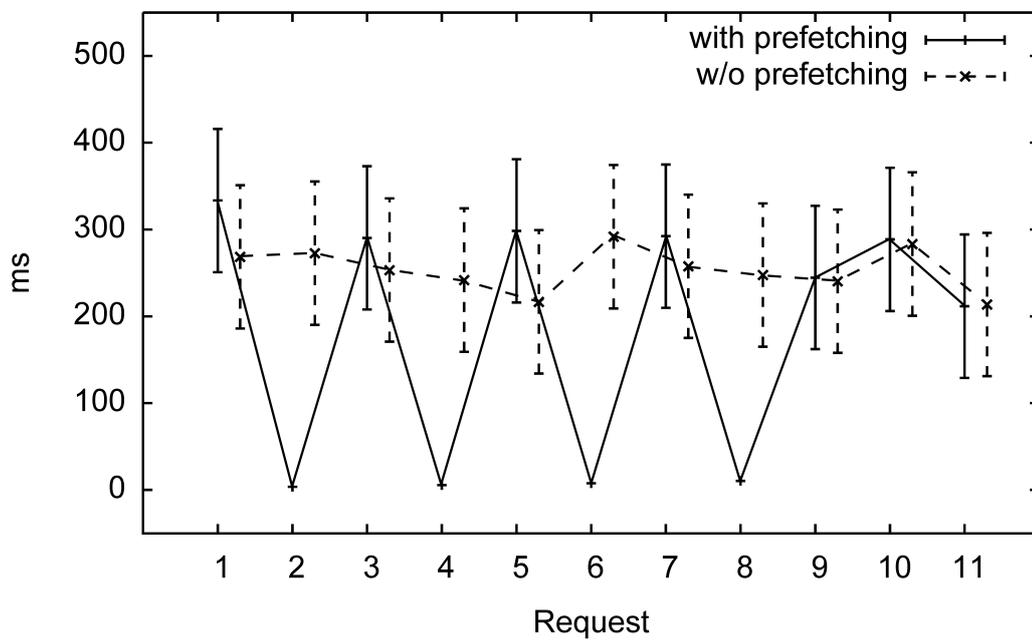


Figure 6.17.: Comparison of prefetching and baseline behavior using UMTS.

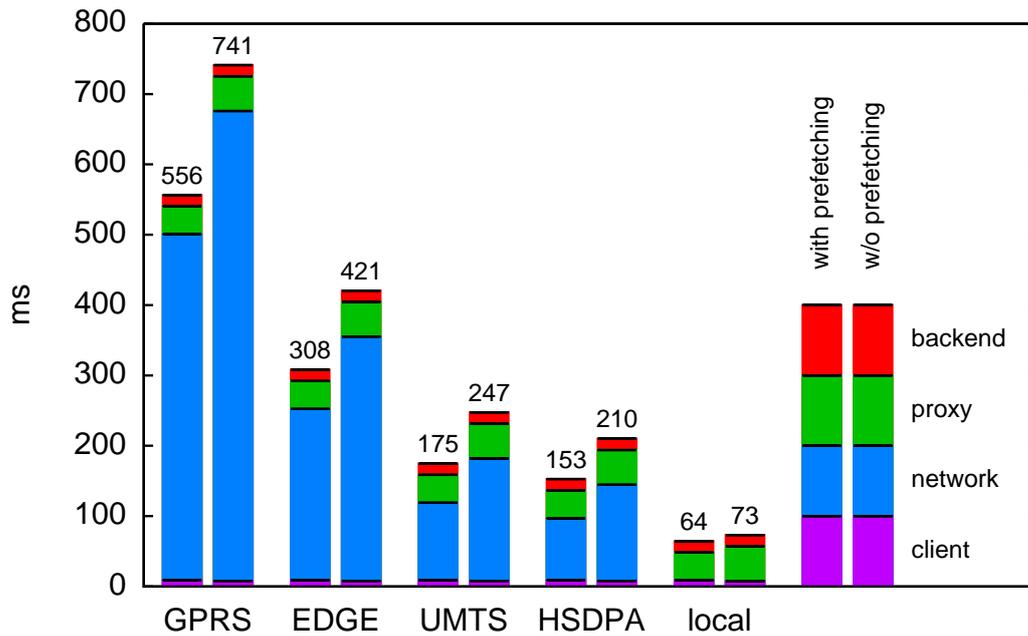


Figure 6.18.: Comparison of user-perceived latency in different networks, averaged across all 11 requests of the scenario.

Figure 6.17 contrasts the results of prefetching and non-prefetching behavior over the whole scenario for the UMTS network condition. Prefetching results in a zig-zag behavior with a lower mean latency than in the non-prefetching condition. The lower end of the bars indicates the minimum latency, the tick in the middle the mean latency, and the upper end the maximum latency.

Figure 6.18 shows the results averaged over all 11 requests from the scenario with all considered network profiles. For the GPRS network profile with average latency, we found an overall speedup of 26%. For the HSDPA profile with average latency the speedup was 29% and 31%, if we take only the middleware into account and neglect the time spent in the back-end services.

To estimate the effect of our enhancements on energy consumption on the mobile devices, we analyzed the average accumulated network times for the scenario. Shorter network times directly correspond to power savings. As shown in Figure 6.19, the network time is indeed shorter with prefetching enabled, thus prefetching could help to save battery power.

Discussion

These initial results suggest that prefetching at the proxy for a mobile SOA middleware can indeed improve the user experience by decreasing user-perceived latency and thereby

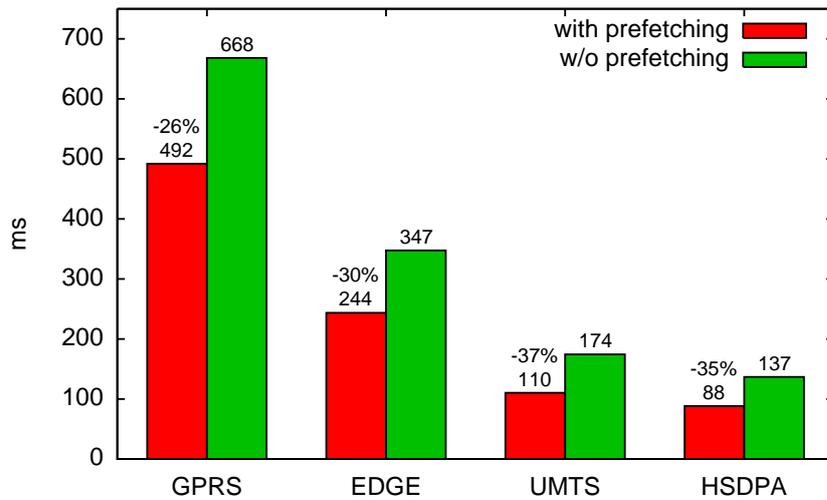


Figure 6.19.: Comparison of accumulated average network times. Reduction of network time directly corresponds with battery power saving.

increase the usability of SOA-backed applications in post-desktop environments. Even more important, our approach lends itself to easy integration in commercial grade systems.

However, the initial experiments described above leave several questions open. The biggest drawback of the experiment is that no realistic workload was available, which makes it impossible to evaluate the performance of the FxL algorithm. To rectify this, we performed additional experiments with a more realistic workload described in section 6.3.3.

Furthermore, several of the caching and prefetching enhancements could not be implemented in “AjaxWeaver,” e.g., the parallelization of prefetching. Also, several important parameters, such as the network latency and the service execution time, were not measured directly but were only estimated. A more complete experimental setup was therefore used in another experiment, which is described in section 6.3.4.

6.3.3 Simulation with Realistic Workload

One major limitation of the estimates generated with the MundoProxy implementation within AjaxWeaver was the lack of realistic service call traces. The benefits of caching and prefetching Web service calls in mobile applications can only be evaluated with realistic workloads [DPSG07]. Unfortunately, it proved difficult to get real Web service call traces from business applications that are in production use. The underlying processes of an enterprise are considered to be a key to its competitive advantage, and the processes are thus kept secret. In addition, traces carry a lot of personal information about the involved entities. Even after anonymization, it is still possible to obtain sensitive statistical infor-

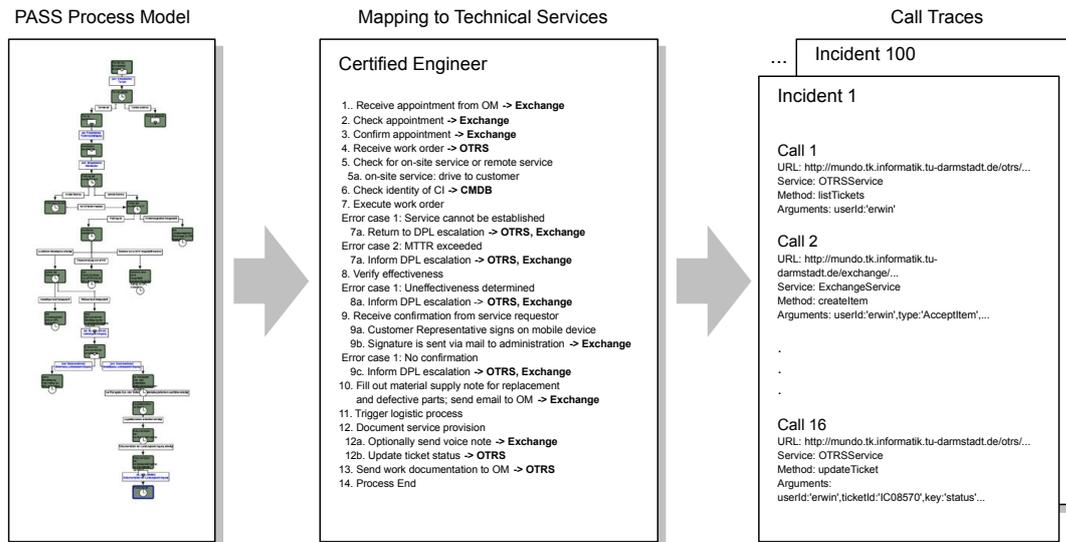


Figure 6.20.: Steps to derive call traces starting from the ITIL standard process.

mation from the data. When considering an IT scenario, e.g., it can easily be determined how often the product of a vendor A fails in comparison to a product by vendor B, or which user performs the most maloperations.

To overcome this problem, we used data that was available to us to create our own traces. The Computer Operations Group together with the Telecooperation Group at TU Darmstadt have had expertise in the area of IT service and maintenance for several years. To obtain realistic call traces, we defined a business processes to handle IT service and maintenance incidents, and then played this process through with a number of selected real incidents.

ITIL is the worldwide de-facto standard for service management and contains broad and publicly available professional documentation on how to plan, deliver and support IT service features [iti10]. The procedure for generating the traces is shown in Figure 6.20. We started from the ITIL incident management process, which is part of the IT service management standard ISO 20000. A version of this process modeled in the process description language PASS (Parallel Activities Specification Scheme) was obtained from the company jCOM1 [jCO10]. Next, the process was refined to incorporate the concrete Web service invocations needed in each process step. This approach is basically the opposite of process mining, where an unknown process is discovered from a log of events, based on the ITIL processes, as described in [FdS08].

To generate concrete call traces, we selected 100 incidents we have handled in the past and played them through based on the ITIL process on a prototype client. We have analyzed 1714 past incidents, documented in 6751 messages. 24% of these incidents were caused by

maloperation. Such incidents do not trigger work orders for service personnel. Instead, the users are just informed about the correct use of the system. 60% of the incidents could be resolved using remote administration tools, and 16% required on-site service. We were especially interested in the on-site service use-case. Here, the certified engineer is working at the customer's site and participates in the business process using his mobile device. We have selected 100 such on-site incidents for our evaluation.

Method

For the first experiment, we used the Web service traces generated by five service technicians in the ITIL scenario. Each service technician generated traces of Web service calls for 20 on-site incidents. We went through these Web service calls of the ITIL traces in the same order a mobile application would, and counted the number of *client cache* hits h . We tested several different conditions for adding data to the *client cache* and emptying the *client cache*. We also counted the number of prefetched requests p sent to the client, as this corresponds to more network traffic and longer online times, which reduces battery runtime of the mobile device. As we just analyzed *hit rates*, it was not necessary to invoke the real Web services in the back-end. Instead, requests were prefetched immediately to the *client cache*. We obtained values for h and p for the following five test cases.

- i) **prefetchable:** Several Web service calls in the ITIL traces must not be prefetched, because they change state on the server, e.g., accepting a ticket. Such calls are often predictable, but they must be triggered by the user and cannot be called in advance. Whether a function is prefetchable or not must be annotated in the service interface description. An upper bound for the number of cache hits is thus given by putting all *prefetchable* Web service calls into the cache from the beginning.
- ii) **persistent-cache:** In this case, all service calls were simply cached, and the cache was never emptied. This results in a maximum number of cache hits, because our prediction algorithm only prefetches requests that have been observed before, and these would also be in the cache. However, such a setting is highly impractical as it would use stale data a lot of the time the cache is used.
- iii) **cache-only:** Here, the calls were simply cached, and at the beginning of each incident the cache was emptied to avoid stale data. Hence, the cache only helps if the same call is made more than once within the same incident.
- iv) **predict-single:** In this case, the cache was emptied at the beginning of each incident. Every time a request was made, it was added to the cache. Hence, repeating Web service requests during the same incident were handled from the cache. Additionally, every time a Web service call was sent to the server (as opposed to obtaining it from the cache), the FxL algorithm was used to predict a single other Web service call. If any call was predicted with a confidence greater than a threshold, this predicted call was then also added to the cache.

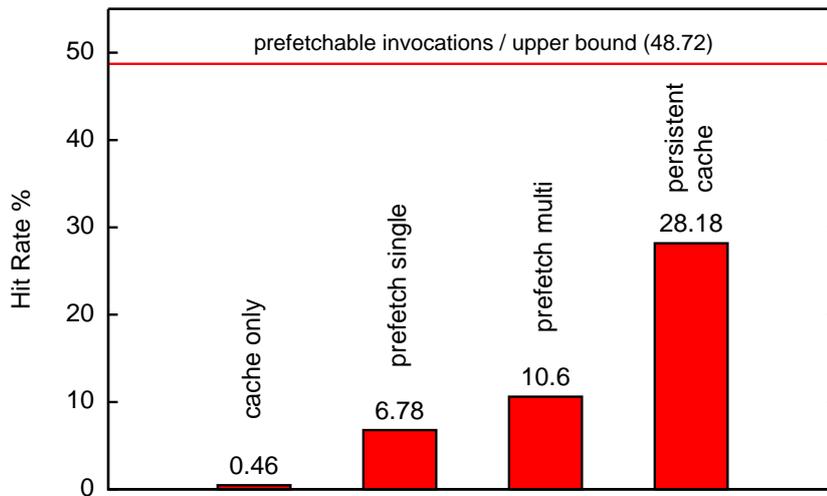


Figure 6.21.: Comparison of prediction with pure caching and the impractical case of a persistent cache. The upper bound is given by the amount of prefetchable invocations.

- v) **predict-multi:** This case works similar to *predict-single*, except that multiple predictions per request were used. All Web service calls that were predicted by FxL with a confidence over a threshold t were added to the cache. A higher threshold t results in lower values for p and h .

Results

The results are shown in Figures 6.21 and 6.22. Figure 6.21 displays the average hit rates using all 100 incidents for the five cases described above. Figure 6.22 shows the average prediction success rates, relative to predictable, for the conditions *predict-single*, *predict-multi*, and *cache-only*. The hit rate is the number of cache hits h divided by the number of Web service calls made during one incident. To generate the graph we averaged the results of all five service technicians, each working on his own subset of 20 incidents. As one can see, there is a somewhat slow start, as the prediction algorithm has to learn about the application first before it can provide good prefetching data.

Figure 6.23 shows the correlation between the threshold t and the hit rate in the *predict-multi* case. It can easily be seen that lowering the value of parameter t increases the number of cache hits at the cost of more communication overhead, because more requests are prefetched.

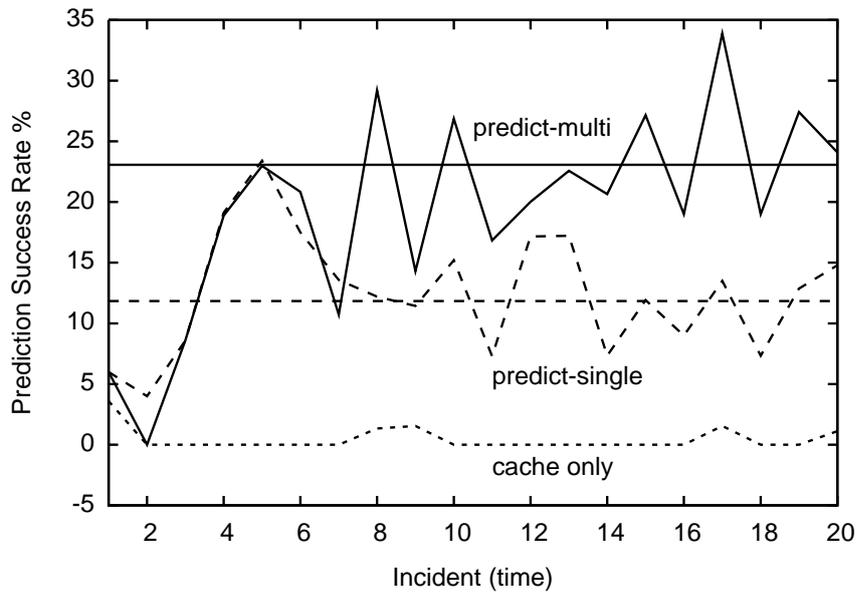


Figure 6.22.: The average prediction success rate “converges” after processing 5–10 incidents for both algorithms.

Discussion

The simulations with the realistic workload show that the FxL algorithm works reasonably well with realistic examples of service traces. These simulations however leave out the effects of the network, e.g., fluctuations in the bandwidth may cover the effects of the cache hits. Furthermore, the effect on the online time which is directly related to the battery runtime cannot be obtained by simulation. To measure these values, we resorted to a more realistic scenario with a real device.

6.3.4 Experimental Evaluation

Method

To evaluate the effects on a real mobile device we selected one of the service technician traces and ran it with the mobile browser. For our measurements, we used the first three incidents of one service engineer as training data, i.e., simply stored the respective requests in the prediction engine. We then played through the fourth incident of this service engineer with the real client on the mobile device. The incident we used for testing consisted of 16 individual requests to the back-end. We measured the latency l perceived by the user and the total online time o by instrumenting the JavaScript code of the client. The latency was

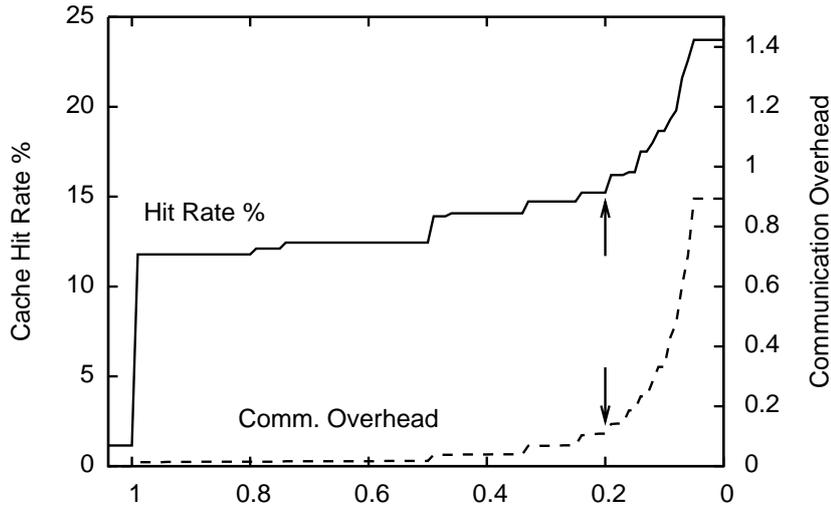


Figure 6.23.: The threshold parameter controls the FxL prediction algorithm. Lower thresholds lead to higher hit rates, but also add communication overhead. A hit rate of 15.2% is already attainable at 10% overhead.

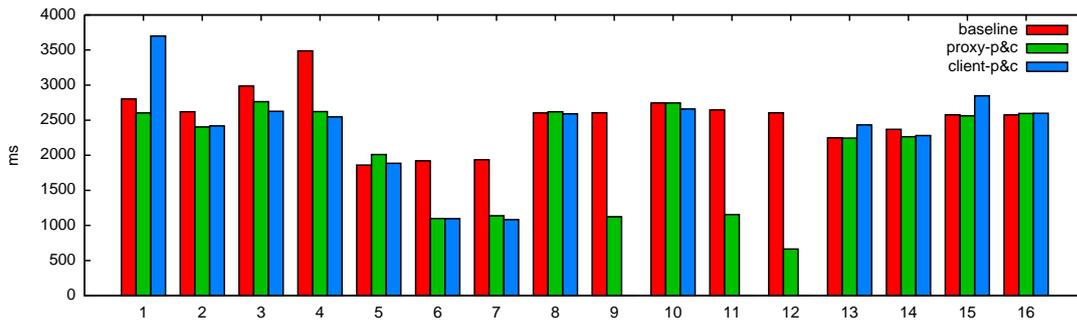


Figure 6.24.: Comparison of user-perceived latency under three different test conditions using an iPhone connected via EDGE.

measured by comparing timestamps just before a Web service request was made until the response was available. The online time was measured from the beginning of the request until the last prefetched response was available. We obtained measurements for l and o in three different test cases.

- i) **baseline:** In this case we did not use probabilistic prefetching at the proxy and no additional data was piggybacked to the proxy responses. As the incident does not contain duplicate requests, no requests could be answered from the *client cache*.
- ii) **proxy prediction & cache:** In this case we used a *proxy cache* which was proactively filled with predicted requests but did not send any piggybacked data to the client.

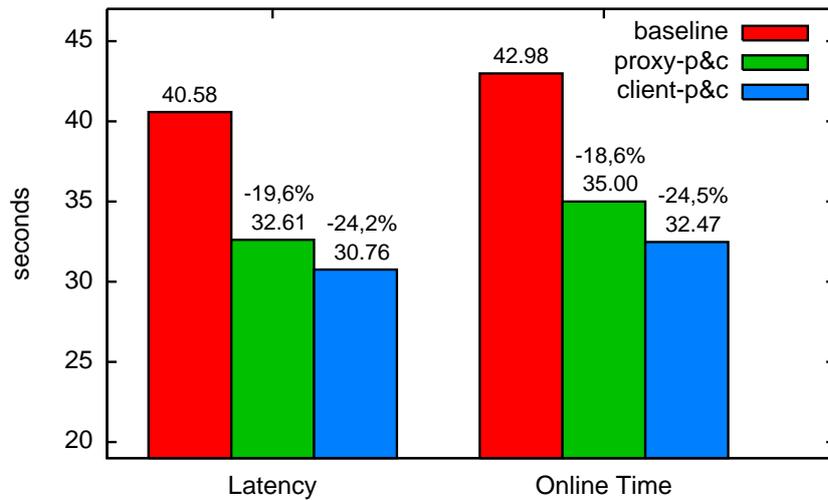


Figure 6.25.: Comparison of total user-perceived latency and online time in the analyzed incident. The bars were scaled to start at 18.97 seconds, because this is the baseline back-end time.

Thus, some requests cause a much lower latency at the client. However, every request causes at least a minimum latency due to the overhead for connection setup.

- iii) **client piggybacking & cache:** In this case we used a *proxy cache* as above but also sent the cache contents piggybacked to the client, where they proactively fill the *client cache*. In this setup the client may skip some server requests completely as they can be answered directly from the *client cache*.

In the cases that used prediction, the threshold was set to $t = 0.2$, as this is a good tradeoff between additional cache hits and prefetching overhead, as can be seen in Figure 6.23.

Apparatus

We used an iPhone (iPhone 3G, OS 3.1.3) as client device, accessing the MundoProxy server via an EDGE connection. MundoProxy was implemented as a Java servlet in Tomcat. As back-end services to support our process we selected three well-known industry-strength products: OneCMDB (Configuration Management Database) [Lok09], the Open Ticket Request System (OTRS) [OTR09], and Microsoft Exchange 2007 [Mic09]. These services ran on a different computer in the local network. Thus, the setup was as shown in Figure 6.26, matching the general setup of SOA-backed Web applications (see Figure 2.5).

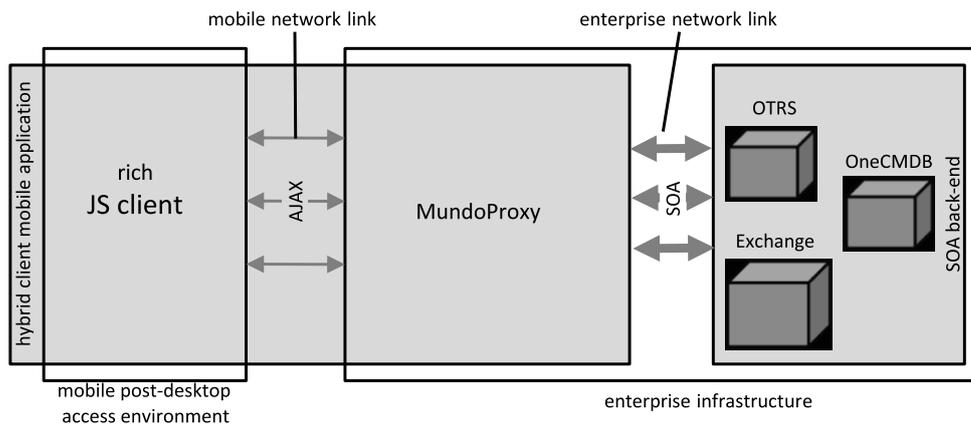


Figure 6.26.: Hybrid client architecture for a mobile process application accessing an SOA back-end. The rich client is implemented in JavaScript.

Results

Figure 6.24 shows a comparison of the user-perceived latency for all 16 requests of the incident in the three cases. Request #6 could be answered from the *proxy cache*, i.e., it was prefetched after one of the preceding requests. It has not been piggybacked to the *client cache*, however, because the call to the Web service in the back-end took so long that the result was not yet available in the *proxy cache* before request 6 was actually made. In contrast, request #9 could be piggybacked to the client. As a result, there is no user-perceived latency for handling this request in the *client piggybacking & cache* condition. Figure 6.25 shows a comparison of the total values for l and o for all requests of the analyzed incident.

The time spent in the SOA back-end for the requests that can be answered from the *proxy cache* accounts for approx. 20% of the total user-perceived latency. Thus, introducing a proactively filled *proxy cache* alone leads to a 20% decrease in user-perceived latency. Using piggybacking for transmitting the *proxy cache* contents to the client allowed for a reduction in latency, due to connection setup of requests piggybacked to the client. This reduced user-perceived latency by a further 4%, yielding a total reduction of user-perceived latency of 24%. Using the proactively filled *proxy cache* reduced the online time of the client by 19%. Here, piggybacking led to a further reduction of online time by 6%, totaling to a 25% reduction of online time.

Discussion

The results of our initial estimation are affirmed by the results of this experiment — it is indeed possible to reduce the latency perceived by the user and the total online time by applying caching and prefetching. However, the effect comes mainly from avoiding the

processing time of the SOA services in the back-end, not the overhead of the mobile network. However, by using piggybacking, which reduces the latency caused by the overhead of the mobile network, a further 4% of latency reduction could be achieved. The effect of piggybacking on online time is slightly larger, causing a reduction of 6%.

During the experiments, the network connection quality was pretty stable. The benefit of additional cache hits would be even greater in circumstances where this is not the case. If a request can be answered from the cache in adverse networking conditions, this may make the difference between a user's using the mobile application and the user's deviating to other practices, e.g., bulk reporting at the end of the workday.

6.4 Chapter Summary

In summary, the three case studies conducted with **MundoMonkey** show that it is indeed possible to implement interaction strategies in an application-independent fashion. Application-independent does not mean application-agnostic. It is possible to extract enough information from the HTML UI description that is available at runtime to provide usable interaction. Thereby, the results from the user study carried out with the voice interaction strategy show that the resulting usability is comparable to the usability that is attainable through more low-level and therefore more complicated tools. Table 6.5 shows which requirements are tested in the respective interaction strategies.

	R2	R3	R4
Voice Input	✓	✓	✗
Pen Input	✗	✓	✓
Federated	✓	✓	✓
On-Screen Keyboard	✗	✓	✓

Table 6.5.: Comparison of the MundoMonkey interaction strategies presented in the case studies to the requirements from Chapter 2

If we compare the features of MundoMonkey to the existing systems for reifying mappings between application UI and access environment, MundoMonkey provides more features than existing systems as shown in Table 6.6.

The implementation of **MineManager** for MundoCore and independently for ReWiRe has shown that the framework is easy to instantiate. Second, the user experiments indicate that the concepts behind MineManager make sense to the end-user. The users were able to manage post-desktop access environments using the MineExplorer application.

System	Structured Output Modification	Input Not Restricted to Mouse & Keyboard
OpenInterface	X	✓
Interactors	✓	X
MundoMonkey	✓	✓

Table 6.6.: Comparison of different approaches for reifying the mapping between interaction services and UI. MundoMonkey provides more features than existing systems from Table 3.2

We have shown how the user-perceived latency for mobile SOA access can be reduced by employing an adaptive prefetching algorithm and caching. Experiments with realistic Web service call traces showed that using a *proxy cache* proactively filled with requests predicted by the FxL algorithm can reduce the user-perceived latency by 20%. Piggybacking additional results into the *client cache* leads to an overall reduction of user-perceived latency of 24%. At the same time, the online time can be reduced by 25%. Reduced online time directly corresponds to energy savings and longer battery runtime of the mobile device (see Table 6.7).

	relative reduction of	
	latency	online time
none	0%	0%
prediction	20%	19%
piggybacking	24%	25%

Table 6.7.: Reduction of user-perceived latency and online times achieved by the different caching and prefetching approaches on realistic call traces.

Using the proposed approach, the decrease in user-perceived latency comes without any additional tuning of the Web services or the front-end application. The probabilistic prediction algorithm employed automatically adapts to the underlying application, thereby removing the burden from the developer to manually optimize for latency. In addition, it also adapts to the usage pattern of individual users. Even if the latency reduction for a particular setup cannot be exactly predicted, our approach should be adopted, as it comes without any drawbacks.



Chapter 7

Conclusion

Web applications are already frequently used in post-desktop environments today. For example, modern smartphones enable access to Web applications in mobile settings. Other Web access systems are customized and tailored towards users with special needs. However, none of the existing Web access systems is flexible enough to cope with arbitrary post-desktop access environments. This thesis presented several enhancements to the Web access system that address this shortcoming. The entirety of these enhancements is the MundoWeb system, as presented in Chapter 4. It comprises solutions to three distinct sub-problems of Web access in post-desktop environments:

- handling in- and output resources
- suitable meta interaction capabilities
- dealing with the user-perceived latency

The solutions are implemented and evaluated in the MundoMonkey, MineManager and the MundoProxy components, respectively.

This chapter concludes the thesis by summarizing its contributions in Section 7.1. Section 7.2 revisits the requirements from Chapter 2 and describes how these are addressed by the MundoWeb system. Finally, Section 7.3 presents some prospects for future work and possible generalizations.

7.1 Thesis Contributions

Analysis Existing research fails to evaluate the principles underlying the Web as access system with respect to their meaning for interaction in post-desktop access environments. This does not do justice to the immense practical success of the Web in post-desktop settings. To be able to compare other approaches to the Web access system, we introduced a novel theoretical framework and derived a new set of requirements. This set of requirements is not only backed by a case study but also validated by comparing it to two other sets of requirements from literature. By applying this framework, we showed that the existing Web access system already has many desirable properties. We also showed in which aspects the Web access system needs to be improved.

MundoWeb The overall concept of MundoWeb addresses the remaining shortcomings the Web has with respect to interaction in post-desktop access environments. MundoWeb is the first approach that addresses all the requirements necessary to support post-desktop access environments. MundoWeb is based on the Web access system and does not adopt a clean-slate approach. This means that MundoWeb can be even used with legacy Web applications.

MundoMonkey One characteristic of post-desktop access environments is the diversity of resources found in them. Existing approaches for handling this wealth of resources are either optimized towards processing of input but then do not provide means for structured output modification; or they are optimized for modifying output, in which case their input handling capabilities are limited to mouse and keyboard. MundoMonkey interaction strategies overcome these limitations as they can flexibly handle input from arbitrary sources, thanks to the connection to the MundoCore middleware, *and* can at the same time perform modifications of the UI, using the rich DOM of Web applications.

Interaction Strategies While working on MundoMonkey, several interaction strategies and interaction resources have been implemented. Some of the implemented strategies, such as the voice strategy proved superior to some widely used existing strategies. Although this has not been further investigated, it is likely that usability improvements to other systems could result from adopting some of the concepts used in that strategy. Other strategies, such as the one for pen interaction, have been refined and reused in further research [SBM09].

MineExplorer In federated post-desktop environments the user has to deal not only with the diversity of resources, but also with the dynamics of the environment. Resources appear and disappear; the accessed applications must be easy to control for the user. MundoWeb provides a single meta UI to the user, allowing him to manage and control the post-desktop access environment. This complements the existing Web browser meta UI for controlling applications. The MineManager framework developed in this thesis is the first to enable the use of a digital meta UI synchronized with physical changes to the access environment.

An instance of this framework, called MineExplorer, has been implemented for the iPhone and the MundoCore middleware. It has been evaluated with user tests that showed the validity of the underlying concepts. The MineManager framework is not restricted to a single middleware implementation but has also been implemented on top of the ReWiRe middleware.

MundoProxy One problem of post-desktop access environments is the high latency perceived by the user. While this problem has been addressed for Web applications before, the MundoProxy is the only solution that is usable in the case of mobile access environments without requiring any changes to the implementation of the back-end services or the Web application. Application independence is achieved by using intelligent prefetching algorithms that predict future user requests based on observed previous requests. The peculiarities of mobile access environments are addressed by the piggybacking mechanisms that enable efficient use of network bandwidth as well as of the energy of the mobile device battery.

Prefetching Evaluation To evaluate the MundoProxy, a realistic workload has been created. This workload is available to other interested researchers. It fills a gap, as it is next to impossible to obtain real call traces from production Web applications as these are kept secret by companies.

7.2 Revisiting the Requirements

MundoWeb, comprising the three components MineExplorer (based on the MineManager framework), MundoMonkey, and the MundoProxy, has been designed to meet the requirements elicited in Chapter 2. Here we will briefly revisit them and explain how they are addressed.

R1 — Reuse of Existing Web Applications MundoWeb has been designed to be compatible with existing Web applications. The voice case study, for example, shows that MundoMonkey is capable of working with the existing wikipedia application. The MundoProxy has equally been tried out with existing applications, from SAP in the evaluations described in Section 6.3.2 as well as with our custom-built ITIL support infrastructure (Section 6.3.3). In both cases no changes to the underlying Web services in the back-end have been performed.

R2 — Federated Access Environments MineExplorer and MundoMonkey work on top of MundoCore, a ubiquitous computing middleware. Access environments can be constructed in an ad-hoc manner, out of individual interaction resources. MundoMonkey can handle the resources in federated access environments, as the case study about federated access environment shows.

R3 — Changing the Input Processing MundoMonkey is capable of handling input from various sources, as illustrated by the case studies. It is also possible to modify the

output on one interaction resource in response to the input received from another interaction resource, as the example of the multi-pointer interaction strategy shows.

R4 — Modification of Output MundoMonkey retains all the output modification features of Web scripting systems. Unlike pure pipeline-based systems, it therefore supports a rich model for output modification at runtime. The example of the on-screen keyboard shows how this feature can be utilized in an interaction strategy.

R5 — Application Independent Reduction of User Perceived Latency The MundoProxy has been evaluated quantitatively in various settings, including mobile access environments, showing that it is capable of handling latency issues arising in SOA-backed Web applications accessed in post-desktop environments.

R6 — Application and Environment Management and Configuration MineExplorer supports the configuration of (federated) access environments. It thereby complements the meta operations provided by the normal Web browser, forming a complete solution for this requirement. The interface of MineExplorer and the concepts underlying it have been approved by users in a user test.

MundoWeb is the first approach to address all requirements for interacting with Web applications in post-desktop environments. State-of-the-art approaches only address a subset (see Table 3.1).

7.3 Outlook

In this section, we look at several ways in which MundoWeb could be extended, discuss some possible generalizations, and suggest further research directions.

7.3.1 Further Challenges in the Topic of the Meta UI

Since the publication of the first paper describing MineManager, the topic of meta UIs for post-desktop access environments has gained a lot of interest, e.g., spawning a workshop at the EICS conference (Enhancing interaction with supplementary Supportive User Interfaces¹). It seems obvious that post-desktop computing will see a meta UI, i.e., the post-desktop equivalent of the task bar and minimize/maximize buttons found in desktop systems. MineManager presents one solution to this challenge, which provides the right concepts to manage post-desktop access environments and the Web applications accessed in them. If one has other needs, the MineManager concepts may be insufficient and need to be augmented.

The interface of the MineExplorer is based on a simple list-based view, which is easy to scan on a mobile device and familiar to the user. For example, MineExplorer looks similar

¹ <http://www.supportiveui.org/>

to the contact list on the iPhone. As our own user experiments show, this might not be ideal, for example when a huge amount of interaction resources needs to be handled. Further research in that direction is necessary.

7.3.2 Generalization to Other Types of Back-Ends and Access Environments

Obviously, the MundoProxy needs to be aware of the type of back-end used for the application. For example, if an SOA back-end is used, which is most common for business Web applications, calls to an SOA infrastructure must be cached. If another type of back-end is used, e.g., a J2EE back-end, calls to this back-end must be cached. However, the prediction code is defined in terms of generic calls, i.e., combinations of method names and concrete parameters. So the concept is useful for other types of back-ends as well. Likewise, the part implementing the cache in the browser needs to be changed. Again the piggybacking mechanism is agnostic to the type of back-ends marshaled over the HTTP connection. It should be relatively easy to implement similar solutions based on the provided algorithms.

It would also be interesting to test the approach with a REST-based back-end. In this case, the call semantics, i.e., whether a request can be prefetched or not, should become clear from the HTTP method used by the call. It would no longer be necessary to annotate the back-end services.

The latency is not affected by the nature of the access environment and works equally well in desktop access environments, mobile access environments and post-desktop access environments. It is, however, particularly useful in mobile post-desktop environments, as these often feature only a small screen, limited input capabilities and low-bandwidth network connections. These characteristics make it much harder for the user to perform any task compared to the situation in desktop access environments. The increased effort for the user can manifest as additional time required to complete a task or additional stress experienced by the user.

7.3.3 Controlling a Smart Environment from a Web Browser

The MundoMonkey browser extension as developed in this thesis had the aim of allowing the user to control Web applications. Therefore, the connection to MundoCore was mainly used for receiving input data from the post-desktop environment and to a limited degree for sending output data to resources in the post-desktop access environment. However, in principle other features of the post-desktop access environment could also be used. For example, this way a script in a Web page could get access to a coffee machine in the

same MundoCore environment as the browser. This could be desirable, as this would open up a much more lightweight path towards development of smart environments. Virtually every user knows how to access a Web page, and this would be all it needs to inject new functionality into a smart environment. Orchestration and coordination of the different resources could be performed by a crowd of developers. This would enable a better user experience for smart environments, just like end-user scripts do for the Web.

Chapter A

Tasks from the MineExplorer User Study

The MineExplorer User Study was conducted with German-speaking participants, therefore the tasks were given in German. Section 6.2 contains English translations of a sample task as well of the questions found in the questionnaire. This annex includes the task set as it was handed out to the participants, including German descriptions of the tasks for the experimenter.

A.1 Aufgaben

Aufgabe 1 / Aufgabe 3

Der Proband stellt sich vor, er sei in einem öffentlichen Gebäude. Wie immer hat er sein persönliches Gerät dabei, nämlich sein iPhone. Er empfängt in seiner E-Mail Applikation eine Nachricht, die er beantworten möchte. Für längere E-Mails ist ihm aber das Display zu klein und schreiben würde ihm mit einer Tastatur auch leichter fallen. Daher möchte er die Geräte in seiner Umgebung nutzen. Seine Aufgabe ist es nun eine öffentliche Tastatur und einen Monitor zu assoziieren. AnschlieSSend sollen beide Geräte mit der Email-Applikation verbunden werden. Da die E-Mail Applikation auf seinem persönlichen Gerät ausgeführt wird ist sie bereits assoziiert.

Konfigurationen

INITIALKONFIGURATION

Available US	Associated US	Connected US
(Keyboard) (Display1)	(E-Mail Application)	-

ZIELKONFIGURATION

Available US	Associated US	Connected US
-	(Keyboard) (Display1) (E-Mail Application)	(E-Mail Application, Keyboard) (E-Mail Application, Display1)

Aufgabe 2 / Aufgabe 4

Der Benutzer hat die E-Mail fertig geschrieben und weggeschickt. Da er die Geräte in seiner Umgebung nicht mehr benötigt, soll er die Verbindungen trennen und die Assoziationen aufheben.

Konfigurationen

INITIALKONFIGURATION

Available US	Associated US	Connected US
-	(Keyboard) (Display 1) (E-Mail Application)	(E-Mail Application, Keyboard) (E-Mail Application, Display1)

ZIELKONFIGURATION

Available US	Associated US	Connected US
(Keyboard) (Display 1)	(E-Mail Application)	-

Aufgabe 5

Nun befindet sich der Benutzer in einer anderen Umgebung. Diese Umgebung bietet noch viel mehr Geräte (wie eine Kaffee Maschine oder eine Webcam) die er verwenden könnte. Sie sind aber irrelevant für ihn sind, da er wieder eine E-Mail schreiben und dafür einen Monitor und eine Tastatur verwenden möchte.

Konfigurationen

INITIALKONFIGURATION

Available US	Associated US	Connected US
(Saeco Coffee Machine) (Wii Remote) (Webcam) (Keyboard) (Display1)	(E-Mail Application)	-

ZIELKONFIGURATION

Available US	Associated US	Connected US
(Saeco Coffee Machine) (Wii Remote) (Webcam)	(Keyboard) (Display1) (E-Mail Application)	(E-Mail Application , Key- board) (E-Mail Application, Dis- play1)

Aufgabe 6

In der Umgebung des Benutzers befinden sich weitere Menschen, die ebenfalls Geräte verwenden. Um wieder eine E-Mail schreiben zu können muss er sich also die jeweiligen freien Geräte aussuchen und eine Konfiguration mit diesen vornehmen.

Konfigurationen

INITIALKONFIGURATION

Available US	Associated US	Connected US
(Keyboard) (Display1 <i>is in use</i>) (Display2) (Display3 <i>is in use</i>)	(E-Mail Application)	-

ZIELKONFIGURATION

Available US	Associated US	Connected US
(Display1 <i>is in use</i>) (Display3 <i>is in use</i>)	(E-Mail Application) (Display2) (Keyboard)	(E-Mail Application, Keyboard) (E-Mail Application, Display2)

Aufgabe 7

Der Benutzer muss nun für die Geräte die er nutzen will bezahlen. Aus diesem Grund soll er durch zu Hilfenahme der Detailinformationen die günstigsten Geräte aussuchen und eine Konfiguration mit den Geräten vornehmen.

Konfigurationen

INITIALKONFIGURATION

Available US	Associated US	Connected US
(Keyboard) (Display1 <i>expensive</i>) (Display2 <i>expensive</i>) (Display3 <i>cheapest</i>)	(E-Mail Application)	-

ZIELKONFIGURATION

Available US	Associated US	Connected US
(Display1) Display2)	(E-Mail Application) (Display3) (Keyboard)	(E-Mail Application, Keyboard) (E-Mail Application, Display3)

Aufgabe 8

Der Proband stellt sich vor, dass er gerade dabei ist seine E-Mail zu schreiben. Nach kurzer Zeit tritt ein Fehler in seiner Konfiguration auf. Seine Aufgabe ist es dann zu versuchen die Konfiguration wiederherzustellen bzw. eine Alternativkonfiguration zu finden mit der er seine E-Mail in einer anderen Form eingeben kann.

Konfigurationen

INITIALKONFIGURATION

Available US	Associated US	Connected US
(Voice Recognition Service)	(Keyboard: <i>Association will fail</i>) (Display 1) (E-Mail Application)	(E-Mail Application , Keyboard) (E-Mail Application , Display 1)

ZWISCHENKONFIGURATION

Available US	Associated US	Connected US
(Voice Recognition Service) (Keyboard: <i>not Associable</i>)	(Display 1) (E-Mail Application)	(E-Mail Application , Keyboard)

ZIELKONFIGURATION

Available US	Associated US	Connected US
(Keyboard)	(Voice Recognition Service) (Display 1) (E-Mail Application)	(E-Mail Application , Voice Recognition Service) (E-Mail Application , Display 1)

Aufgabe 9

Der Proband stellt sich vor, dass er wieder dabei ist seine E-Mail zu schreiben. Bei einem Fehler soll er ähnlich zu Aufgabe 8 entsprechend reagieren.

Konfigurationen

INITIALKONFIGURATION

Available US	Associated US	Connected US
(Display 2)	(Voice Recognition Service) (Display 1) (E-Mail Application)	(E-Mail Application , Voice Recognition Service) (E-Mail Application , Display 1) <i>Connection will fail</i>

ZWISCHENKONFIGURATION

Available US	Associated US	Connected US
(Display 2)	(Voice Recognition Service) (Display 1) (<i>not connectable</i>) (E-Mail Application)	(E-Mail Application , Voice Recognition Service)

ZIELKONFIGURATION

Available US	Associated US	Connected US
(<i>Display 1</i>)	(Keyboard) (Display 2) (Voice Recognition Service) (<i>Display 1</i>) (E-Mail Application)	(E-Mail Application , Voice Recognition Service) (E-Mail Application , Display 2)

List of Figures

2.1	Xerox Star user interface: The beginning of the Desktop Computing era. . .	9
2.2	Pen-and-paper-based operation environment.	11
2.3	Voice-based operation environment in a living room.	11
2.4	An interactive application contains domain code and a interaction code. It is executed in an operating environment and accessed by the user in an access environment. The access system may comprise operating system services and library code. It may also be distributed, connecting to the access environment over a network connection. In this case, the access system runs in another operating environment in the access system, which is not drawn in the picture.	13
2.5	Components of the Web access system used in SOA backed Web applications comprise the Web browser, the Web server and optionally a Web application framework, like Ruby on Rails [ROR].	18
2.6	The Cameleon framework standardizes four layers of models for UI development.	20
2.7	First step of the scenario	22
2.8	Second step of the scenario	23
2.9	Third step of the scenario	23
2.10	Effect of the context-aware suggestions in the first step of the scenario (see Figure 2.7)	24
2.11	Effect of the context-aware prefilling in the second step of the scenario (see Figure 2.8)	24
2.12	Effect of context-aware highlighting in the third step of the scenario (see Figure 2.9)	25
2.13	Participants did not make more errors in the <i>wrong</i> setting compared to the inactive baseline. With correct context-aware support, the number of errors decreased.	25
2.14	Results from the questionnaire.	26
2.15	Contextual information injected into the Deutsche Bahn Web Application .	27
2.16	Contextual information injected into an enterprise CRM application.	28
3.1	Taxonomy for systems from related work.	41
3.2	Meta UI for splitting a Web application to different devices from [SCCF08]	44
3.3	GUI generated by the MASP environment from a AUI.	45
3.4	Instance of a <i>plastic</i> UI as generated by the I-AM system.	45



3.5	Examples of Phidgets hardware	50
3.6	OpenInterface operation environment configuration tool.	52
3.7	ICON operation environment configuration tool.	52
3.8	Three variants of a GUI generated by SUPPLE.	56
3.9	Comparison of state of the art with respect to support for output modifications (Requirement R4). Web-based approaches, in particular approaches using end-user scripting hit the sweet spot in the trade-off between modification capabilities and modification complexity.	61
3.10	State of the art with respect to support for open adaptiveness and simultaneous modification of output and input processing. No current approach provides the ability to modify the processing of input as well as output combined with the capability to let the end-user introduce such modifications in an open-adaptive way. This sweet spot in the upper right quadrant of the diagram is currently	62
3.11	Reification makes the mapping between the input and output data and the UI explicit (3.11b). Without reification, the mapping is implicitly defined (3.11a).	63
4.1	Overview of the steps the user performs to access and interact with a CRM Web application in a post-desktop access environment using the Web access system with the proposed extensions. The meta operations are supported by the new model for managing Web applications in post-desktop access environments. Redistribution/remolding and interacting with the Web application are supported by a novel approach for adapting interaction with Web applications to post-desktop access environments. Finally, access to the back-end is improved by the intelligent caching and prefetching mechanism.	68
4.2	Integration of the proposed concepts in the Web access system (see Figure 2.5)	70
4.3	The upper half of the picture shows the physical access environment in which a user interacts with a room-control Web application using a pointing resource and a speech input resource. Output is presented to the user on the TV screen. The lower half schematically shows how the interaction resources are connected to the UI with interaction strategies by MundoMonkey.	76
79figure.caption.101		
4.5	Without piggybacking, the proxy returns only one response to the client for each request.	83
4.6	With piggybacking, the proxy can send multiple responses to the client, potentially generating additional cache hits.	85
5.1	MundoWeb overall architecture in the context of the Web access system (see Figure 2.5).	88
5.2	Zooming a picture with the MLI strategy.	92



5.3	The meta model serves as a canonical representation of the physical access environment. The meta UI links to this and is synchronized by reacting to events.	98
5.4	Overview of the elements of the STUD meta model.	99
5.5	Life-cycle of a service or task. Interaction services necessarily become available before they can be associated. If they are deassociated they become available again. Tasks may be directly associated from an unavailable state. If they are deassociated, they return to unavailable state.	103
5.6	Flow of events in the sample scenario.	106
5.7	Sequential implementation of piggybacking	118
5.8	Prefetching is performed in idle periods on the proxy, the client continuously parses the response text and the UI update is performed before the piggybacked results are added to the cache.	119
6.1	Sample page used in the user study. The character link could be activated by saying "Click on character, please".	123
6.2	One rule of the grammar generated for the sample page.	124
6.3	Setup in the voice-interaction-strategy condition used in the study	125
6.4	Setup for the built-in Internet Explorer strategy condition used in the study	125
6.5	Our voice-interaction strategy achieved a significantly higher SUS usability score compared to the built-in Internet Explorer strategy.	126
6.6	Screenshot of the message board application using pen input.	127
6.7	User interacting with a Web application in the browser using a pen.	128
6.8	Example for a federated access environment in a living room.	130
6.9	In Mundo a pervasive environment is dynamically constructed by end-users. Users can <i>associate</i> resources into the MINE.	132
6.10	MineExplorer on the iPhone	133
6.11	One task from the user test.	134
6.12	Time on task for the different tasks from the user test.	135
6.13	SUS scores for the 20 participants.	136
6.14	Questions regarding the intelligibility of the underlying concepts.	137
6.15	This diagram shows how a request is handled at the MundoProxy and at which checkpoints we took time measurements.	139
6.16	Time comparison for two requests with and without prefetching. Depicted are the UMTS average times for requests no. 3 and 4 of the scenario.	146
6.17	Comparison of prefetching and baseline behavior using UMTS.	146
6.18	Comparison of user-perceived latency in different networks, averaged across all 11 requests of the scenario.	147
6.19	Comparison of accumulated average network times. Reduction of network time directly corresponds with battery power saving.	148
6.20	Steps to derive call traces starting from the ITIL standard process.	149



6.21	Comparison of prediction with pure caching and the impractical case of a persistent cache. The upper bound is given by the amount of prefetchable invocations.	151
6.22	The average prediction success rate “converges” after processing 5–10 incidents for both algorithms.	152
6.23	The threshold parameter controls the FxL prediction algorithm. Lower thresholds lead to higher hit rates, but also add communication overhead. A hit rate of 15.2% is already attainable at 10% overhead.	153
6.24	Comparison of user-perceived latency under three different test conditions using an iPhone connected via EDGE.	153
6.25	Comparison of total user-perceived latency and online time in the analyzed incident. The bars were scaled to start at 18.97 seconds, because this is the baseline back-end time.	154
6.26	Hybrid client architecture for a mobile process application accessing an SOA back-end. The rich client is implemented in JavaScript.	155

List of Tables

2.1	The table shows how the requirements elicited in [TZV03] map to the requirements used in this thesis.	34
2.2	The table shows how the requirements elicited in [NMH ⁺ 02] map to the requirements used in this thesis.	36
2.3	Overview of the requirements for improving the Web access system.	38
3.1	Overview of the support for the different modification scenarios in the related work.	60
3.2	Comparison of different approaches for reifying the mapping between interaction services and UI. Instrumental Interaction, as a theoretical framework, does not provide any implementation.	64
3.3	Types of meta UIs used in the related work.	65
4.1	Complete meta functionality needed for accessing Web applications in post-desktop environments.	81
5.1	Overview of the operations and events in the meta object protocol for STUD.101	
5.2	Meta operations from Table 4.1. The differentiation between connecting input and output resources is handled by interaction strategies. They are treated as equal in the meta UI.	105
5.3	Network characteristics according to [Goo09]	118
5.4	Comparison with other prefetching and caching approaches.	120
6.1	Format of the events generated by the pen interaction resource.	127
6.2	Proxy overhead measured w/o prefetching.	143
6.3	Proxy overhead measured with prefetching.	144
6.4	Client overhead for caching measured with and w/o prefetching.	145
6.5	Comparison of the MundoMonkey interaction strategies presented in the case studies to the requirements from Chapter 2	156
6.6	Comparison of different approaches for reifying the mapping between interaction services and UI. MundoMonkey provides more features than existing systems from Table 3.2	157
6.7	Reduction of user-perceived latency and online times achieved by the different caching and prefetching approaches on realistic call traces.	157



List of Acronyms

AJAX	Asynchronous JavaScript and XML
AUGUR	Augmenting User Interfaces with Prediction, Modality and Context through Reasoning
API	Application Programmer Interface
AUI	Abstract User Interface
CGI	Common Gateway Interface
CHI	ACM Conference on Human Factors in Computing Systems
CRM	Customer Relationship Management
CSS	Cascading Stylesheet
CUI	Concrete User Interface
DIS	ACM Designing Interactive Systems Conference
DL	Dialog Layer of the Arch Model
DOM	Document Object Model
EICS	The ACM Symposium on Engineering Interactive Computing Systems
EIS	Enterprise Information Software
ERP	Enterprise Resource Planning
FC	Functional Core Layer
FCA	Functional Core Adapter Layer
FUI	Final User Interface
GUI	Graphical User Interface
HCI	Human Computer Interaction
HID	Human Interface Device
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
J2EE	Java 2 Platform, Enterprise Edition

JSON JavaScript Object Notation
LED Light Emmiting Diode
LI Logical Interaction Layer
MDA Model Driven Architecture
MDD Model Driven Development
MDUID Model Driven User Interface Development
ME Minimal Entity
MVC Model View Controller
OMG Object Management Group
OS Operating System
PC Personal Computer
PDA Personal Digital Assistant
PHP PHP Hypertext Preprocessor
PIM Platform Independent Model
PI Physical Interaction Layer
PSM Platform Specific Model
PUC Personal Unified Controller
RDP Remote Desktop Protocol
REST Representational State Transfer
RFID Radio-Frequency Identification
SOA Service Oriented Architecture
SOAP Simple Object Access Protocol
SPA Sequence Prediction Algorithm
STUD Service - Task - User - Device
SUS System Usability Scale
SVG Scalable Vector Graphic
TCP Transmission Control Protocol
UI User Interface
UIC User Interface Code
UID User Interface Description
UIDL User Interface Description Language

UIML User Interface Markup Language
UIMS User Interface Management System
UIST ACM Symposium on User Interface Software and Technology
UPnP Universal Plug and Play
USB Universal Serial Bus
UML Unified Modeling Language
URL Uniform Resource Locator
URI Uniform Resource Identifier
VM Virtual Machine
WIMP Windows, Icons, Menus, and Pointers
WSDL Web Service Description Language
WSDL-S Web Service Description Language with Semantic Extensions
WSMO Web Service Modeling Ontology
WWW World Wide Web
XML Extensible Markup Language



Bibliography

- [ABB⁺09] Erwin Aitenbichler, Alexander Behring, Dirk Bradler, Melanie Hartmann, Leonardo Martucci, Max Mühlhäuser, Sebastian Ries, Dirk Schnelle-Walka, Daniel Schreiber, Jürgen Steimle, and Thorsten Strufe. Shaping the future internet. In *Proceedings of the International CompanionAble Workshop IoPTS*, 2009.
- [AE08] Ibrahim Armac and Daniel Evers. Client side personalization of smart environments. In *Proceedings of the international workshop on Software architectures and mobility*, pages 57–59, Leipzig, Germany, 2008. ACM.
- [AFM⁺05] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web service semantics - wsdl-s, 11 2005. <http://www.w3.org/Submission/WSDL-S/>.
- [Ait09] Erwin Aitenbichler. Mundocore c++, Dec 2009. <https://wiki.tk.informatik.tu-darmstadt.de/bin/view/Mundo/WebHome>.
- [AKM04] Erwin Aitenbichler, Jussi Kangasharju, and Max Mühlhäuser. Talking Assistant: A Smart Digital Identity for Ubiquitous Computing. In *Advances in Pervasive Computing*, pages 279–284. Austrian Computer Society (OCG), Austrian Computer Society (OCG), 2004.
- [AKM07] Erwin Aitenbichler, Jussi Kangasharju, and Max Mühlhäuser. Mundocore: A light-weight infrastructure for pervasive computing. *Pervasive and Mobile Computing*, 3(4):332–361, Aug 2007.
- [ANO] Anoto group ab. <http://www.anoto.com/>.
- [AS07] Richard Atterer and Albrecht Schmidt. Tracking the interaction of users with ajax applications for usability testing. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1347–1350, New York, NY, USA, 2007. ACM.
- [Bal08] Lionel Balme. *Interfaces homme-machine plastiques: Une approche par composants dynamiques*. PhD thesis, Laboratoire d’Informatique de Grenoble (LIG), Université Joseph Fourier, 2008.
- [BBS01] Greg J. Badros, Alan Borning, and Peter J. Stuckey. The cassowary linear arithmetic constraint solving algorithm. *ACM Trans. Comput.-Hum. Interact.*, 8(4):267–306, 2001.

-
- [BDB⁺04] Lionel Balme, Alexandre Demeure, N. Barralon, Joëlle Coutaz, Gaëlle Calvary, et al. Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces. *Lecture Notes in Computer Science*, pages 291–302, 2004.
- [Bea00] Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453, The Hague, The Netherlands, 2000. ACM.
- [BF05] Hrvoje Benko and Steven Feiner. Multi-monitor mouse. In *Extended abstracts on Human factors in computing systems*, pages 1208–1211, New York, NY, USA, 2005. ACM.
- [BF07] Hrvoje Benko and Steven Feiner. Pointer warping in heterogeneous multi-monitor environments. In *Proceedings of Graphics Interface*, pages 111–117, New York, NY, USA, 2007. ACM.
- [BL07] Jeffrey P. Bigham and Richard E. Ladner. Accessmonkey: a collaborative scripting framework for web users and developers. In *Proceedings of the international cross-disciplinary conference on Web accessibility*, pages 25–34, New York, NY, USA, 2007. ACM.
- [BLFA08] Marco Blumendorf, Grzegorz Lehmann, Sebastian Feuerstack, and Sahin Albayrak. Executable models for human-computer interaction. In *DSV-IS*, pages 238–251, 2008.
- [BLMA⁺01] Michel Beaudouin-Lafon, Wendy E. Mackay, Peter Andersen, Paul Janecek, Mads Jensen, Henry Michael Lassen, Kasper Lund, Kjeld Høyer Mortensen, Stephanie Munck, Anne V. Ratzler, Katrine Ravn, Søren Christensen, and Kurt Jensen. Cpn/tools: A post-wimp interface for editing and simulating coloured petri nets. In *Proceedings of the International Conference on Application and Theory of Petri Nets*, pages 71–80, London, UK, 2001. Springer-Verlag.
- [Blu09] Marco Blumendorf. *Multimodal Interaction in Smart Environments A Model-based Runtime System for Ubiquitous User Interfaces*. PhD thesis, Technische Universität Berlin, 2009.
- [BM05a] Elmar Braun and Max Mühlhäuser. Automatically generating user interfaces for device federations. In *Proceedings of the IEEE International Symposium on Multimedia*, pages 261–268, Washington, DC, USA, 2005. IEEE Computer Society.
- [BM05b] Elmar Braun and Max Mühlhäuser. Interacting with federated devices. In *Advances in Pervasive Computing, Adjunct Proceedings of the Third International Conference on Pervasive Computing*, pages 153–160. Austrian Computer Society, 2005.

-
-
- [BMRB07] Rafael Ballagas, Faraz Memon, Rene Reiners, and Jan Borchers. istuff mobile: Rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1107–1116, New York, NY, USA, 2007. ACM Press.
- [BPM09] Alexander Behring, Andreas Petter, and Max Mühlhäuser. Rapidly modifying multiple user interfaces of one application - leveraging multi-level dialogue refinement. In *ICSOFT (1)*, pages 344–347. INSTICC Press, Jul 2009.
- [Bro96] J. Brooke. Sus - a quick and dirty usability scale. *Usability Evaluation in Industry*, pages 189–194, 1996.
- [BSF04] Rafael Ballagas, Andy Szybalski, and Armando Fox. Patch panel: Enabling control-flow interoperability in ubicomp environments. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*, page 241, Washington, DC, USA, 2004. IEEE Computer Society.
- [BSSW02] Dirk Balfanz, Diana K. Smetters, Paul Stewart, and H. Chi Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *NDSS*, 2002.
- [Bux83] William Buxton. Lexical and pragmatic considerations of input structures. *SIGGRAPH Comput. Graph.*, 17(1):31–37, 1983.
- [Bux02] William Buxton. Less is more (more is less). In *The invisible future: the seamless integration of technology into everyday life*, pages 145–179. McGraw-Hill, Inc., New York, NY, USA, 2002.
- [BWR⁺05] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *Proceedings of the ACM symposium on User interface software and technology*, pages 163–172, New York, NY, USA, 2005. ACM.
- [Cao02] Guohong Cao. Proactive Power-Aware Cache Management for Mobile Computing Systems. *IEEE Transactions on Computers*, 51(6):608–621, 2002.
- [CBA⁺07] Joëlle Coutaz, Lionel Balme, X. Alvaro, Gaëlle Calvary, A. Demeure, and J.-S. Sottet. An mde-soa approach to support plastic user interfaces in ambient spaces. In *Proceedings of the 4th international conference on Universal access in human-computer interaction*, pages 63–72, Berlin, Heidelberg, 2007. Springer-Verlag.
- [CCDG05] Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Commun. ACM*, 48:49–53, March 2005.
- [CCT01] Gaëlle Calvary, Joëlle Coutaz, and David Thevenin. A unifying reference framework for the development of plastic user interfaces. In *Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer*

-
- Interaction*, EHCI '01, pages 173–192, London, UK, 2001. Springer-Verlag.
- [CCT⁺02] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Nathalie Souchon, Laurent Bouillon, Murielle Florins, and Jean Vanderdonckt. Plasticity of user interfaces: A revised reference framework. In *TAMODIA*, pages 127–134, 2002.
- [CCT⁺03] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
- [CDK05] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley Longman, Amsterdam, 4th rev. ed. edition, June 2005.
- [CGB04] Michael H. Cohen, James P. Giangola, and Jennifer Balogh. *Voice User Interface Design*. Addison-Wesley Professional, February 2004.
- [Chi00] Shigeru Chiba. Load-time structural reflection in java. In *Proceedings of the European Conference on Object-Oriented Programming*, pages 313–336, London, UK, 2000. Springer-Verlag.
- [CHML06] Scott Carter, Amy Hurst, Jennifer Mankoff, and Jack Li. Dynamically adapting guis to diverse input devices. In *Proceedings of the international ACM SIGACCESS conference on computers and accessibility*, pages 63–70, New York, NY, USA, 2006. ACM.
- [CLC05] Tim Clerckx, Kris Luyten, and Karin Coninx. Dynamo-aid: A design process and a runtime architecture for dynamic model-based user interface development. In Rémi Bastide, Philippe Palanque, and Jörg Roth, editors, *Engineering Human Computer Interaction and Interactive Systems*, volume 3425 of *Lecture Notes in Computer Science*, pages 871–876. Springer Berlin Heidelberg, 2005.
- [CLV⁺03] Karin Coninx, Kris Luyten, Chris Vandervelpen, Jan Bergh, and Bert Creemers. Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In *Human-Computer Interaction with Mobile Devices and Services*, volume 2795 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2003.
- [CNS⁺95] Joëlle Coutaz, Laurence Nigay, D. Salber, A. Blandford, J. May, and R. M. Young. Four easy pieces for assessing the usability of multimodal interaction: The CARE properties. In *Proceedings of INTERACT*, pages 115–120, Lillehammer, June 1995.
- [Cou87] Joëlle Coutaz. Pac: An object oriented model for implementing user interfaces. *SIGCHI Bull.*, 19:37–41, October 1987.

-
- [Cou07] Joëlle Coutaz. Meta-User Interfaces for Ambient Spaces. In *Task Models and Diagrams for Users Interface Design (TAMODIA '07)*, pages 1–15, 2007.
- [DCC08] Alexandre Demeure, Gaëlle Calvary, and Karin Coninx. Comet(s), a software architecture style and an interactors toolkit for plastic user interfaces. In T. C. Graham and Philippe Palanque, editors, *Interactive Systems. Design, Specification, and Verification*, pages 225–237, Berlin, Heidelberg, 2008. Springer-Verlag.
- [Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001.
- [DF04] Pierre Dragicevic and Jean-Daniel Fekete. Support for input adaptability in the icon toolkit. In *IProceedings of the 6th international conference on Multimodal interfaces*, pages 212–219, New York, NY, USA, 2004. ACM.
- [DOM03] Document object model (dom) level 2 html specification, Jan 2003. <http://www.w3.org/TR/DOM-Level-2-HTML/>.
- [DPSG07] Josep Domenech, Ana Pont, Julio Sahuquillo, and Jose A. Gil. A User-Focused Evaluation of Web Prefetching Algorithms. *Computer Comm.*, 30(10):2213–2224, 2007.
- [DS01] Paulo Pinheiro Da Silva. User interface declarative models and development environments: a survey. In *Proceedings of the 7th international conference on Design, specification, and verification of interactive systems, DSV-IS'00*, pages 207–226, Berlin, Heidelberg, 2001. Springer-Verlag.
- [ED05] Kamal Elbashir and Ralph Deters. Transparent Caching for Nomadic WS Clients. In *IEEE ICWS*, pages 177–184, 2005.
- [EJM00] A. N. Eden, B. W. Joh, and T. Mudge. Web Latency Reduction via Client-Side Prefetching. In *IEEE ISPASS*, pages 193–200, 2000.
- [FBSA08] Sebastian Feuerstack, Marco Blumendorf, Veit Schwartze, and Sahin Albayrak. Model-based layout generation. In *Proceedings of the working conference on Advanced visual interfaces*, pages 217–224, New York, NY, USA, 2008. ACM.
- [FCLJ99] Li Fan, Pei Cao, Wei Lin, and Quinn Jacobson. Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance. In *ACM SIGMETRICS*, pages 178–187, 1999.
- [FdS08] Diogo Ferreira and M. M da Silva. Using process mining for ITIL assessment: a case study with incident management. In *Proceedings of the UKAIS Conference*, apr 2008.
- [Fie00] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

-
- [GB02] Saul Greenberg and Michael Boyle. Customizable physical interfaces for interacting with conventional applications. In *Proceedings of the ACM symposium on User interface software and technology*, pages 31–40, New York, NY, USA, 2002. ACM.
- [GC07] Karthik Gopalratnam and Diane J. Cook. Online sequential prediction via incremental parsing: The active lezi algorithm. *IEEE Intelligent Systems*, 22(1):52–58, 2007.
- [GF01] Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the ACM symposium on User interface software and technology*, pages 209–218, New York, NY, USA, 2001. ACM.
- [Goo09] Google Inc. Android Emulator Documentation, March 2009.
- [Goo10] Google. Google cloud print, 2010. <http://code.google.com/apis/cloudprint/docs/overview.html>.
- [GRE09] Greasepot - the weblog about greasemonkey, Oct 2009. <http://www.greasespot.net/>.
- [GRV04] Robert L. Glass, V. Ramesh, and Iris Vessey. An analysis of research in computing disciplines. *Commun. ACM*, 47(6):89–94, 2004.
- [GSH⁺09] Andreas Göb, Daniel Schreiber, Louenas Hamdi, Erwin Aitenbichler, and Max Mühlhäuser. Reducing user perceived latency with a middleware for mobile soa access. In *Proceedings of the IEEE Conference on Web Services (ICWS)*, pages 366–373, Los Angeles, USA, 2009.
- [GW04] Krzysztof Gajos and Daniel S. Weld. SUPPLE: automatically generating user interfaces. In *Proceedings of the international conference on Intelligent user interface*, pages 93–100, New York, NY, USA, 2004. ACM Press.
- [GWW08] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *Proceeding of the SIGCHI conference on Human factors in computing systems*, pages 1257–1266, New York, NY, USA, 2008. ACM.
- [HGJSS09] Volker Hoyer, Florian Gilles, Till Janner, and Katarina Stanoevska-Slabeva. Sap research rooftop marketplace: Putting a face on service-oriented architectures. In *Proceedings of the Congress on Services*, pages 107–114, Washington, DC, USA, 2009. IEEE Computer Society.
- [HI01] Karen Henriksen and Jadwiga Indulska. Adapting the web interface: an adaptive web browser. In *Proceedings of the Australasian conference on User interface*, pages 21–28. IEEE Computer Society, 2001.

-
-
- [HM09] Melanie Hartmann and Max Mühlhäuser. Context-aware form filling for web applications. In *Proceedings of the IEEE International Conference on Semantic Computing*, pages 221–228. IEEE, 2009.
- [HMS⁺01] Lars Erik Holmquist, Friedemann Mattern, Bernt Schiele, Petteri Alahuhta, Michael Beigl, and Hans-Werner Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *Proceedings of Ubicomp*, pages 116–122, 2001.
- [Hra08] Andreas Hrabal. Multimediale Interaktion mit der Wii-Remote. Master’s thesis, Telekooperation, TU Darmstadt, 2008.
- [HS07] Melanie Hartmann and Daniel Schreiber. Prediction algorithms for user actions. In *Proceedings of Lernen Wissen Adaption, ABIS*, 2007.
- [HS08] Melanie Hartmann and Daniel Schreiber. Proactively adapting interfaces to individual users for mobile devices. In *Proceedings of the 5th international conference on Adaptive Hypermedia and Adaptive Web-Based Systems, AH ’08*, pages 300–303, Berlin, Heidelberg, 2008. Springer-Verlag.
- [HS09] Melanie Hartmann and Daniel Schreiber. Augur: Interface adaptation for small screen devices. In Tsvi Kuflik, Shlomo Berkovsky, Francesca Carmagnola, Dominikus Heckmann, and Antonio Krüger, editors, *Advances in Ubiquitous User Modelling*, pages 94–110. Springer-Verlag, Berlin, Heidelberg, 2009.
- [HSK07] Melanie Hartmann, Daniel Schreiber, and Matthias Kaiser. Task models for proactive web applications. In *Proceedings of WEBIST*. INSTICC Press, 2007.
- [HSL08] F. Hamidi, L. Spalteholz, and N. Livingston. Web-speak: A customizable speech-based web navigation interface for people with disabilities. In *Proceedings of the 23rd Annual International Technology & Persons with Disabilities Conference (CSUN08)*, 2008.
- [HSM08] Melanie Hartmann, Daniel Schreiber, and Max Mühlhäuser. Tailoring the interface to individual users. In *International Workshop on Ubiquitous User Modeling*. ACM Press, 2008.
- [HSM09] Melanie Hartmann, Daniel Schreiber, and Max Mühlhäuser. Augur: providing context-aware interaction support. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems, EICS ’09*, pages 123–132, New York, NY, USA, 2009. ACM.
- [HSSM10] Felix Heinrichs, Jürgen Steimle, Daniel Schreiber, and Max Mühlhäuser. Letras: An architecture and framework for ubiquitous pen-and-paper interaction. In *Proceedings of the ACM SIGCHI symposium on Engineering interactive computing systems*, New York, NY, USA, Jun 2010. ACM.

-
-
- [HWDB08] Louenas Hamdi, Huaigu Wu, Serhan Dagtas, and Abdelghani Benharref. Ajax for Mobility: MobileWeaver AJAX Framework. In *Proceedings of the WWW*, pages 1077–1078. ACM, 2008.
- [iti10] itil.org. The portal for information regarding ITIL, ISO20000 and COBIT. <http://www.itil.org>, 2010.
- [IU97] Hiroshi Ishii and Brygg Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA, 1997. ACM.
- [JB02] Nico Jacobs and Hendrik Blockeel. Sequence prediction with mixed order markov chains. In *In Proceedings of the Belgian/Dutch Conference on Artificial Intelligence*, pages 147–154, 2002.
- [jCO10] jCOM1. Welcome to the Future of BPM: S-BPM, 2010.
- [JF02] Brad Johanson and Armando Fox. The event heap: A coordination infrastructure for interactive workspaces. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 83, Washington, DC, USA, 2002. IEEE Computer Society.
- [JFW02] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE pervasive computing*, pages 67–74, 2002.
- [JHWS02] Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. Pointright: experience with flexible input redirection in interactive workspaces. In *Proceedings of the ACM symposium on user interface software and technology*, pages 227–234, New York, NY, USA, 2002. ACM.
- [JRV⁺89] Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. The xerox star: A retrospective. *Computer*, 22(9):11–26, 28–29, 1989.
- [JSO] Json javascript object notation. <http://www.json.org/>.
- [Kas82] David J. Kasik. A user interface management system. In *Proceedings of the 9th annual conference on Computer graphics and interactive techniques, SIGGRAPH '82*, pages 99–106, New York, NY, USA, 1982. ACM.
- [KBWA94] Rick Kazman, Len Bass, Mike Webb, and Gregory Abowd. Saam: a method for analyzing the properties of software architectures. In *Proceedings of the international conference on Software engineering*, pages 81–90, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
- [Küh07] T. Kühne. Making modeling languages fit for Model-Driven development. In *International Workshop on Language Engineering (ATEM)*, 2007.

-
-
- [KLLL04] Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. Papier-mache: toolkit support for tangible input. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 399–406, New York, NY, USA, 2004. ACM.
- [KM07] Tobias Klug and Max Mühlhäuser. Modeling human interaction resources to support the design of wearable multimodal systems. In *Proceedings of international conference on Multimodal interfaces*, pages 299–306, New York, NY, USA, 2007. ACM.
- [KS07] Thomas Kühne and Daniel Schreiber. Can programming be liberated from the two-level style: multi-level programming with deepjava. In *Proceedings of OOPSLA*, pages 229–244, 2007.
- [LAAVM09] Jean-Yves Lionel Lawson, Ahmad-Amr Al-Akkad, Jean Vanderdonckt, and Benoit Macq. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proceedings of the ACM SIGCHI symposium on Engineering interactive computing systems*, pages 245–254, New York, NY, USA, 2009. ACM.
- [LHML08] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. Coscripiter: automating & sharing how-to knowledge in the enterprise. In *Proceeding of the SIGCHI conference on Human factors in computing systems*, pages 1719–1728, New York, NY, USA, 2008. ACM.
- [Lok09] Lokomo Systems. OneCMDB, 2009.
- [LPR05] Holger Lausen, Axel Polleres, and Dumitru Roman. Web service modeling ontology, 6 2005. <http://www.w3.org/Submission/WSMO/>.
- [LS94] Philip Laird and Ronald Saul. Discrete sequence prediction and its applications. *Mach. Learn.*, 15(1):43–68, 1994.
- [LSHA08] Stefan Link, Thomas Schuster, Philip Hoyer, and Sebastian Abeck. Focusing graphical user interfaces in model-driven software development. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, pages 3–8, Washington, DC, USA, 2008. IEEE Computer Society.
- [LTVC06] Kris Luyten, Kristof Thys, Jo Vermeulen, and Karin Coninx. A generic approach for multi-device user interface rendering with uiml. In *CADUI*, pages 175–182, 2006.
- [LV11] Kris Luyten and Geert Vanderhulst. Ubiquitous access to the internet of things. In *Workshop on Interacting with Smart Objects*, 2011.
- [LVC06] Kris Luyten, Jo Vermeulen, and Karin Coninx. Constraint adaptability of multi-device user interfaces. In *Workshop on The Many Faces on Consistency, CHI’2006 workshop*, 2006.

-
- [LVM⁺05] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. Usixml: A language supporting multi-path development of user interfaces. In Rémi Bastide, Philippe Palanque, and Jörg Roth, editors, *Engineering Human Computer Interaction and Interactive Systems*, volume 3425 of *Lecture Notes in Computer Science*, pages 134–135. Springer, 2005.
- [Mac09] I. Scott MacKenzie. The one-key challenge: searching for a fast one-key text entry method. In *Proceedings of the international ACM SIGACCESS conference on Computers and accessibility*, pages 91–98, New York, NY, USA, 2009. ACM.
- [Mae87] Pattie Maes. Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12):147–155, 1987.
- [MG08] Max Mühlhäuser and Iryna Gurevych. *Handbook of Research on Ubiquitous Computing Technology for Real Time Enterprises*. Information Science Reference, 1 edition, January 2008.
- [MH09] Max Mühlhäuser and Melanie Hartmann. Interacting with context. In *Proceedings of the 1st international conference on Quality of context*, pages 1–14, Berlin, Heidelberg, 2009. Springer-Verlag.
- [MHA00] Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 368–375, 2000.
- [MHA07] Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. Interaction techniques for ambiguity resolution in recognition-based interfaces. In *Proceedings of the ACM SIGGRAPH courses*, page 11, New York, NY, USA, 2007. ACM.
- [MHP00] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, 2000.
- [Mic] Microsoft. Asp.net. <http://msdn.microsoft.com/en-us/library/dd561932>.
- [Mic09] Microsoft. Exchange Web Services Architecture, 2009.
- [Mic10] Microsoft. Windows presentation foundation, April 2010. <http://msdn.microsoft.com/en-us/library/ms754130.aspx>.
- [MMBE00] Brad A. Myers, Robert C. Miller, Benjamin Bostwick, and Carl Evankovich. Extending the windows desktop interface with connected handheld computers. In *Proceedings of the conference on USENIX Windows Systems Symposium*, pages 8–8, Berkeley, CA, USA, 2000. USENIX Association.

-
-
- [MNWM04] B.A. Myers, J. Nichols, J.O. Wobbrock, and R.C. Miller. Taking handheld devices to the next level. *Computer*, 37(12):36–43, 2004.
- [MPS03] Giulio Mori, Fabio Paternò, and Carmen Santoro. Tool support for designing nomadic applications. In *Proceedings of the international conference on Intelligent user interfaces*, pages 141–148, New York, NY, USA, 2003. ACM.
- [MR92] Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 195–202, New York, NY, USA, 1992. ACM Press.
- [MSH09] Markus Miche, Daniel Schreiber, and Melanie Hartmann. Core services for smart products. In *Smart Products: Building Blocks of Ambient Intelligence (AmI-Blocks), collocated with AmI*, 2009.
- [Mye90] Brad A. Myers. A new model for handling input. *ACM Trans. Inf. Syst.*, 8(3):289–320, 1990.
- [Nat] National Instruments. Engineer your algorithms by combining graphical and textual programming. <http://zone.ni.com/wv/app/doc/p/id/wv-23>.
- [NBW05] Stina Nylander, Markus Bylund, and Annika Waern. Ubiquitous service access through adapted user interfaces on multiple devices. *Personal Ubiquitous Comput.*, 9(3):123–133, 2005.
- [NCM07] Jeffrey Nichols, Duen Horng Chau, and Brad A. Myers. Demonstrating the viability of automatically generated user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1283–1292, New York, NY, USA, 2007. ACM.
- [NMH⁺02] Jeffrey Nichols, Brad Myers, Thomas K. Harris, Roni Rosenfeld, Stefanie Shriver, Michael Higgins, and Joseph Hughes. Requirements for automatically generating Multi-Modal interfaces for complex appliances. In *Proceedings of the IEEE International Conference on Multimodal Interfaces*, page 377. IEEE Computer Society, 2002.
- [NMH⁺03] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Kevin Litwack. Personal universal controllers: controlling complex appliances with guis and speech. In *Extended abstracts on Human factors in computing systems*, pages 624–625, New York, NY, USA, 2003. ACM.
- [Nok10] Nokia. Qt cross-platform application and ui framework, June 2010. <http://qt.nokia.com/products>.
- [NRCM06] Jeffrey Nichols, Brandon Rothrock, Duen Horng Chau, and Brad A. Myers. Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 279–288. ACM, 2006.

-
- [OGT⁺99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, 1999.
- [OJN⁺00] Dan R. Olsen, Jr., Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal interaction using xweb. In *Proceedings of the ACM symposium on User interface software and technology*, pages 191–200, New York, NY, USA, 2000. ACM.
- [Ols07] Dan R. Olsen, Jr. Evaluating user interface systems research. In *Proceedings of the ACM symposium on User Interface Software and Technology*, pages 251–258, New York, NY, USA, 2007. ACM.
- [ONP01] Dan R. Olsen, Jr., S. Travis Nielsen, and David Parslow. Join and capture: a model for nomadic interaction. In *Proceedings of the 1ACM symposium on User interface software and technology*, pages 131–140, New York, NY, USA, 2001. ACM.
- [ope] Openinterface forge. <https://forge.openinterface.org/>.
- [OTR09] OTRS Team. Open Ticket Request System, 2009.
- [Pat99] Fabio Paterno. *Model-Based Design and Evaluation of Interactive Applications*. Springer, Berlin, 1 edition, November 1999.
- [PdSB00] Raquel O. Prates, Clarisse S. de Souza, and Simone D. J. Barbosa. Methods and tools: a method for evaluating the communicability of user interfaces. *interactions*, 7(1):31–38, 2000.
- [PJKF03] Shankar Ponnekanti, Brad Johanson, Emre Kiciman, and Armando Fox. Portability, extensibility and robustness in iros. In *PerCom*, pages 11–19, 2003.
- [PK08] Mikko Pervilä and Jussi Kangasharju. Performance of Ajax on Mobile Devices: A Snapshot of Current Progress. In *2nd Intl. Workshop on Improved Mobile User Experience*, 2008.
- [PLF⁺01] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. Icraft: A service framework for ubiquitous computing environments. In *Proceedings of the international conference on Ubiquitous Computing*, pages 56–75, London, UK, 2001. Springer-Verlag.
- [Pro09] The Web Standard Project. Acid2 browser test, Oct 2009. <http://www.webstandards.org/action/acid2>.
- [PSMM08] F. Paterno, C. Santoro, J. Mantyjarvi, and G. Mori. Authoring pervasive multimodal user interfaces. *International Journal of Web Engineering and Technology*, 4(2):235–261, 2008.

-
- [PSS09] Fabio Paterno, Carmen Santoro, and Lucio Davide Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4):1–30, 2009.
- [pur] Pure data. www.puredata.info.
- [RB03] Tom Rodden and Steve Benford. The evolution of buildings and implications for the design of ubiquitous domestic environments. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 9–16, Ft. Lauderdale, Florida, USA, 2003. ACM.
- [RBA09] Dirk Roscher, Marco Blumendorf, and Sahin Albayrak. A meta user interface to control multimodal interaction in smart environments. In *Proceedings of the international conference on Intelligent user interfaces*, pages 481–482, New York, NY, USA, 2009. ACM.
- [Rek04] Jun Rekimoto. Synctap: synchronous user operation for spontaneous network connection. *Personal and Ubiquitous Computing*, 8(2):126–134, 2004.
- [RHC⁺02] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):65–67, 2002.
- [ROR] Ruby on rails guides. <http://guides.rubyonrails.org/>.
- [RRMvdA05] U.V. Riss, A. Rickayzen, H. Maus, and W.M. van der Aalst. Challenges for business process and task management. *Journal of Universal Knowledge Management*, 0(2):77–100, 2005.
- [RS08] Sebastian Ries and Daniel Schreiber. Evaluating user representations for the trustworthiness of interaction partners. In *International Workshop on Recommendation and Collaboration (ReColl’08) in conjunction with the International Conference on Intelligent User Interfaces (IUI’08)*. ACM Press, 2008.
- [Rus06] Alex Russell. Comet: Low latency data for the browser, 3 2006. <http://alex.dojotoolkit.org/2006/03/comet-low-latency-data-for-the-browser/>.
- [SAGM10] Daniel Schreiber, Erwin Aitenbichler, Andreas Goeb, and Max Mühlhäuser. Reducing user perceived latency with a middleware for soa. *International Journal of Web Services Research*, 7(4), 2010.
- [SBM09] Jürgen Steimle, Oliver Brdiczka, and Max Mühlhäuser. Coscribe: Integrating paper and digital documents for collaborative knowledge work. *TLT*, 2(3):174–188, 2009.

-
-
- [SCCF08] Jean-Sébastien Sottet, Gaëlle Calvary, Joëlle Coutaz, and Jean-Marie Favre. A model-driven engineering approach for the usability of plastic user interfaces. In Jan Gulliksen, Morton Harning, Philippe Palanque, Gerrit van der Veer, and Janet Wesson, editors, *Engineering Interactive Systems*, volume 4940 of *Lecture Notes in Computer Science*, pages 140–157. Springer, 2008.
- [SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM.
- [SdF07] Sailesh Sathish and Cristiano di Flora. Supporting smart space infrastructures: a dynamic context-model composition framework. In *MobiMedia*, page 67, 2007.
- [SGAM10] Daniel Schreiber, Andreas Goeb, Erwin Aitenbichler, and Max Mühlhäuser. Reducing user perceived latency in mobile processes. In *Proceedings of the IEEE Conference on Web Services (ICWS)*, 2010.
- [SH08] Daniel Schreiber and Melanie Hartmann. Association: Unobtrusively Creating Digital Contracts with Smart Products. In *Workshop on Smart Products: Building Blocks of Ambient Intelligence*, pages 27–34, 2008.
- [SH09] Daniel Schreiber and Melanie Hartmann. Adding flexible input device support to a web browser with mundomonkey. In *Proceedings of Lernen Wissen Adaption, ABIS*, 2009.
- [SHF⁺07] Daniel Schreiber, Melanie Hartmann, Felix Flentge, Max Mühlhäuser, Thomas Ziegert, and Manuel Görtz. Web based evaluation of proactive multimodal user interfaces. In *International Workshop on Usability of Multimodal User Interfaces*, 2007.
- [SHF⁺08] Daniel Schreiber, Melanie Hartmann, Felix Flentge, Max Mühlhäuser, Manuel Görtz, and Thomas Ziegert. Web based evaluation of proactive user interfaces. *Journal on Multimodal User Interfaces*, 2(1):61–72, 2008.
- [SHM09] Daniel Schreiber, Melanie Hartmann, and Max Mühlhäuser. Mundomonkey: customizing interaction with web applications in interactive spaces. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09, pages 285–290, New York, NY, USA, 2009. ACM.
- [SIG05] SIGACCESS. Sigaccess mission statement, Jan 2005. http://www.sigaccess.org/home/mission_statement/.
- [SLLH08] Leo Spalteholz, Kin Fun Li, Nigel Livingston, and Foad Hamidi. Keysurf: a character controlled browser for people with physical disabilities. In *Proceeding of the international conference on World Wide Web*, pages 31–40,

-
- New York, NY, USA, 2008. ACM.
- [SSF08] Todor Stoitsev, Stefan Scheidl, Felix Flentge, and Max Mühlhäuser. Enabling end users to proactively tailor underspecified, human-centric business processes: "programming by example" of weakly-structured process models. In *ICEIS*, pages 307–320, 2008.
- [Ste09a] Jürgen Steimle. Designing pen-and-paper user interfaces for interaction with documents. In *Proceedings of the International Conference on Tangible and Embedded Interaction*, pages 197–204, New York, NY, USA, 2009. ACM.
- [Ste09b] Jürgen Steimle. Designing pen-and-paper user interfaces for interaction with documents. In *Proceedings of the International Conference on Tangible and Embedded Interaction*, pages 197–204, New York, NY, USA, 2009. ACM.
- [SZG⁺03] Axel Spriestersbach, Thomas Ziegert, Guido Grassel, Michael Wasmund, and Gabriel Dermler. A single source authoring language to enhance the access from mobile devices to web enterprise applications. In *WWW2003 Developers Day Mobile Web Track*, 2003.
- [THLT07] R. Tergujeff, J. Haajanen, J. Leppanen, and S. Toivonen. Mobile SOA: Service Orientation on Lightweight Mobile Devices. In *Proceedings of the IEEE Conference on Web Services (ICWS)*, pages 1224–1225, 2007.
- [TJH10] C. Timmerer, J. Jabornig, and H. Hellwagner. A survey on delivery context description Formats—A comparison and mapping model. *Journal of Digital Information Management*, 8(1), 2010.
- [TZV03] Shari Trewin, Gottfried Zimmermann, and Gregg Vanderheiden. Abstract user interface representations: how well do they support universal access? In *Proceedings of the conference on Universal usability*, pages 77–84, Vancouver, British Columbia, Canada, 2003. ACM.
- [UIM92] A metamodel for the runtime architecture of an interactive system: the uims tool developers workshop. *SIGCHI Bulletin*, 24(1):32–37, 1992.
- [UPN] Universal Plug and Play (UPnP).
- [VC04a] Chris Vandervelpen and Karin Coninx. Towards model-based design support for distributed user interfaces. In *Proceedings of the Nordic conference on Human-computer interaction*, pages 61–70, Tampere, Finland, 2004. ACM.
- [VC04b] Chris Vandervelpen and Karin Coninx. Towards model-based design support for distributed user interfaces. In *Proceedings of the Nordic conference on human-computer interaction*, pages 61–70, New York, NY, USA, 2004. ACM.
- [vdAHW03] W.M.P. van der Aalst, A. H. M. Ter Hofstede, and M. Weske. Business process management: A survey. In *Proc. of the 1st Intl. Conf. on Business*

-
- Process Management, volume 2678 of LNCS*, pages 1–12. Springer-Verlag, 2003.
- [VLC07] Geert Vanderhulst, Kris Luyten, and Karin Coninx. Middleware for ubiquitous Service-Oriented spaces on the web. In *Advanced Information Networking and Applications Workshops, International Conference on*, volume 2, pages 1001–1006, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
- [VLC08] Geert Vanderhulst, Kris Luyten, and Karin Coninx. ReWiRe: Creating Interactive Pervasive Systems that cope with Changing Environments by Rewiring. In *Proceedings of the International Conference on Intelligent Environments*, pages 1–8, 2008.
- [VSL⁺09] Geert Vanderhulst, Daniel Schreiber, Kris Luyten, Max Muhlhauser, and Karin Coninx. Edit, inspect and connect your surroundings: a reference framework for meta-uis. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, EICS '09, pages 167–176, New York, NY, USA, 2009. ACM.
- [WBM00] David J. Ward, Alan F. Blackwell, and David J. C. MacKay. Dasher - a data entry interface using continuous gestures and language models. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, UIST '00, pages 129–137, New York, NY, USA, 2000. ACM.
- [Wei99] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3:3–11, July 1999.
- [Wel89] P. D. Wellner. Statemaster: A uims based on statechart for prototyping and target implementation. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Wings for the mind*, CHI '89, pages 177–182, New York, NY, USA, 1989. ACM.
- [WM03] Jingtao Wang and Jennifer Mankoff. Theoretical and architectural support for input device adaptation. In *Proceedings of the conference on universal usability*, pages 85–92, New York, NY, USA, 2003. ACM Press.
- [WNS07] Nadir Weibel, Moira C. Norrie, and Beat Signer. A model for mapping between printed and digital document instances. In *Proceedings of the ACM symposium on Document engineering*, pages 19–28, New York, NY, USA, 2007. ACM.
- [xfo09] Xforms 1.1, Oct 2009. <http://www.w3.org/TR/2009/REC-xforms-20091020/>.
- [YNK09] Takuto Yanagida, Hidetoshi Nonaka, and Masahito Kurihara. Personalizing graphical user interfaces on flexible widget layout. In *Proceedings of the ACM SIGCHI symposium on Engineering interactive computing systems*, pages 255–264, New York, NY, USA, 2009. ACM.

-
- [Zin08] Michael Zinn. Ausnutzung von Semantik für eine generische Sprachsteuerung. Master's thesis, Telekooperation, TU Darmstadt, 2008.

Wissenschaftlicher Werdegang des Verfassers²

2000–2006 Studium der Informatik an der Technischen Universität Darmstadt, Darmstadt

2005–2006 Diplomarbeit am Lehrstuhl Objektorientierte Metamodellierung
Technische Universität Darmstadt, Darmstadt
“DJ - Eine Programmiersprache mit Tiefer Instanzierung”

2006–2011 Wissenschaftlicher Mitarbeiter am Lehrstuhl für “Telekooperation” an der
Technischen Universität Darmstadt, Darmstadt

² Gemäß §20 Abs. 3 der Promotionsordnung der TU Darmstadt