RWTH AACHEN UNIVERSITY

Nomadic Interfaces in UbiComp

Diploma Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Sören Busch

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Klaus Wehrle

Registration date: February 15th, 2011
Submission date: August 9th, 2011

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, August 2011
Sören Busch

Contents

List of Figures

List of Tables

Abstract

While the different device classes needed to realize the idea of Ubiquitous Computing are readily available at this very moment, the integration of different devices is still severely lacking and cross-device interaction is often still limited to file transfers. Modern research does not provide us with an open system to transfer an application from on device to another.
We propose the concept of "Nomadic Applications" in which an application is no longer tied to a specific system but free to move around between devices using interactions we call "Nomadic Operations". These operations allow users to copy applications to other devices, transferring their whole state with them.
In this thesis the concept of Nomadic Applications is applied to the scenario of personal workspaces. A software system consisting of applications for three different UbiComp device classes was developed using the process of iterative user-centered design. Common practices in today's workspaces have been identified through literature research. The resulting idea is a software system that combines the concept of Nomadic Applications with large displays to provide users with digital whiteboards, from and to which they can freely migrate applications.
After establishing a set of design requirements, a paper prototype was used as a design starting point. It was evaluated in a user study and with these results a software prototype was created. The software system consists of three applications: "NomadicDesktop" that enhances Mac OS X applications with Nomadic Operations, "NomadicBoard" that provides an interface to use Nomadic Applications on a large touch-display, and "NomadicPasteboard" that stores applications on a mobile device. The design and implementation of this system, using the underlying "NomadicApps Framework" will be demonstrated in this thesis.
The resulting software system was evaluated in a two-week qualitative study. The study showed successfully, how the combination of these applications can support different tasks, as the board was used both, as a context and reminder display and as a tool to transition between personal and collaborative work.

Überblick

Ubiquitous Computing ist noch keine Realität, obwohl die dafür notwendigen Geräte schon heute vorhanden sind. Ein Grund dafür ist die mangelnde Integration verschiedener Geräte und die Tatsache dass Interaktion zwischen Geräten oft weiterhin auf Dateitransfere beschränkt ist. Auch moderne Forschung bietet keine offenen Systeme, die Anwendungen unabhängig von ihren Geräten machen.
In unserem vorgeschlagenen Konzept von "Nomadic Applications" sind Anwendungen nicht länger an bestimmte Systeme gebunden sondern können sich, mit Hilfe von Interaktionen, die wir "Nomadic Operations" nennnen und die Nutzern erlaubern, Anwendungen auf andere Geräte zu kopieren, frei zwischen Geräten bewegen.
In dieser Arbeit wird das "Nomadic Applications"-Konzept in dem Szenario persönlicher Arbeitsplätze angewendet. Ein Softwaresystem, drei Anwendungen besteht, wurde mit Hilfe von iterativem userzentriertem Design entwickelt. Die Identifikation häufiger Gebräuche an modernen Arbeitsplẗzen resultierte in einem Softwaresystem dass das "NomadicApplications"-Konzept mit digitalen Whiteboards kombiniert, so dass Nutzer Anwendungen frei von diesem und auf dieses migrieren könnnen.
Nach dem Aufstellen von Designanforderungen, wurde ein Papierprototyp als Design-Anfangspunkt erschaffen. Dieser Prototyp wurde in einer Nutzerstudie evaluiert und mit den Resultaten konnte ein Softwareprototyp kreiert werden. Dieses Softwaresystem besteht aus drei Anwendungen: "NomadicDesktop" erweitert übliche Mac OS X Programme mit nomadischen Möglichkeiten, "NomadicBoard" bietet die Möglichkeit nomadische Anwendungen auf großen berührungsempfindlichen Monitoren zu nutzen und "NomadicPasteboard" kann als mobiles Lager für Anwendungen dienen. Design und Implementierung dieses Systems mit dem unterliegendem "NomadicApps Framework" wird in dieser Arbeit beschrieben.
In einer zweiwöchigen qualitativen Studie konnte beobachtet werden, wie Benutzer das System für zwei verschiedene Zwecke nutzen: Als persönliches Kontextdisplay und als Hilfsmittel für spontane kollaborative Arbeit. So wurde erfolgreich gezeigt, wie das Konzept "Nomadic Applications" auf verschiedene Art und Weisen das Nutzererlebnis am persönlichen Arbeitsplatz aufwerten kann.

Acknowledgements

First of all I want to thank Prof. Dr. Jan Borchers and the whole Media Computing Group for giving me the opportunity to work on this great project.

Jonathan Diehl deserves my utmost gratefulness for laying all the groundwork for this project, giving me feedback all along the way, helping me whenever I needed it, and not strangling me, when he had to read my code.

Thanks go out to everyone, who helped me with all my evaluations, especially the two guys that let me put a really large screen in their office.

Thank you Mario Fraikin, for letting me use and recycle parts of your project. Thank you Coelestin Urban, for proofreading my thesis.

A very special thanks goes to my dear friend Barbara Iwaniuk, who always lent me a shoulder, when times were not that easy, as well as all of my other friends that make life worth living.

And lastly, thanks to my family for supporting me throughout all of university without ever doubting me.

Conventions

Throughout this thesis we use the following conventions.

Important terms are written in *emphasized typeset* when they appear in the text for the first time.

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English.

Chapter 1

Introduction

In his pivotal article Weiser [1991] created the idea of ubiquitous computing to

> "make using a computer as refreshing as taking a walk in the woods."

In his vision, he describes a world, in which computing devices are so tightly woven into everyone's life that they fade into ambience. Awareness of actually using computing devices decreases, due to them being everywhere. Accessibility and integrity of these devices reaches a level, where nobody has to be concerned of any low-level operations anymore.

20 Years later the reality is still far from catching up with his ideas. While the spread of UbiComp devices increased dramatically in the last years, using them in a way Weiser suggested is still hard to realize due to a number of limitations.

UbiComp devices are available but interaction is not

Studies by Dearman and Pierce [2008] and Oulasvirta and Sumari [2007] suggest that while people get increasingly used to operate multiple devices, the drawbacks often outweigh the benefits. For one using multiple devices in one task requires careful preparation to have all the resources, where they need to be at a given time and minimize the

need for transfer. Accessing information across devices is also widely perceived as a problem, because keeping all your data in sync can grow as a burden quickly.

Nomadic Applications are no longer tied to a device

To solve these problems, we propose a metaphor in which interfaces are nomadic. Using a set of easily accessible meta-operations, applications or application parts can be freely moved between devices, carrying all the relevant data with them and adapting to device interfaces on-the-fly. While doing this, the application state stays intact, so the interaction does not feel considerably different from moving windows between two monitors.

1.1 Research Goals

In this thesis I will give an answer to the question, if Nomadic Applications can improve a user's work experience in his personal workspace.

I will present a software system that enhances existing work practices by extending applications with nomadic capabilities and show how users can benefit from this system. Additionally I will demonstrate how such a system that extends existing applications with nomadic functionality can be designed and implemented.

1.2 Nomadic Applications

While the average number of computing devices a regular person uses on a daily basis is steadily increasing (according to Dearman and Pierce [2008], an average person had a collection of 5.96 computing devices in 2008) integration of different devices has not significantly evolved since the introduction of network computing. The connection of different devices is still mostly limited to simple network or USB file transfers.

Figure 1.1: Nomadic Applications in action. A browser gets migrated to another device, where it adapts to a device-appropriate form

Nomadic Applications bring cross device interaction to application level

The concept of Nomadic Applications tries to change this approach to device connectivity. It tries to move the focus away from low-level interactions and elevates the applications themselves to the objects of cross-device interaction.

An environment in which applications are nomadic brings us one step closer to the goal of ubiquitous computing. One of the major handicaps in realizing UbiComp lies in the missing integrity between different devices. Having an open system to transfer applications, lowers that handicap and takes away the importance, of which device you are using at a time.

This concept consists of four different components, similar to Balme et al. [2004]'s definition of *distributed, migratable and plastic user interfaces*:

1.2.1 Application Migration

Applications can move around freely, running independently on each device

The central goal of Nomadic Applications is to enable users to migrate their applications from one device to another, without being concerned about any low-level operations needed. Applications that arrive on another device should run independently of the original device, so no further connectivity is required after migration.

There are various approaches on migrating applications between devices. Possibilities include: Using only web-applications, which can easily be transferred, identifying an application's opened file and transferring it or extracting the information directly from the application. Approaches that do not work in this scenario as they require persistent connectivity include pixel-based replication or relaying user input.

1.2.2 UI Adaptation

Interfaces have to adapt to different device classes

Different UbiComp device classes have different requirements for their interfaces. A usual desktop computer for example provides enough screen estate to allow applications to display a wide range of control elements. Mobile devices however, are more limited in both their display and input capabilities and applications on them often only provide basic functionality. To preserve accessibility of applications when moving them between devices, their UIs have to adapt to the given device classes.

How to migrate applications in a way that opens up opportunities for adaptation?

This means that we have to migrate applications in way that gives the receiving end the opportunity to adapt the interface. Approaches like pixel-based replication do not provide this opportunity. We will approach this problem by replicating the state information of a given application. This way a receiving device can use that information to create an appropriate interface around it.

1.2.3 UI Distribution

Migrate only certain UI elements

Nomadic Applications also open up the realm of UI Distribution, that is moving only certain elements from applications to other devices to enable remote control with these elements. An example of this would be migrating toolbars from complex applications to other devices, so you have them at hand elsewhere while freeing up the main screen.

Not of concern here

While UI Distribution opens up opportunities for a whole

range of applications, I will focus my project on the other two aspects and not touch upon it any further in this thesis.

1.2.4 Nomadic Operations

Nomadic Operations enable Nomadic Applications to work

To make use of the possibilities of Nomadic Applications, a new set of interactions has to be employed that enables users to migrate or distribute their applications between different devices. These are further on called *Nomadic Operations*.

Examples of Nomadic Operations include, but are not necessarily limited to:

- Moving an application from your own to a target device
- Moving a UI element from your own to a target device
- Retrieving applications or UI elements from other devices
- Moving applications or UI elements between two other devices
- Relocating input from your own to another device

Only a subset of these possible interaction will be used in this project. I will focus on being able to move whole applications from your own to other devices.

One question that opens up at this point is, whether applications should be copied or moved, i.e. does the sender retain the application? Providing users only with the abilities to move applications seals away sample scenarios, like providing a co-worker with a copy of a document while continuing to work on the original yourself. Move operations also can be recreated by copying an application and closing the original. In conclusion the opportunity to copy an application is more flexible and for that reason it is the operation we will use further on.

1.3 Iterative User-Centered Design

Nielsen [1993] describes user-centered design or *usability engineering* as

> "a set of activities that ideally take place throughout the lifecycle of a product happening at the early stages before the user interface has even been designed."

Design, analyze, implement in each iteration step

Usability-centered design stands in contrast to the *waterfall model*, often used in software development, where a fully working product is developed first and any analysis and maintenance is performed afterwards. Instead user-centered design puts emphasis on evaluating any results that appear as early as possible. It is best described by the DIA cycle. The cycle can start and end at any step and after each step of designing the steps of implementing and analyzing follow. Any results from the analysis are then used to perform a new step of designing and start the cycle all over again.

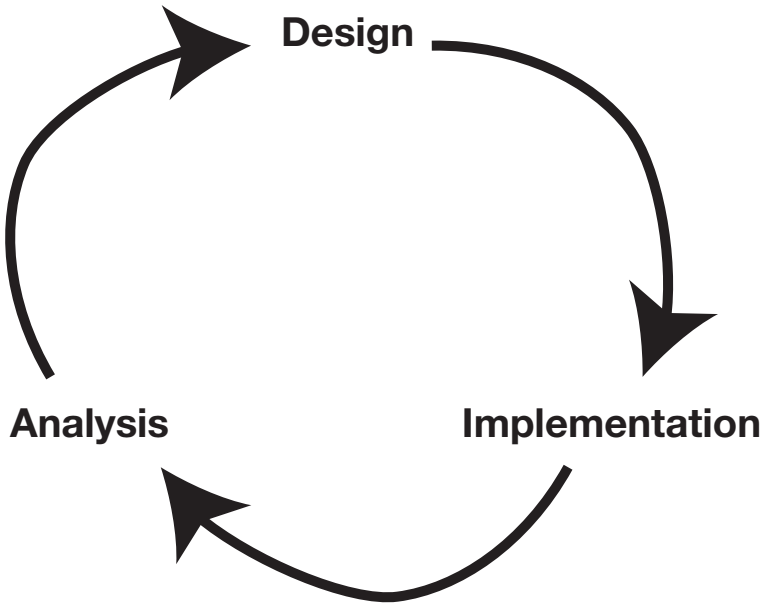

Figure 1.2: The DIA cycle. It can start and end at any point

All instances of design, implementation and analysis can vary in fidelity. Design and implementation can range from storyboards to fully working software products. Analysis can take the form of literature research, questionnaires, controlled experiments or long-term studies. After each completion of the cycle the fidelity should increase, but lacking results might also lead to stepping back and restarting at earlier stages.

I built prototypes and evaluated them

I used the methods of user-centered design throughout my project. After an initial step of literature research, I derived design requirements that I implemented in a paper prototype. After testing the prototype, I used the conclusions to change my designs and implement them in a software prototype, which I tested as well. After updating the software prototype according to the results I performed a longer study in a real-life scenario. In the end I evaluated that study and I will use it to give ideas and recommendations for a proceeding design step.

1.4 Chapter Overview

Chapter 1 The first chapter consists of key motivations behind this project. It also covers the concept of *Nomadic Applications* and the *user-centered* approach to system design.

Chapter 2 Chapter 2 presents three studies that show common practices and problems in today's work environments. These studies guide me in setting the goals for my system.

Chapter 3 In the third chapter, I will present other research projects that are connected to my work. It is divided into three different research types: *Augmented spaces* and their definition, *software solutions* that aim for similar goals and finally *interaction techniques* that might get used. I will explain relevance and distinction of these projects to my thesis.

Chapter 4 In chapter 4, I will define a set of design requirements the system should fulfill. Decisions, on which device classes should get support and the ways in which they will be supported, are drawn. I will also decide on the specific approach to implement Nomadic Operations in my system.

Chapter 5 In chapter 5, I will present my first work, a paper prototype designed with the preceding requirements in mind. This prototype will provide a starting point for all following steps. It also gets evaluated in a user study and conclusions are drawn from this.

Chapter 6 In the sixth chapter, I present the software prototype that I designed and implemented, following the paper prototype. The implementations of the three applications that it consists of and the underlying framework all those applications use get presented. I will also show the results of another user study in which I evaluated my applications.

Chapter 7 A two-week study in a real-world scenario with the created software prototype was conducted. The process of the study and the results drawn from interviews will be presented in chapter 7.

Chapter 8 In the final chapter, I will summarize my project and present its contribution. I will also present steps I would suggest for any future development of this project.

Chapter 2

Personal Workspaces

Before any work on a system that supports users in their personal work, can be begun, we first need to take a look at the way at how this work is performed. We need to identify common practices, problems that occur with them and ways in rectifying these problems using the concept of Nomadic Applications.

In this chapter I will present three studies that address such practices, namely *usage of multiple devices*, *taking information scraps* and *usage of whiteboards*. Each of these studies shows a number of problems people encounter in their everyday work. I will propose ways of dealing with these problems and derive components for a system to achieve these ways. In the end I will give *usage scenarios* that shows possibilities of employing such a system.

2.1 Multiple Devices in Users's Environments

To have a basis on which we can create Nomadic Applications, we have to first understand how and why users make use of multiple computing devices. Dearman and Pierce [2008] studied this behavior with 27 participants in extensive interviews. Figure 2.1 shows that people use an aver-

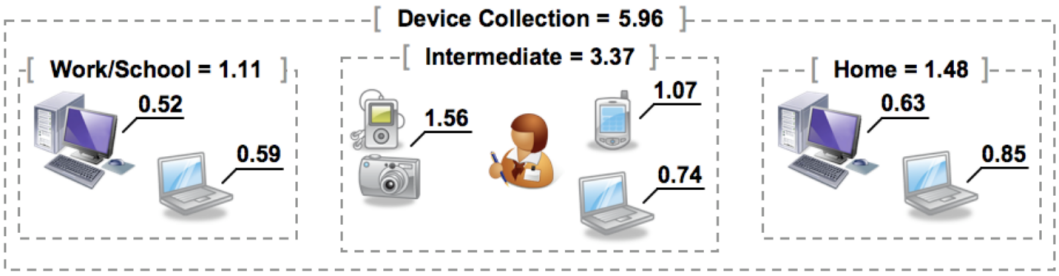

Figure 2.1: An average user's device collection, distributed by context of use. Note that this result also includes devices like digital cameras, feature phones, etc. with no real computing functionality.

age of 5.96 devices (4.4 if you count out cameras, feature phones, etc.) on a regular basis.

Reported reasons for using multiple devices are manifold:

Form factor that a device provides

- Different devices provide different *form factors*, resulting from their physical design and interaction modes, that are more or less appropriate for certain tasks. A tablet PC for example, might have a better screen for reading and can be used in a more comfortable position than a big laptop. On the other hand it lacks the input capabilities to perform more complex tasks.

Portability of a device

- The form factor also influences the *portability* of a device. While possibly limited in other ways, portable devices allow the user to choose the setting in which they work on them.

Completion time

- The perceived *completion time* it takes for one task can vary between devices. This can be a result of differing computing powers, but can also materialize through different configurations between devices. For short tasks a simple factor like startup time can also influence the user's decision on which device he wants to use in a given scenario.

Separation of work and home

- Some users use multiple devices to separate their *work and private activities*. This behavior can be forced, as they are provided with a computer to use at work, or entirely voluntarily, because this separation is deemed desirable. This happens more often in industry work than in academics.

- Different devices can be supplied with different operating systems and software. Users' preferences on which particular software to use for any given task can vary and lead to device switching as well.

Software and OS differences

- *Special purpose devices* tend to support only one activity, but suit this activity rather well, whereas *general purpose devices* have a variety of roles, but no specialization. Users often prefer to use special purpose devices for specific tasks.

Special and general purpose devices

- Computers tend to have a limited lifecycle and are replaced by new ones on a regular basis. This can lead to cases, where users can not use their new computer with their old work for compatibility problems and they do now have two computers that they use on a regular basis.

Transitioning between old and new device

Participants reported a number of problems they regularly encounter, when working with multiple devices. The most reported one is the difficulty of exchanging information between them. Most of these exchanges are still file-based and users employ different strategies:

Information exchange seen as biggest problem

- *Portable media* like USB sticks or external hard drives
- *Emailing* files to oneself
- *Network shares*
- *External services* that either act as a file server or are specialized in storing specific types of files and granting online access to them

Most people see these only as patchwork solutions that come with their own set of problems. Local file-based solutions are often cumbersome and they need users to remember exactly what they put where. Online services on the other hand raise concerns about privacy and do need persistent connectivity to work. Automatic file synchronization mechanisms were deemed too limited, as they are usually tied to two specific devices and do not offer anything more than "all-or-nothing"-synchronization.

No solution feels ideal

File-based exchanges do not transfer interaction histories

Another reported problem with file-based exchange was the lack of interaction histories, as these are always tied to one device. A number of users reported that they use certain applications on on device only (sometimes even remote-controlling them) to keep all application specific histories and settings in one place

2.1.1 Support Multiple Device Use with Nomadic Applications

Nomadic Applications ease information exchange

Bringing Nomadic Applications into the equation can help ease users in their daily interactions with multiple devices. As Nomadic Applications enable users to directly move their applications between computers carrying all necessary information with them, they do not need to be concerned about any low-level file operations. Additionally all operations are performed only in the required networks, so in most cases no internet connectivity is needed and privacy should be of no concern.

They also open up opportunities for task-splitting

Nomadic Applications also open up more opportunities of assigning different roles in one task to different computers, another thing that has been reported as troubling. With today's methods, using multiple computers for one task requires a lot of setup beforehand. Exchanges during task execution should be kept to a minimum, as the overhead for transfer operations usually outweighs the benefits.

Being able to directly move applications between devices reduces that overhead and should enable users to take advantage of using multiple devices in unison more easily. The study gives an example of a user that uses two dedicated computers for coding and testing purposes. In this given case, Nomadic Applications would ease the task of copying the application to the testing device after each step of compiling.

2.2 Information Scraps

Information Scraps: Information that doesn't get stored in PIM Tools designed for it

Another interesting aspect about people's work environments, is the way in which they store their personal information. Bernstein et al. [2008] studied how people employ information scraps in their everyday life and work. They define information scraps as pieces of information that eludes the PIM tools designed to manage it. Examples of this behavior would include:

- A calendar entry written on a post-it note
- Contact information send to one's own e-mail
- A To-Do list saved as a text file on one's computer's desktop

Information types often include To-Do lists, meeting notes and contact information, but lots of other types can be found

Different aspects of information scraps were studied. One of them was the type of information that is typically stored. The types most commonly found were To-Do lists, contact information and How-Tos. But while these types were predominant among all participants, it also appeared that there is a long tail of information types that only a small subset of users (often only one of them) used to store, so in the end it is close to impossible to generalize which types of information need to be supported for a given user.

A variety of storage systems is used, although four of them are predominant

Another aspect that was investigated upon, was the tools, that people use to store their information with. Similarly to the information types, it was found that there was a subset of tools (Notebooks, E-Mail, Post-It notes and text files) that was popular among all participants. Still a lot of them used their own - oftentimes very obscure - methods of storage.

People prefer to use their own notation and storage system

One identified reason, why information scraps are omnipresent in a lot of people's lives, is the fact that PIM tools often enforce their kind of notation and storage upon the user. A lot of them however like to use their own storage systems and shorthands. While this often causes their notes to be impossible to understand by everyone else, they can use what they are accustomed to, which helps both in efficiency and understanding. Also, as was said before, PIM

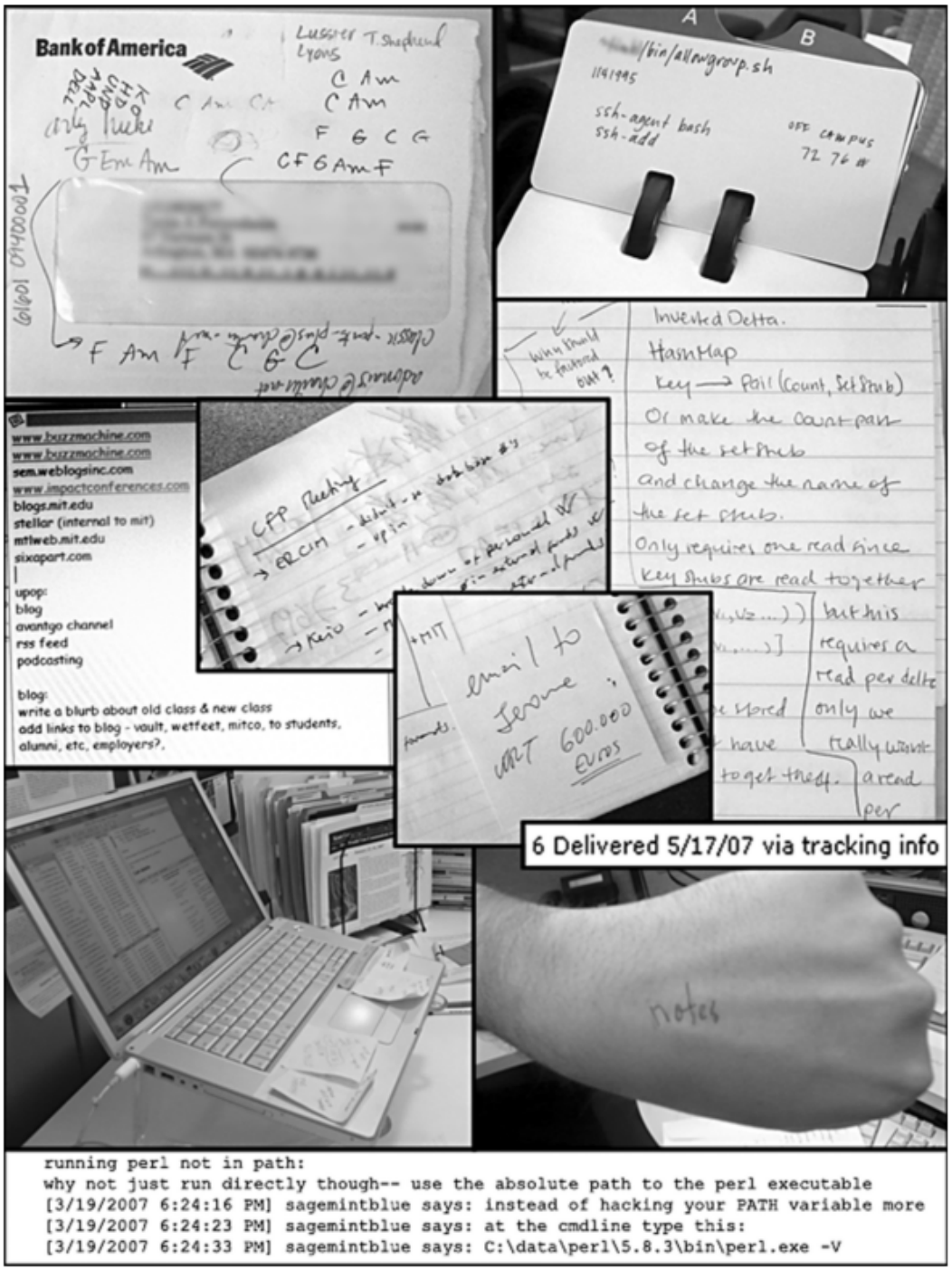

Figure 2.2: Some examples of information scraps. Taken from Bernstein et al. [2008]

systems are usually not built to support the manifold types of information, people want to store.

PIM tools often take more effort to use then less-fitting alternatives

Another reason why people tend to forego the use of their PIM tools is the fact that they often find it's more effort to use these tools in comparison to their homemade alternatives. To quote one test subject:

> "If it takes three clicks to get it down, it's easier to e-mail."

User controls visibility of information

There is one last important reason, why people like to store their notes the way they do. Using their own tools they can ensure that the information is where they need it when it is needed, while using other tools can often mean that they simply forget about these things.

2.2.1 Supporting information scraps with Nomadic Applications

Nomadic Applications enable direct storage of information without limitations of PIM tools

As we can see, the practice of taking information scraps is often used to overcome the lack of accessibility and availability in modern PIM tools. With Nomadic Application, users do not need to take any relevant information out of its associated application. Instead, they can store the application where it needs to be. This also eliminates the need for any notations, as the relevant data is available itself at all times.

If we take the example of users noting the URLs of websites, they want to visit later on, we can directly see the advantages: Instead of having to copy the URL manually and taking it down manually, the user can simply move the browser with the opened webpage to another device, where he gets reminded of it later on.

To support this behavior as well as possible, we need to keep two things in mind:

- People use a wide array of storage types. To be able to transition them into using Nomadic Application, we

have to account for different device classes on which they might want to store their information

- People store different types of information. We need to enable them to store as much of those as possible with Nomadic Applications as well.

2.3 Whiteboards

Whiteboards are a common tool in all kinds of work environments. Tang et al. [2009] studied how people use their physical whiteboards both in single-user and collaborative environments with a special focus on switching between related tasks.

135 participants took part in a survey on their whiteboard usage, of which 11 ”heavy” users gave an additional one-hour interview in front of their most important whiteboard.

	Individual	Collaborative
Synchronous	• structuring • visualizing • large writing surface 15%	• brainstorming • collaborative design 30%
Asynchronous	• To-Do lists • notes • reminders • references 61%	• collaborative task list • project plan • schedule 26%

Table 2.1: Groups of typical whiteboard tasks

Usage principles can be grouped

Typical whiteboard tasks were grouped into four different categories (see 2.1). These were separated by either individual or collaborative usage or synchronous and asynchronous usage. Additionally a percentage of whiteboards that contain remnants of the specific type was given.

Combination of different modes possible

Scenarios that some of their participants presented showed cases in which the usage of the board combined several of the categories. To give one example, one user would use the board mostly for sketching out his ideas (an individual synchronous activity), but keep them on the board as a reminder of unresolved problems (making it an asynchronous activity). Later on he uses the board synchronously again, in that he continues to work on his sketches.

2.3.1 Combining Nomadic Applications with Whiteboards

We can support users even further, if we provide them with large displays that act as *digital whiteboards*. These would act as an enhancement to traditional whiteboards.

Nomadic applications support asynchronous activities

Giving users the opportunity to migrate applications to these displays, broadens the options for asynchronous activities. It would allow them to store their information in application form, instead of only having it available visually. This would also ease the transition of information to the board, as it takes away the need to manually copy it.

Appropriate whiteboard applications support synchronous activities

To support synchronous activities, a digital whiteboard at least needs to provide all interactions available on a traditional whiteboard. These can also be enhanced by the opportunities a computer can bring to a formerly pen-based system.

Combining Nomadic Applications with these traditional whiteboard interactions ensures that possible combinations of whiteboard activities are still possible as well. These can also take advantage of Nomadic Applications, as any work performed on the board can easily be transferred to other systems.

2.3.2 Core Principles of Whiteboards

Core principles of whiteboard usage can be identified

From questionnaire and interviews a number of core principles on how people tend to use their whiteboards can be derived, which should be kept in mind when designing a new system:

Whiteboard as a container

- Whiteboards are expected to be *persistent* and *visually available* containers for information. Interpretation of its contents are dependent on the contextual location of the board.

Meaning through representation

- Information on whiteboards can convey meaning beyond itself, e.g. though the use of spatial organization. Users were shown to create meaningful applications with just having primitives available.

Flexible representations enable appropriation

- If we support the use of primitives, as stated in the previous point, instead of powerful applications, users have an easier time of adapting their methods of representing information to the new technology.

Location and context of use

- The location of a whiteboard conveys meaning in itself. Affordances and functionality required can vary, depending on the context of the board.

2.4 Usage Scenarios

Assuming we had a system, that allows for seamless application transfer between different computers and digital whiteboards, there is a number of sample scenarios that show how such a system could be employed in a user's daily routine:

Board as temporary application storage

- The board could act as a storage for applications that should be kept "out of sight but not out of mind" for the time being. A user could move applications that he needs later on to the board. Later, when he needs it again, he can fetch it back to his computer.

- A digital whiteboard could act as project planning tool. Any part of a project that needs to be worked on gets stored as an application. If the user decides to do work upon a certain part, he gets the application from the board. When he is finished he puts the result back on. This scenario is imaginable in a collaborative setting as well.

  Board as project planning tool

  In this example, users can combine the opportunities of spatial arrangement with having the application data directly at hand. According to the principals in 2.3.2 the meaning of the applications available can be increased beyond the data they are representing.

- A single-person task can temporarily become collaborative, for example when a user wants to talk about a problem he has, with his co-workers. In this case the board can be used to have a presentation tool for your work readily at hand, without needing any setup.

  Board as ad-hoc presentation tool

With these scenarios in mind, we can now begin to work on creating such a system.

Chapter 3

Related work

First I will show *Augmented Spaces*, large rooms with a lot of special hardware that try to realize the ideas of Ubiquitous Computing in confined spaces. Next, I will present a number of projects that try to realize similar ideas without special hardware and only sophisticated *Software Solutions*. In the last section, we take a look at two *Interaction Techniques* that could be used for Nomadic Operations.

In this chapter we will see that there are a number of approaches that relate to the core principles of Nomadic Applications and Nomadic Operations. However none of these approaches provide the opportunity of having UI migration and UI adaptation for existing applications in an open environment.

3.1 Augmented Spaces

Augmented Spaces provide rich user interactions but are usually tied to a fixed location

Augmented spaces try to realize the ideas of Ubiquitous Computing in settings that are usually confined to a designated room. These spaces often provide a number of specialized devices, like large displays, tabletops and projectors to enhance collaborative work in them. They also employ methods for users to use their personal devices in this augmented setting.

The tight integration of devices in an augmented space, often through the use of special network infrastructures, allows for accessible exchange of information, similar to the Nomadic Applications concept. These methods however are oftentimes limited to the space as they need the special hardware and software setup that the rooms provides.

3.1.1 i-Land

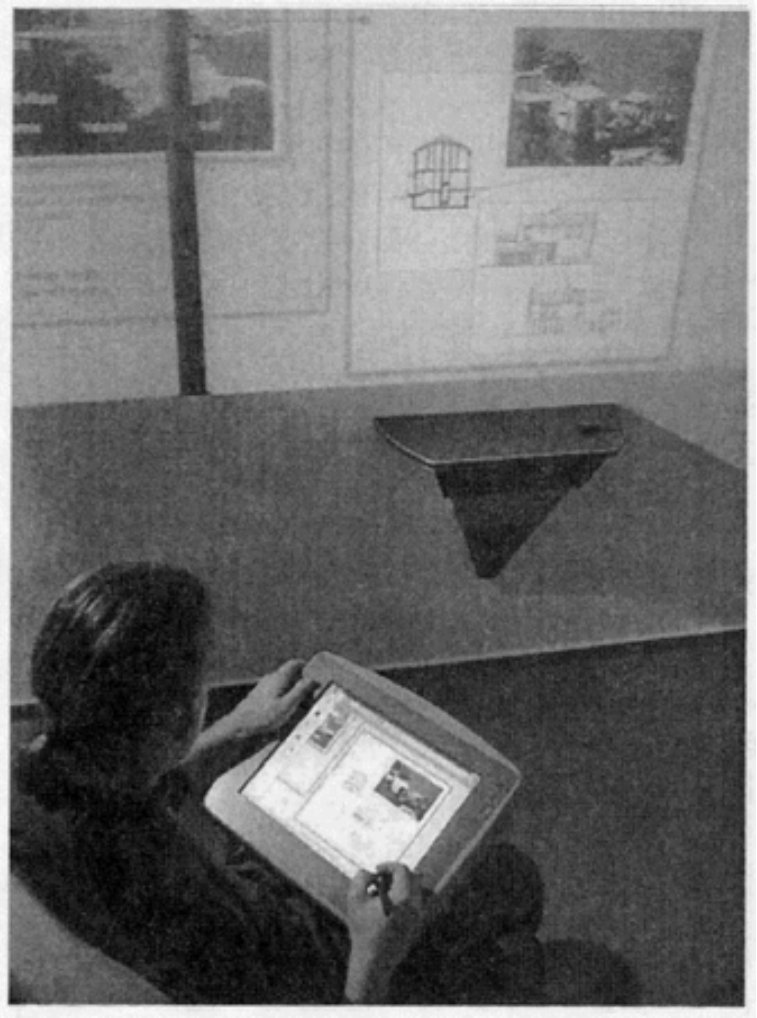

Figure 3.1: Remote Annotations in i-Land

i-Land by Streitz et al. [1999] is one of the older efforts to

create an augmented space, in which you can freely move information around.

Large touch-sensitive wall and table for public display

As with most augmented spaces, i-Land is centered around a large display wall. The so-called *DynaWall* is touch-sensitive and provides enough space for two persons to work simultaneously on it. It also has mechanisms that allow both users to share their work across the sides of the wall.

The room has another collaborative component in the *InteracTable*, a large touch-sensitive display table. It supports cooperation by allowing users to move and rotate objects across the whole table.

Chairs with inbuilt computers allow for private work or public sharing

Users that don't work on any of the public devices can use the *CommChairs*; mobile armchairs that have slate displays built in their armrests. Visitors can work on these displays in private or use them to remote control the public displays or share information on them. Another variant of the chair has a docking facility instead of a slate display, so people can use their own laptops on them.

Digital information can be linked to physical objects

i-Land uses a technique called *Passage* for linking digital information to physical objects, so that you can transport them through the room. Once you place an object on a device's *bridge* it gets recognized (via weight detection) and you can link any information to it. Placing the object on another device's *bridge* enables you to retrieve your information without any need of knowing, where it is located in the virtual realm.

Highly tied to specialized software

i-Land is mostly realized through a special software called *BEACH*. While this enables the devices to be tightly integrated with one another, it also means that it is very problematic to use out-of-the-box software with it. Without this software people will have a hard time integrating these concepts into their day-to-day work. This is one of the key points I want to address by using Nomadic Applications.

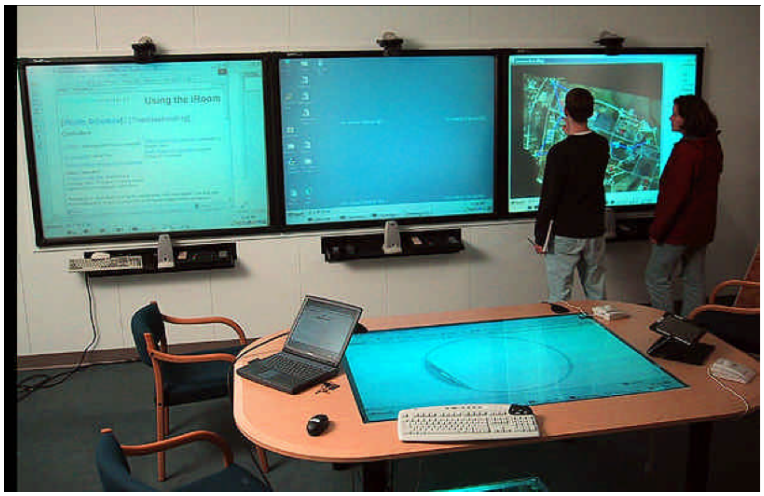

Figure 3.2: The Stanford iRoom with 3 large displays and a tabletop. Taken from Johanson et al. [2002]

3.1.2 iRoom

The iRoom provides a workspace specifically tailored to cross-device interaction

Another augmented space that supports the transition of applications between devices is the iRoom by Johanson et al. [2002]. It is designed to allow simple exchange of data and control between a wide number of applications.

iROS Meta-Operating System handles cross device interaction

To allow applications to make use of iRoom's modalities, a Meta-Operating System called *iROS* was created. It consists of 3 parts. The *Event Heap* stores and forwards events between devices. The *Data Heap* stores data independent of the applications that are supposed to handle it, while also allowing format conversion. *iCrafter* advertises services through the Event Heap. Once a service is selected, iCrafter passes the best applicable interface to the given device. The interface and service then communicate via the Event Heap.

Decoupling for flexibility and robustness

A big focus of iRoom lies on decoupling. Decoupling the system from specific devices enables deployment in larger variety of spaces. Decoupling applications from the operating system allows for development of independent software. And decoupling applications from one another makes errors on one device not affect the others, meaning

Figure 3.3: The wall display of NiCE. Taken from Haller et al. [2010]

the stability of the whole system increases.

As a result, the iRoom provides a highly flexible system, that is not fixed to specific devices as tightly as other augmented spaces. Still, it mainly supports employment in fixed rooms and requires a lot of specialized software.

3.1.3 The NiCE Discussion Room

In an effort to combine the qualities of paper, whiteboards and digital media Haller et al. [2010] created the NiCE Discussion Room.

A large display wall supports both digital and traditional pens

The centerpiece of the room is the NiCE whiteboard, which consists of three layers. The topmost one is of acrylic laminate and it serves as surface on which you can draw with traditional whiteboard markers. The second one is an Anoto-foil and it allows for Anoto digital ink pens (see

Haller et al. [2006]) to be tracked as well as serving as a projection surface to display the tracked data. The last layer is metallic to enable magnetic pins to stick to the wall.

Laptops can project their screen onto the wall

The room supports participants in using their personal laptops as well. Computers can be connected via VGA and the whole screen, or just a chosen portion of it, can be projected anywhere on the wall.

Paper in combination with Anoto pens allow for traditional writing that can be stored digitally

Users that prefer to use paper to take their notes can also do so. The writing or sketching on the paper is tracked with Anoto pens and a digital representation gets stored in the system. All these representations can be displayed on the wall.

User study showed positive results

A user study conducted in the NiCE discussion room showed that participants were able to harness the different interactions, which the system provides. Users were most content with the ease of sharing different kinds of data between devices. On the other hand users found it hard to be aware of the other participants, as the wall is very big. Also the overlays used to organize content were deemed too complicated to learn in such a short session

Very elaborate system that requires a lot of setup

NiCE shows very well how you can integrate laptops, paper and a whiteboard to enhance the experience of collaborative settings. To do this however, it needs a lot of very specialized tools and a room that is setup to support this system in the long-term, something you often don't have when you deal with personal workspaces.

3.1.4 ARIS

ARIS is a window manager that provides app relocation for augmented spaces

In contrast to the previous publications, ARIS by Biehl and Bailey [2004] does not actually focus on the space itself but instead concentrates on building an interface to support application relocation in a given augmented space.

ARIS is a window manager that builds upon the Gaia OS by Román et al. [2002], a middleware operating system designed to support augmented spaces. Its main goal is to provide an accessible way to move any application from

Figure 3.4: An example of interaction with ARIS. An application (red) gets relocated from a laptop to one of the wall displays. Taken from Biehl and Bailey [2004]

any device in the room (not necessarily the one you are working on) to any other device in the room. A secondary goal is to also be able to relocate the input from the device you are working on to any other device in the room.

An iconic map interface has proven to be the most accessible way to relocate applications

The developers of ARIS performed a lot of studies, on how to achieve these goals in the most accessible way. In the end ARIS provides an iconic map that can be invoked from any running application. This map shows a top-down overview of the space with all its devices. Every running application is displayed as a rectangle on its particular device. If a user wants to move an application around, he can drag it from one device to another one. To relocate his input, he leaves the mouse cursor over another device, before the map window closes.

The iconic map interface is too tied to a pre-defined setup

While their user test proves the iconic map interface to be superior to its alternatives, it his hardly applicable for any kind of ad-hoc solutions. For the map interface to work as intended, the whole setup of the room has to be known beforehand so you can have an accurate visual representation of it. As I try to avoid being tied to any specific environments this kind of interaction will not be employed.

Additionally ARIS is very heavily tied to the Gaia OS. While that allows for this kind of tight integration in a fixed environment it also takes away from the opportunity to test it any real-life environment.

3.2 Software Solutions

Augmented spaces are not the only approach to realize environments, in which applications are no longer tied to one device. As iRoom (3.1.2) already demonstrated, a lot of problems with cross-device interaction can be solved just with capable software.

Data access is starting to get less device-dependent

Applications or extensions to existing applications have been developed that allow for transitioning to other devices, without needing a special hardware setup. In the advent of cloud-computing, accessing your data from any possible device is slowly becoming a given possibility. Commercial applications like Evernote[1] or Remember the Milk[2] enable users to take their notes and access them anywhere they want.

On-the-fly UI migration, on the other hand, is still not very common. The following research projects realized things that are very similar to the idea of Nomadic Applications

3.2.1 Multibrowsing

Making the browser a nomadic application

Johanson et al. [2001] realized a first attempt at making the browser a nomadic application in what they called *Multibrowsing*. In their system clients have the possibility of opening links on other displays or pulling browser windows from other devices onto your own display.

Enhanced clients can send webpages to and pull them of targets

The system knows two kinds of clients. *Targets* allow webpages to be redirected to them or be pulled off of them. *Enhanced Clients* have the ability to redirect pages to Targets as well as pulling off their frontmost displayed page.

Targets provide a service called the *butler service*, which communicates with Internet Explorer via its COM/OLE API. Each butler service provides a unique name to all enhanced clients. Two methods can be called from outside.

[1] http://www.evernote.com
[2] http://www.rememberthemilk.com

Figure 3.5: The context menu of a multibrowsing-enhanced browser. Taken from Johanson et al. [2001]

One to open a local browser with a given URL. The other to get the URL of the frontmost browser window.

Enhanced clients are standard Internet Explorer applications enhanced with the MB2Go plugin. It adds two new option to the hyperlink context menu as seen in in figure 3.5. These new options allow the client to either send a link to a target or get the displayed webpage of a target.

Special links support non-enhanced browsers as well

Additionally their system supports *multibrowse fat-links*. These links cause their gateway-servlet to forward an event to a specified target. Using this links, multibrowsing operations can be accessed from browsers that don't have the necessary plugin installed as well.

Approach heavily tied to browsing

While multibrowsing supports something I want to address in my project as well, i.e. being able to move web content between different devices, their approach is heavily tied to the browser (especially with the inclusion of *multibrowse fat-links*), whereas I want to be flexible to support a wider range of applications.

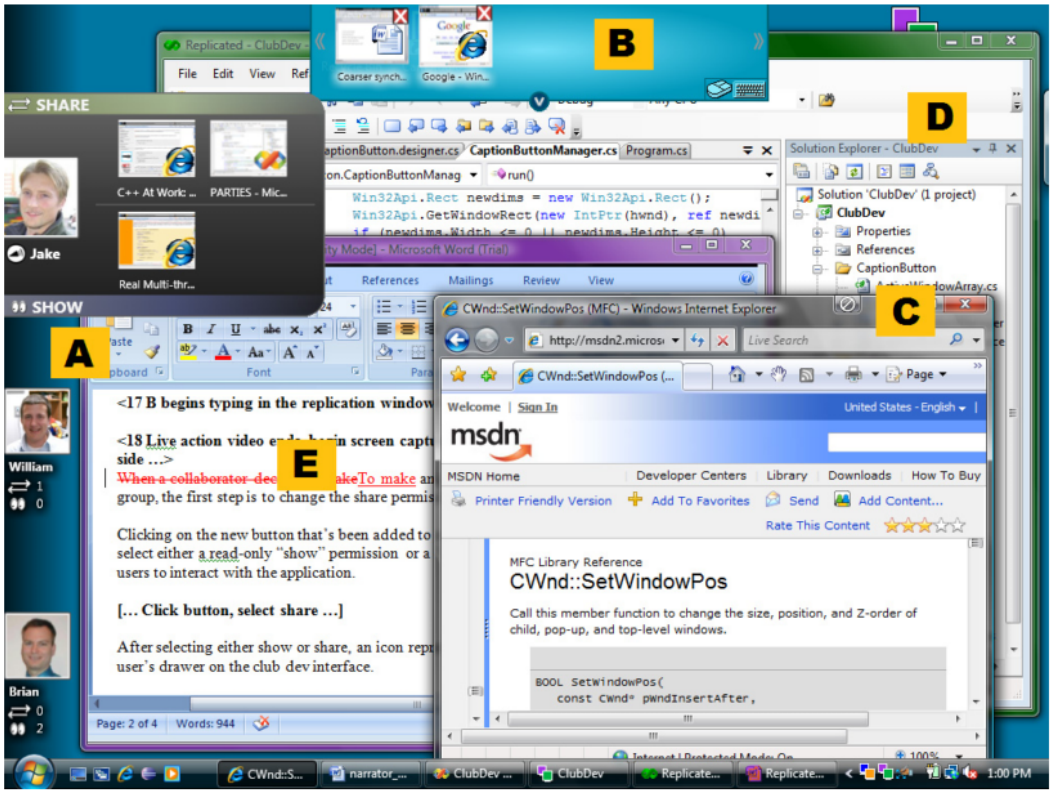

Figure 3.6: A Screenshot of the Impromptu Interface showing (A) the collaborator bar, (B) the shared screen dock, (C) the control level of an application, (D) a replicated window in share mode and (E) a replicated window in show mode. Taken from Johanson et al. [2001]

3.2.2 Impromptu

Impromptu by Biehl et al. [2008] was developed as a framework to support collaboration in multiple display environments with a special focus on software development work environments.

Users can fetch shared windows of collaborators

Users of the system can make application windows available to the group in varying control levels (view-only or modify). Once a user decides to share a window it appears besides his icon in everyone's *collaborator bar*. Other users can drag the window onto their desktop to work with it.

Shared screen to be accessed by anyone

In addition to the collaborator bar, Impromptu provides a shared screen that can be accessed by every member

through the shared screen dock on their desktop. Local input can also be redirected to the shared screen.

Implemented with window replication

Impromptu works through window replication. With this model, applications keep running on their device and the pixels of its window get replicated on other devices. Ownership of an application never transfers to any other member of the group.

Study showed good use of the system

A field study conducted with two software development teams over the course of three weeks showed positive results. Developers used Impromptus features to solve complicated problems together and found it easy to transition between individual and collaborative work. A quantitive analysis showed, that users preferred to use the features to share their work to other members, but rarely used the provided support for input redirection.

Window replication not suitable for Nomadic Applications

While Impromptu shows well, how UI migration can enhance work environments, its approach to implementing it is hardly suitable for my project. Window replication is completely dependent on the device originally running the application being turned on and available. An application that is truly nomadic needs to run independently of its original device. Replication takes away scenarios, in which users do want to store applications as an independent copy.

3.2.3 CHAMELEON-RT

CHAMELEON-RT by Balme et al. [2004] is an architecture reference model that facilitates the development of distributed, migratable and plastic (DMP) user interfaces, an approach similar to Nomadic Applications.

Three defining qualities of DMP interfaces

They define the terms as follows:

- *UI Distribution* means that an interface uses foreign interaction resources. UI Distribution appears in varying granularities, so for example Pick' n Drop (3.3.1) is distributed at the workspace level, while Impromptu (3.2.2) is distributed at the pixel level.

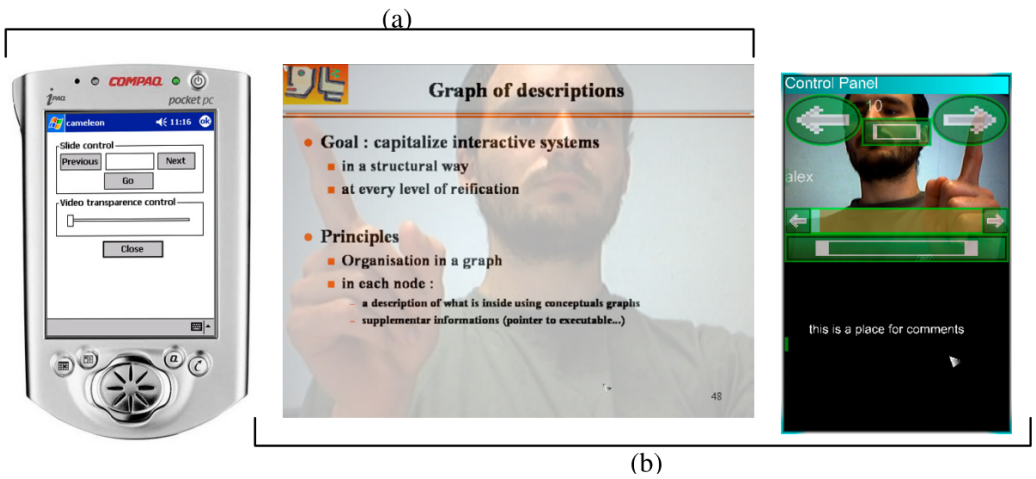

Figure 3.7: The distributed user interface of CamNote. (a) shows the interface distributed to PC and PocketPC, while (b) shows it using only a PC. Taken from Balme et al. [2004]

- A UI that supports *Migration* transfers itself to different interaction resources. It can be *total*, where a whole application moves *partial*, in which case only certain parts of an application are transferred. It is also differentiated in the dimension *static* and *dynamic*, depending on whether the migrations happens between sessions or on-the-fly.
- A UI being *plastic* means that it preserves its usability properties when distribution or migration happens, similar to the term *UI Adaption* I coined in 1.2.

CamNote shows application of these principles

These principles are demonstrated with the software *CamNote*, a tool for slide presentation. CamNote's main feature is the ability to have its control panel switched from PC to a PocketPC, if it is available in the cluster. Both controls provide different UIs fitted to their device (like the addition of a video preview of the presenter on the PC), demonstrating UI Plasticity in action.

Multi-Layer model to build DMP UIs

CAMELEON-RT itself now provides a multi-layered reference model to support the development of applications that fulfill these principles. The key-layer of CAMELEON-RT is its *DMP Middleware Layer*. In it, *observers* in conjunction with the *situation synthesizer* provide information about the

current situation (e.g. platform, place and user) a system is in. The *evolution engine* creates a predefined reaction to that situation and passes it along to the *configurator* to execute it and build an appropriate UI.

Try to approach the problem in a different way

While CAMELEON-RT covers a lot of aspects about UI distribution, migration and adaptation and their definitions should always be kept in mind, its approach differs from our approach to Nomadic Applications in two main ways:

- Its main focus lies on the creation of new applications that support their principles. Nomadic Applications should support existing applications as much as possible, so users can be kept in their comfort zone instead of having an additional burden to handle.
- While covering a lot of ground, it also builds up a fairly complex model that needs a considerable amount things to be taken into considerations, when developing new applications. However simple methods, like appropriate state exchanges already offer opportunities for UI migration and adaptation.

3.3 Interaction Techniques

Cross-device interaction is not feasible with today's input methods

In most of today's computing environments user input is limited to one device. This makes sense as long as all output is also connected to that one computer. Once applications become nomadic however, input has to extend to other devices as well.

Cross-device interaction is a common problem in all research on Ubiquitous Computing. It is defined as the interaction that enables operations to span multiple devices. Techniques have been developed to move objects around in UbiComp environments and these should be applicable to Nomadic Applications as well.

In theory, Nomadic Applications should support any number of interaction techniques to realize Nomadic Operations. Our focus lies in developing the enabling technol-

ogy and any of the presented techniques can make use that technology.

3.3.1 Pick-and-Drop

Figure 3.8: Pick-and-Drop in action. Taken from Rekimoto and Saitoh [1999]

Pick-and-Drop overlays network copies with physical interaction

Rekimoto [1997] present Pick-and-Drop as an interaction technique for moving objects between different devices. With Pick-and-Drop cross-device interaction gets moved to a physical layer, so users do not need to care about the underlying workings of the software.

Pick-and-Drop helps users ease into the task of file copying between devices. To copy an object from one device to another, a user has to select a given object with a pen, lift the pen and then put it down on another device. The object then gets copied to the target.

Realized with unique pen IDs that objects get bound to

On the software level, Pick-and-Drop realizes its operations, by assigning a unique ID to each pen. Once a pen "picks up" an object, the object gets linked to the ID on a network server called *Pen Manager*. Then, when the pen

touches another display, a network copy operation is initialized by the *Pen Manager*.

Pick 'n Drop has seen field use in more than one project. NiCE (see 3.1.3) uses it for moving content over large distances, while i-Land (see 3.1.1) uses a modified version, which they called *take-and-put*, that does not require any pens. In a similar fashion, I plan on using a pen-less version of this technique to realize exchange between devices.

Proven successful in augmented spaces

3.3.2 Hyperdragging

Another interaction technique proposed by Rekimoto and Saitoh [1999] is Hyperdragging. It provides a spatially continuous surface, through which digital objects can be moved.

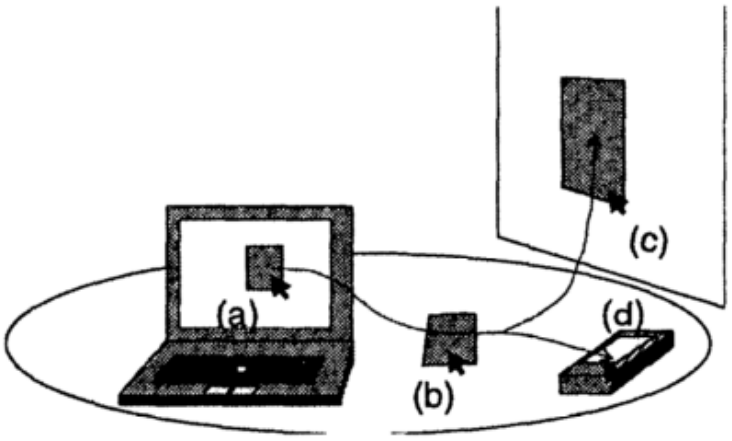

Figure 3.9: An outline on Hyperdragging. An object gets dragged from a laptop (a) to a tabletop (b) to a wall display (c) and a VCR tape (d). Taken from Rekimoto and Saitoh [1999]

In a hyperdragging-enabled environment, a laptop gets recognized, once it is put on one of the surfaces. A user can now move any object from the laptop to the surface by dragging it to the edge of the screen. An object on the surface can further be dragged around and put onto other de-

Dragging objects from your computer to surfaces and vice-versa

vices that are connected to the surface. Furthermore, information can be linked to physical objects, similarly to the *Passage* technique as described in 3.1.1.

Object recognition is necessary

Hyperdragging is realized with visual markers on notebooks. Once a notebook's position and orientation is known, information can correctly be migrated from it to the surface. In general the *InfoTable* that is used in the system uses two cameras for motion recognition and object recognition.

Hyperdragging is well suited for Nomadic Applications, but needs too much setup

Hyperdragging is a powerful technique that can be used for Nomadic Applications. Dragging objects from one screen to the next is a very natural interaction that should be easy to learn for any potential user. It should especially be kept in mind, when one would want to employ Nomadic Applications in an environment with Tabletop-displays. However, I will not be providing a continuous surface, which makes the technique less appropriate. Hyperdragging also needs to provide location awareness for the devices in the system, a problem that is not trivial to solve and would make deploying the system on an ad-hoc basis a lot more problematic.

Chapter 4

Requirements

Explorative Approach: Define own requirements, evaluate afterwards

Before I can start to work on any system itself, there needs to be a set of requirements that the system needs to satisfy. In this project I aim to give users new opportunities to support them in their personal work, rather than solve a particular problem that existed beforehand. Deriving any requirements from questionnaires or inquiries is therefore likely to not yield any meaningful results. Instead, I will define a set of requirements derived from the results of studying people's behavior in their workspaces in 2. I will evaluate these requirements with a paper prototype at a later point in time.

In this chapter I will first describe which types of *UbiComp Device Classes* I aim to support with my system and in what ways they will be integrated. I will also describe an approach to design the *Nomadic Operations* that are necessary for such a system.

4.1 Device Classes

Build system for three device classes

With this system I aim to support three kinds of device classes: *Desktop computers*, which in this case means any kind of computer that people use to do their personal work on and also includes laptops and notebooks. *Large Displays* that are connected to standard computers as well, but

are bigger than the displays people use for their routine work. *Mobile Devices* include all kinds of devices that one can carry around on a regular basis, like smartphones and tablet computers.

4.1.1 Desktop Computer

Do not disturb users' usual work environment

Nomadic Applications should support users in their usual workflow and not pose an additional burden that they have to put up with. As such, I aim for users being able to operate with their everyday applications in their accustomed ways, while making them nomadic at the same time.

Limit interaction to small status bar icon

To achieve this, a desktop application should be limited to a status bar icon. From this icon the nomadic operations can be invoked and no further functionality has to be learned by the user.

4.1.2 Large Display

Focus of design

In this project I will specifically support users, by giving them a "digital whiteboard" on which they can store their applications. In turn my focus of design will lie in designing a system to provide such a board, by using a large display connected to a regular computer.

Standard desktop computer metaphors do not work for large displays

The UI metaphors used in todays operating systems do not suit large displays well. They rely a lot on fullscreen windows and being able to switch between tasks, while a usual whiteboard displays a lot of information side-by-side.

Hide Operating System

The first step to turn a computer with a large display into a digital whiteboard therefore should be to hide as much of the underlying operating system as possible. Both accessibility and screen estate are premium resources. Hiding system information and controls means, there is a lot less of possible interaction a user has to worry about and more space to actually use.

Application support

In a similar way to the operating system, applications have to be adapted to the new affordances a large display provides.

Light-weight versions of standard applications

Support of the applications, a user is accustomed to, is as important as it is for the desktop computer. A lot of functionality that modern applications provide is not needed here, however. The display mainly serves as a storage container, while actual complex work is supposed to be done on the desktop computer. So an excess amount of controls again waste screen estate and take away accessibility.

Provide only necessary interactions

If we can provide stripped down versions of important applications that only provide the most necessary functionality, we can limit the cognitive overhead for users of the display, while also maximizing the effective use of the screen estate.

Allow strong primitives

While a wide array of supported applications obviously gives users a larger repertoire to work with (and in our context, more things they can move away from the desktop), strong primitives are most important to enable users to express themselves, as noted in 2.3.

Combination of text-editing and sketching enables strong expression

If we can support both, text-editing and sketching directly on the board in an accessible way, we should have those strong primitives at hand.

Browser support enables use of variety of web applications

Additionally internet browsers are the tools that people use most across all their devices (see Dearman and Pierce [2008]). Browser support should fill a big need for potential users. With the ever-increasing popularity of web applications, this step gives support to a lot of applications without the need to implement them.

Wider support preferable but not mandatory at this point

Of course this does not mean that a wider range of supported applications is not something to aim for. But for now I want to focus on these three, so I can ensure they are being integrated well in this concept of digital whiteboard.

Window modes

Balance visibility of one application and storage of multiple applications

As noted earlier, screen estate is one of the most important resources in a system like this and to compare to a normal whiteboard we need to support storing multiple applications side by side. 2.3.2 stated, how we can convey additional meaning on a board through arrangement. However, we cannot forget that in order to guarantee enough visibility, applications might take a big part of the display. To find a tradeoff between these two contrasting forces, windows should be able to switch between two modes:

Default window for interaction

In their default mode they act just like a normal window in a computer system. You would use this, if you want to interact with the application or just be sure that you can recognize everything in it.

Miniaturized windows for storage

Miniaturized windows are considerably smaller than the default ones and hide all application controls. Additionally they use a zoomed-out viewport, so that, while you probably will not be able to recognize every detail in the application anymore, you still have a general overview over it. You would use this mode, if you want to put aside an application for later use, while still seeing that it is actually there.

Mode switching has to be accessible

Switching between these modes should be kept as accessible as possible. Miniaturized windows in particular have no interaction on their own, so users should not be able to do anything with them but move them around and maximize them again.

4.1.3 Mobile Devices

Use mobile device as a temporary application storage

The third device class that will be supported are mobile devices. I will not make a distinction between different types of mobile devices here, as in this system they are only supposed to act as a temporary storage of your applications. You can use this storage, so nomadic applications are no longer confined to the proximity of your devices. With a mobile device you can ”carry” your applications around

and use them again in all places that support nomadic applications.

Forego any real interaction with applications

Since a storage like this only needs a very limited amount of possible operations, I will forego having any kind of real interaction with any application on your mobile device for now. Interaction with nomadic applications on mobile devices will be restricted to just the basic set of nomadic operations this way.

4.2 Nomadic Operations

Nomadic Operations are the cornerstone of accessibility

Since Nomadic Operations are the heart of this system, the accessibility of the whole system hugely depends on the accessibility of these operations. If users get overburdened by the task of application migration, they are likely to abandon the whole system in favor of a more traditional approach. Therefore they have to be easily learnable and always accessible.

Unknown number of devices in an unknown space

Another factor that has to go into the design of these operations, is the fact that I want to support ad-hoc deployment of Nomadic Applications as much as possible. For that reason we can not just assume that we have to move applications between only three devices. In a workspace like a large office, we have to account for an unknown number of devices that could all support Nomadic Applications.

Balance accessibility and unambiguity

Again there are two conflicting forces for which a trade-off has to be found: On the one hand Nomadic Operations should not require an excess amount of interaction, on the other hand there might be ambiguities about the target of moved applications that have to be resolved.

Device list prone to overload and malicious intent

One solution would be to provide a list of devices to which you can send an application. This approach bears two potential problems however:

- In a scenario with a lot of supported devices, such a list gets easily overloaded and users will have to

spend too much time looking for their target. Even in smaller scenarios the mapping of device names to actual devices might not always be fully clear (see Biehl and Bailey [2006])

- This might raise privacy concerns. In an environment, where users can send their applications to every other device, we can not guarantee that no one uses this system with malicious intent, e.g. spamming another user

Both of these concerns lead me to believe that using a device list to invoke Nomadic Operations is not the ideal way to go.

Design interactions similar to Pick-and-Drop

Instead I opt to use a metaphor similar to Pick-and-Drop, in which you have to select both, a source application and a target device, before an actual Nomadic Operation happens. Since I aim to use this without any special physical objects, the meaning of the words ”pick” and ”drop” might not be obvious to users in this context. Instead I will use the more familiar wording of ”send” and ”receive” and see how users react to it.

Specifically these will be the two steps, needed to perform a migration operation:

- Pushing a ”send”-button on the source device causes the current foreground application to get marked for sending.
- Pushing a ”receive”-button on the target device will spawn a copy of the application that got marked for sending.

Chapter 5

Paper Prototype

Paper prototyping has been an invaluable in UI design for a very long time. Sefelin et al. [2003] have shown that the quality and quantity of results from a paper prototype based study does not differ substantially from those of a low-fidelity software prototype study.

Paper prototypes can have a number of different shapes and sizes

In this context, paper prototypes are representations of a software UI that are drawn on one or more pieces of paper. They might offer no interaction themselves or can be controlled by a person to allow for state changes and thus users interacting with them. Interaction can range from a flipbook, where you jump to different pages, depending on what you do, to very complex prototypes assembled with post-its where you can move around all of its parts.

They hold several advantages over software prototypes

Paper prototypes are easy to create, as you do not need anything more then a pen and paper. This rapid creation enables you to realize multiple ideas and encourages you to throw out bad ones without any regret. The crudeness of the prototype also encourages users to critique on your work rather than holding back out of respect.

For my system I want to develop a simple paper prototype to evaluate some of the requirements, I defined in 4 and get an idea of good ways to implement some of the necessary interactions. It will will also provide a starting point to design the look & feel of my system.

In this chapter I will first show the *designs* I prototyped for the different device classes. After that I will run a small user study to *evaluate* them and draw *conclusions* for my software prototype.

5.1 Device Design

Elaborate Prototype for board, simple one for desktop

To start things off, I built two prototypes. A very elaborate one for the large display (which I will call *board* from now on), since it is the focus of my design and a simpler one for the desktop, since it does not provide any more interaction than the Nomadic Operations.

The mobile device is supposed to have no functionality besides Nomadic Operations. I can evaluate these interactions with the other two prototypes. So I decided to skip on building a third prototype, to keep tests later on more concise.

Minimalistic for both prototypes

For both prototypes I chose to be minimalistic and just use white carton and a black felt pen. As the system consists of very few interactions at this point, I wanted to keep everything as clean as possible to not overload participants in the user study.

5.1.1 Board

Prototype consists of board itself and its windows

The board prototype consists of two parts: The actual board with a control bar for annotations and Nomadic Operations and a set of example applications that might rest on the board.

Windows

One window for each type of application

For each of the applications I defined in 4.1.2 I built two example windows: One in default and one in miniaturized mode.

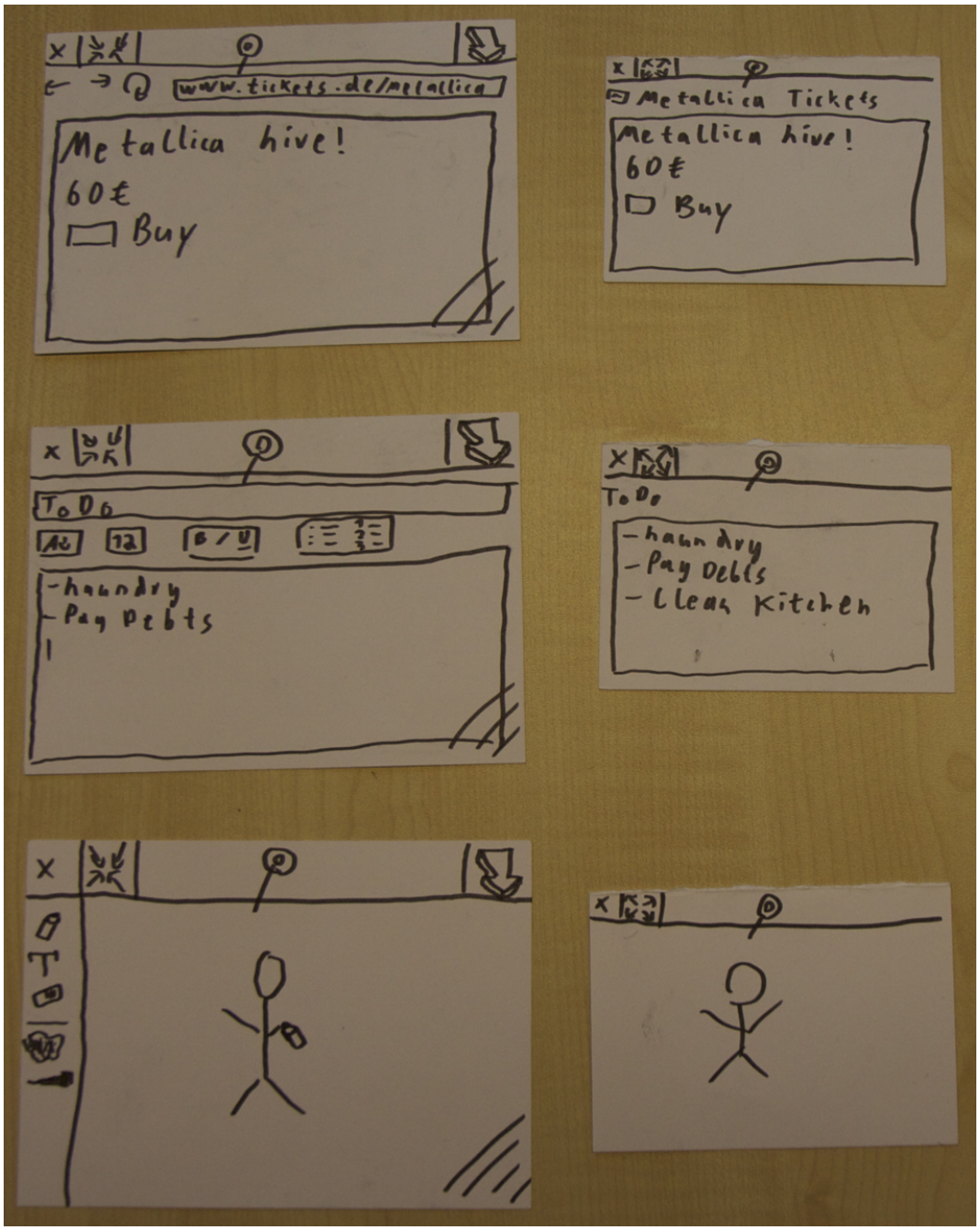

Figure 5.1: 3 sample windows, from top to bottom: Browser, text editor, sketching tool. Left column shows them in their normal state, while right column shows the miniaturized versions without any controls

In detail the following windows resulted:

- A browser window showing a page to order concert tickets. The default variant shows the standard back, forward and refresh controls.
- A text editor window containing a To-Do list with two sample items. The default variant shows a couple of controls for text formatting.
- A sketching window with a stickfigure. The default variant shows a toolbar with a small set of drawing tools.

Three variants for testing

In turn, each of these sextuples existed in three different variants to test different interactions later on:

- *No controls for sending and miniaturizing*

  Focusing a window would automatically mark the window for sending in this variant. Miniaturizing/maximizing the windows would be invoked by double tapping.
- *Send button, no controls for miniaturizing*

  A window would only be marked for sending, when the button is pressed. Miniaturizing works as in the previous variant.
- *Send button, miniaturize button*

  No hidden controls, everything gets invoked with buttons. This is the variant shown in 5.1.

All windows shared a big close button, a pin that can be used to move the window around the screen and a handler for resizing in the lower right corner.

Board Controls

In addition to the different windows the board has some controls itself. All of them are found in a toolbar on the lefthand side.

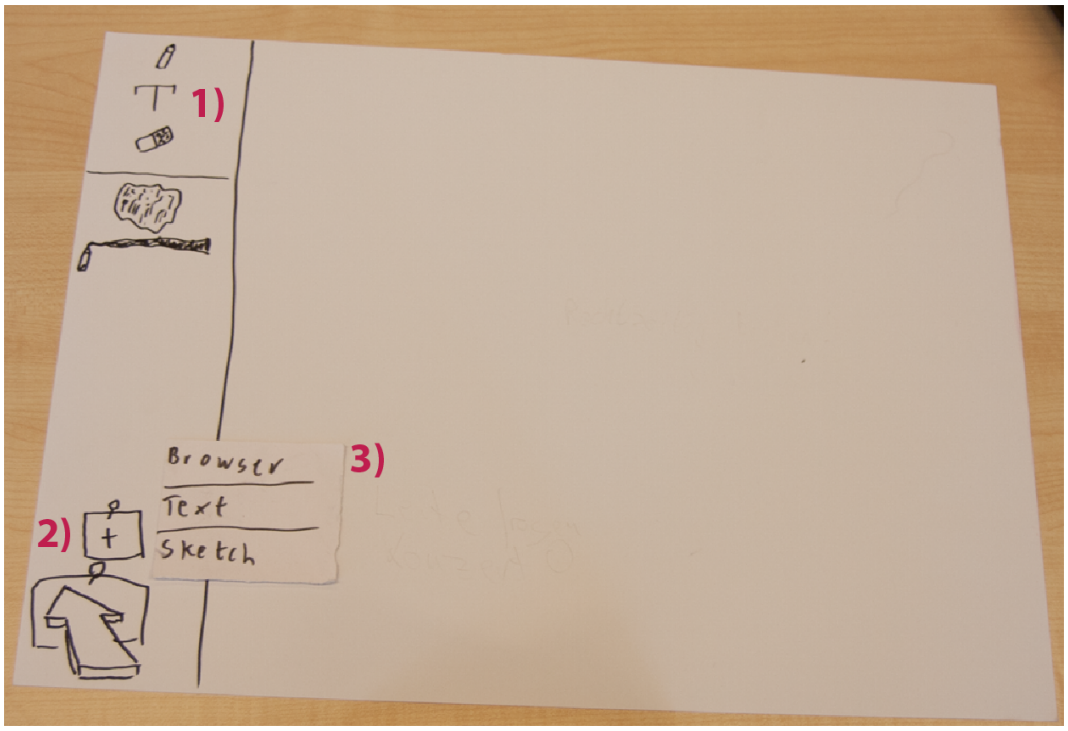

Figure 5.2: A representation of the digital whiteboard. 1) Controls for annotating in the background. 2) Controls for receiving and opening new applications. 3) The popup that appears when the "new application" button was pressed

Standard drawing tools for annotating

The top part of this toolbar provides a group of standard drawing tools, namely a pen, a text tool, an eraser, a color picker and a pen size slider. All of these can be used to make annotations in the background behind any running applications.

Nomadic Operations on the bottom

On the very bottom of the toolbar lies the button to invoke a receive-operation. Above it is a button to spawn a new application. While this is not a Nomadic Operation both can logically be grouped since they will both create a new window on the board.

5.1.2 Desktop

Standard Mac interface with additional status bar item

The desktop part of the prototype shows a representation of a typical Mac OS X interface. The only foreground application is a browser window showing a page to order con-

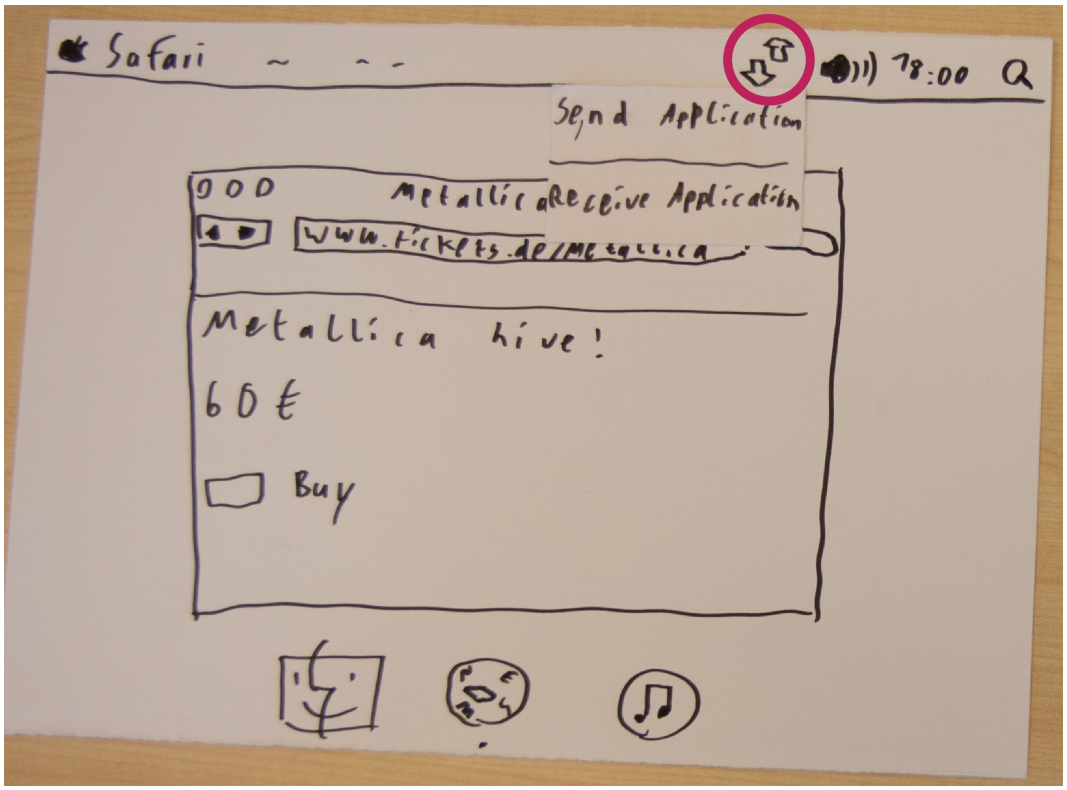

Figure 5.3: An updated version of the desktop interface. This representation shows a browser window and a popup that offers the nomadic operations. Encircled: The status bar item that is used to invoke nomadic operations

cert tickets, like the browser application on the board. The major addition lies in the status bar, where Nomadic Operations are called.

Two variants for status bar

As with the paper windows, I created two variants for this prototype as well to test some interactions:

- *Two arrows in different directions*

  Both operations, send and receive, have their own status bar item, with which they are called.

- *One icon with two arrows*

  Both operations are called with the same icon. Pressing it reveals a pop-up menu from which they can be chosen

5.2 A small user test

With the created prototype, I conducted a small user study. Since I was mostly interested in finding out peoples' reactions to the given set of operations i decided to forego any quantitative analysis and ran the test on a purely informal basis.

5.2.1 Tasks

To keep the tests short and concise, tasks were held simple so users would be able to solve them within 5-10 minutes.

More specifically the following tasks were given:

- *You just found out that one of your favorite band is giving a show in the near future. You have a webpage open, where you could buy tickets, but before you do that you would like to know, if any of your friends liked to join you. Please put the webpage onto the board as a reminder to yourself*

  This task examines, how obvious the send/receive interaction appears to the users. I expected them to not recognize the key icon on the desktop representation on their own. Unless they discovered it really quickly, I would point them to the icon, but leave the rest of the task up to them.

- *You just realized that you still have to clean the kitchen. Please open up the to-do list that is already on the board and add this task, miniaturizing it again afterwards.*

  This task examines how natural interacting with two window modes feels. For this task i actively switched variants of the windows in between tests.

- *You are bored and feel like drawing something. Please open up a new sketch and draw onto it.*

  Now that the user should be somewhat familiar with the prototype, I wanted to know if they recognize the icon to invoke a new application without any pointers.

- *Some days have passed and you do now know, which of your friends would like to have tickets as well. As you have your credit card information stored on your desktop computer, you need to bring back the webpage, before you can order those tickets*

  As the send/receive interaction from desktop to board is different than the one from board to desktop, this task was given. Here I examined, if the previous interactions are enough to help the user figure out the way back by themselves.

This small set of tasks covers all major interactions the prototype provides at this point. If something was still unclear to the participant after solving these tasks, I offered them to play around some more, so that they can discover how the system reacts to their input.

5.2.2 Testing Process

Purely informal testing

I ran the test with 6 different participants on an informal basis. All of them were students from the fields of Computer Science, Mechanical Engineering and Geology. As there was no data to quantitatively analyze I would base all of my findings on the participants' comments

Think-Aloud test with me manipulating the system

During the test I would take care of all operations a real computer system would provide, in particular taking care of the availability, position and size of the windows (and pop-up boxes). Drawing operations could be performed with a pencil if participants wished to do so. During the whole test participants were encouraged to think-aloud so I can associate their thought process with the actual decisions they made.

5.2.3 Findings

Despite the simplicity of the prototype and the briefness of the test, some veritable conclusions could already be drawn regarding the user interactions of the system:

- All but one of the participants took some time for the first task. Once they pushed the send button they expected an immediate reaction on the other device, namely a new window appearing and I needed to tell them, that another interaction is necessary. While this seemed to pose a problem, in the last task all but one of them (see next point) showed that they made the mental connection of how the interaction works and solved it without any pointers

  The two-step interaction of sending/receiving is not obvious but easily learnable

- The one user that tried the variant without any send-button had even more problems in the first task. Since there was no counterpart to the receive-button, its functionality was largely unclear, as well as the fact that there was even anything to receive on that device. In the end I had to point directly to the button for him to solve it. Even then the last task still posed problems, since there was no indication of which application would be received.

  As this idea proved to be very unsuccessful at this point, I decided to scrap it altogether and not test it any further

  A button for sending is absolutely necessary for this kind of interaction

- On the desktop prototype with two separate icons, participants commented that the direction of the arrows does not convey the actual kind of operation to be executed to them and they would just push a random one. The variant with only one icon did omit this problem. There was only one button to press and all options were labeled afterwards, so this proved to be a lot clearer.

  The board prototype did not have this problem, since the send-button is attached to a particular application while the receive-button is global. This lead users to quickly make the necessary mental connection despite the arrow directions not being telling for them.

  Arrow direction is not a good indicator for sending or receiving

- During the second task, all users that were posed with the window variant that had no buttons for miniaturizing/maximizing would double tap a miniaturized window as their first or second interaction without having been given any pointers.

  Later on, when they were to miniaturize it again they immediately chose to go for the double tap again,

  Double tapping to change mode seems to be natural enough

since it already yielded success.

From this I conclude that I can omit any extra buttons for this operation.

Having the same controls in the annotation bar and the sketching window is confusing

- When asked to create a picture and draw on it, two users would (correctly) push the "+"-button to create a new sketch. To actually draw on it they selected the pen from the annotation-bar.

  Another user would simply push the pen and expect a new sketch to appear immediately.

  It is probably wiser to rethink these two applications again and maybe combine them, so there are not two toolbars that essentially do the same thing at first glance.

5.3 Conclusions

Keep send/receive interaction but make it more obvious

The major conclusion I draw from this prototype testing is that the two-step send/receive interaction, while probably not optimal, at least seems to be accessible enough. However, while doing this I have to focus on making it as obvious as possible, so other users do not experience the same confusion as some of the participants.

Minimalistic approach seems to work. Maybe cut down even more

Cutting down on possible interactions helped the participants understand the existing ones for themselves. I will try to reduce the number of unnecessary control elements even more.

Chapter 6

Software Prototype

In the preceding two chapters I defined a set of design requirements, designed a first prototype based upon them and evaluated the prototype in a controlled test.

In this chapter, I will now present my final software prototype I designed and implemented, based on the previous work.

I will first give reasons, why I chose to develop a *vertical prototype*. Next I will present the three applications, developed for the different device classes: The desktop application *NomadicDesktop*, the board application *NomadicBoard* and the mobile application *NomadicPasteboard*. I will also explain the underlying *NomadicApps Framework* that all three applications use at detail. Lastly, I will present another *user test* I performed to check if any general usability problems still persisted.

6.1 Horizontal vs Vertical Prototyping

Limit either features or functionality

In his book *Usability Engineering* Nielsen [1993] defined two dimensions of prototyping:

- *Horizontal prototypes* offer all features a finished prod-

uct is supposed to have, but are limited in their functionality. In software prototyping this usually resorts to UI mockups , in which all kinds of possible interactions can be performed but none of these interactions will cause any real actions.

Horizontal prototypes can be used to evaluate an interface as a whole in a controlled environment and evaluate the interactions an interface provides.

- *Vertical prototypes* are limited in their features, but everything they put on display is fully functional. This usually results in a fully working software that might miss key features to be perceived as a full product.

  Vertical prototypes can be used for in-depth testing of a system in a realistic environment

Vertical prototype allows for evaluation of users' adoption

The system I propose in this thesis is not one that any given user will immediately put to use, but rather one that disappears in the background and will be used when it is needed, similar to a traditional whiteboard. For this reason I want to focus my evaluation on users' adoption of the system.

While a horizontal prototype would allow me to evaluate the interactions in my system even further, the amount of possible interactions is very low to begin with and I believe another usability study will not yield considerably more results, then the paper prototype test i already carried out.

A vertical prototype will allow me to evaluate it in a place where it actually matters: A user's personal workspaces. It is also still possible to do a controlled test with a vertical prototype, to see if any major interaction problems still persist. Therefore I chose to implement three fully working prototypes that realize the design requirements I put up beforehand.

6.2 The Desktop Application

Implemented with Mac OS X and Cocoa

The desktop application called "NomadicDesktop" is designed for computers that run Mac OS X version 10.6 or

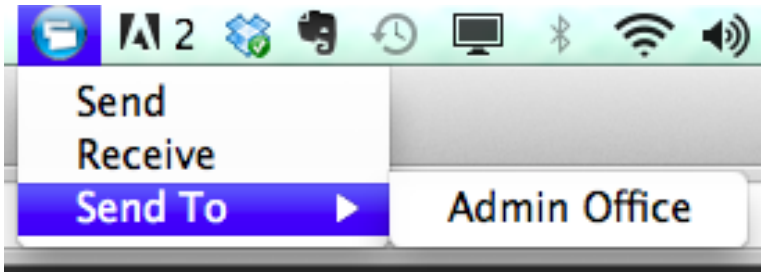

Figure 6.1: The NomadicDesktop status bar application and its pop-up menu

higher. It is implemented using Apple's Cocoa[1] framework and Objective-C as its programming language.

Design similar to the paper prototype

The design of only showing one status bar icon that provides the Nomadic Operations was kept. As can be seen in figure 6.1 a new option to send applications to devices directly, without needing a second interaction step, was added for testing purposes. The device list only shows boards, which are at least semi-public, to minimize privacy concerns.

Application Delegate provides most of the applications functionality

NomadicDesktop is a very simple application, with most of its functionality being implemented in its application delegate, a class that Cocoa makes available in every other part of an application. The application delegate will send and receive application states using the NomadicApps Framework, which I will explain in great detail in 6.5.

Retrieve foreground application and send state

Sending an application state works by using Cocoa's `NSWorkspace` class to detect the foreground application. If it is a supported application NomadicDesktop will extract its state (as described in 6.2.1) and then send it. If not it will show an error.

Launch application with received state

When an application state is received, its type gets examined and the appropriate application gets launched (if necessary) and called with the state.

[1] http://developer.apple.com/technologies/mac/cocoa.html

6.2.1 Application Support

App support mostly relies on AppleScript

As discussed in 4.1.2 I aim to natively support a browser, a text editor and a drawing application in my prototype. Since there needs to be a way to extract the current state from the application or insert one in, I chose to use applications that have appropriate *AppleScript* support, with the exception of the sketching application, where I implemented extraction myself. Using AppleScript I have a convenient and reliable way of communicating with running applications. If possible, I used *Script Bridge* to implement the communication. While it offers the same functionality as AppleScript, it facilitates better readable code.

Browser

Use URLs as application state

For browser support it is mostly sufficient to simply use the URL of the currently open site as a suitable application state. While this does not ensure, all of a browser's state information gets transferred (cookies, post data or entries in website forms will not transfer), most websites choose to encode all important information in the address.

Safari and Chrome get support. Firefox does not

Both Apple Safari[2] and Google Chrome[3] provide AppleScript functions to get the URL of their current foreground website. Firefox provides this support only in outdated versions. This supports two of the three most important browser on Mac OS, one of which is pre-installed on any system, so it should suffice.

Launch websites with Cocoa

Since Cocoa's `NSWorkspace` provides a function to launch a URL with the browser that is set to system default, there is no need to use AppleScript for received URL states.

Text Editor

Stick to plain text

For text editing, I chose to just store the plain text informa-

[2] http://www.apple.com/safari/
[3] http://www.google.com/chrome

tion as an application state. As stated in 5.3 I want to cut down on unnecessary features. For tasks like taking notes or writing To-Do lists there is no need for rich-text formatting.

TextEdit as the application of choice

Apple's TextEdit[4] provides AppleScript support to get the text of the current foreground document as well as opening a new document and inserting text in it. Like Safari it is also pre-installed on any Mac, which makes it the editor of choice here.

Sketching

Use Sketch It!

Sketching support works differently. The application supported here is *Sketch It!* by Fraikin [2011]. As it is built with Nomadic Applications support in mind it already provides all necessary state information.

Remove nomadic functionality from the application

However the original Sketch It! uses its own controllers and UI elements for Nomadic Operations, while I intend to have this operations invoked only by the NomadicDesktop status bar icon. Therefore I used a modified version that has no own nomadic functionality.

Communicate via ThoMo Networking

Instead NomadicDesktop and Sketch It! now communicate via ThoMo Networking[5] . Both applications connect with a unique identifier. NomadicDesktop can now request a sketch with a network call, in which case Sketch It! sends the state information of the currently opened sketch back. Similarly NomadicDesktop can send a received sketch state to Sketch It!

Generic NSDocument-based Applications

AppleScript supports extraction of generic documents as well

Using AppleScript we can also extract the state of any application that uses Apple's document based storing system (e.g. iWork, OmniOutliner), the only caveat being that the document has to be saved to hard drive in its current state.

[4] http://www.apple.com/de/macosx/apps/all.html#textedit
[5] http://hci.rwth-aachen.de/thomonet

Send and receive documents as files

When trying to send a state of any application that is not natively supported, NomadicDesktop will try to make an AppleScript call that delivers the file path of the currently opened document. If it yields any result, it will do one of two things:

- In case the document is stored in a bundle, this bundle will be compressed in a tar archive which then gets encoded as an `NSData` object and stored in the state.
- A single file gets encoded as an `NSData` object directly and stored in the state

In any case the application name, document name, document type, data type and a screenshot of the application window get stored alongside the document in the state.

Use Metadata to open documents again

When a document state gets received, NomadicDesktop can use the given information to unpack the document (if necessary), store it in a temporary folder and launch the appropriate application with that document.

This ensures support of a whole range of applications without actively building any methods to communicate with them.

6.2.2 Window Grabbing

Send screenshot so there is something to display in any given case

As mentioned, before any state that gets send out, a screenshot of the current application window gets stored in it. This ensures that any receiving device will have a representation of that state it can display, even if it does not provide any own functionality to open the document in that state.

CGWindow provides screenshots of all system windows

The CGWindow API that was introduced in Mac OS 10.4 provides the functionality to grab a shot of any system window. Two steps are necessary two perform the task.

- The number of the current foreground window has to be identified. `NSWorkspace` provides the process

serial number of the current foreground application while `CGWindowListCopyWindowInfo` provides a list of all system windows. All that needs to be done is finding the window with the right serial number from that list.

- `CGWindowListCreateImage` needs to be called using that window ID and the option to just grab a single window

The resulting image representation of the current foreground window can be encoded as an `NSData` object and gets stored in the state alongside all other information.

6.3 The Board Application

Designed for Macs with SMARTBoard

The board application called *NomadicBoard* is designed for computers, connected to a large display, that run Mac OS 10.6 or higher as well and also implemented using Cocoa and Objective-C. The interaction design is focused on displays that utilize a single-touch overlay by SMART Technologies[6] called *SMARTBoard*.

Simplistic applications like in the paper prototype

I follow the design principles I used to create the paper prototype and provide stripped down versions of regular applications, which are in turn rendered in windows, especially designed for use on a board with touch input.

Run in kiosk mode

Since the operating system should be hidden as much as possible, NomadicBoard runs in kiosk mode. It uses Carbon's `SetSystemUIMode` method to hide Mac OS' dock and menu bar from the user.

There are three parts to the application:

- The WhiteBoardWindow framework that is responsible for the special windows, used in the software.
- The Document and Controller classes, which handle and display the different types of applications.

[6] http://www.smarttech.com/

- The Application Delegate that takes care of sending and receiving state data.

6.3.1 WhiteBoardWindow Framework

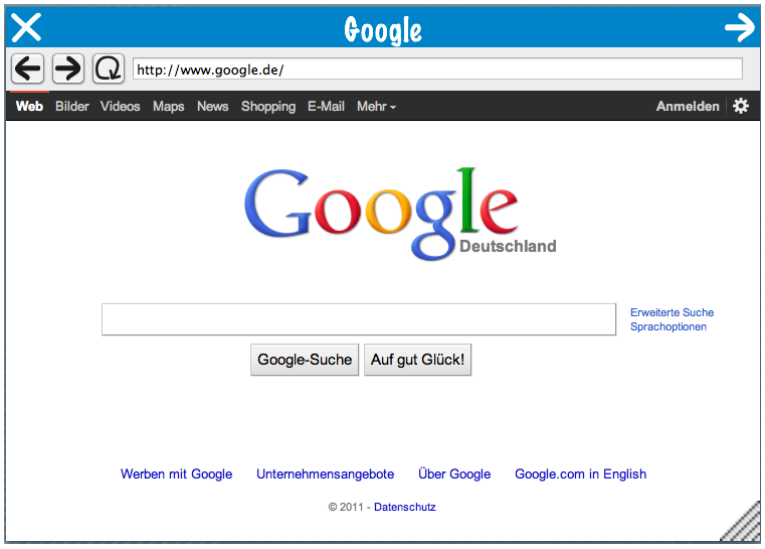

Figure 6.2: A sample WhiteBoardWindow. Top left: The close button, top right: The send button, bottom right: The resize handler

Provide board appropriate windows

The WhiteBoardWindow (from now on called WBW) Framework provides a new class of windows that essentially take care of two tasks:

- Serve window controls that are more appropriate both for Nomadic Applications and for touch-sensitive boards than the ones of the default Mac OS X windows.
- Take care of the switching between the two window modes.

Design windows similar to notes. Account for board affordances

Design wise I went for a more crude look, which conveys the note-like use of applications in this software well. It also reminds potential users that this software is just a prototype. All controls and text are displayed larger than in a

usual OS window, as a whiteboard is not necessarily destined to be as close in proximity as a computer screen. This guarantees visibility even when situated further apart from the board.

The Framework consists of 3 classes:

- `WhiteBoardWindow` a subclass of `NSWindow` that takes care of displaying the content, window movement operations and window mode switches.
- `WhiteBoardTitleBarView` a subclass of `NSView` that takes care of displaying the titlebar with its buttons and the title.
- `ResizeHandlerView` a subclass of `NSView` that takes care of displaying the resize handler and all its operations.

Additionally WhiteBoardWindow includes the protocol `WhiteBoardWindowDelegate`, which is needed for mode switches.

Windows and their controls

Borderless windows need usual functionality reimplimented

`WhiteBoardWindow` uses `NSBorderlessWindowMask` which delivers a window without any own controls or titlebar. While this allows me to use my own titlebar with its controls, it also means that all other functionality of usual windows, like moving them around and resizing them has to be reimplemented again.

Window movement is implemented by performing a hitbox test on the titlebar, when a mouse button is pressed. If it is hit, movement gets started and all further dragging of the mouse will lead the window to reposition itself.

Resizing is implemented in `ResizeHandlerView`. It displays an oversized version of the usual Mac OS X window resize handlers. When it receives a `MouseDown` event it

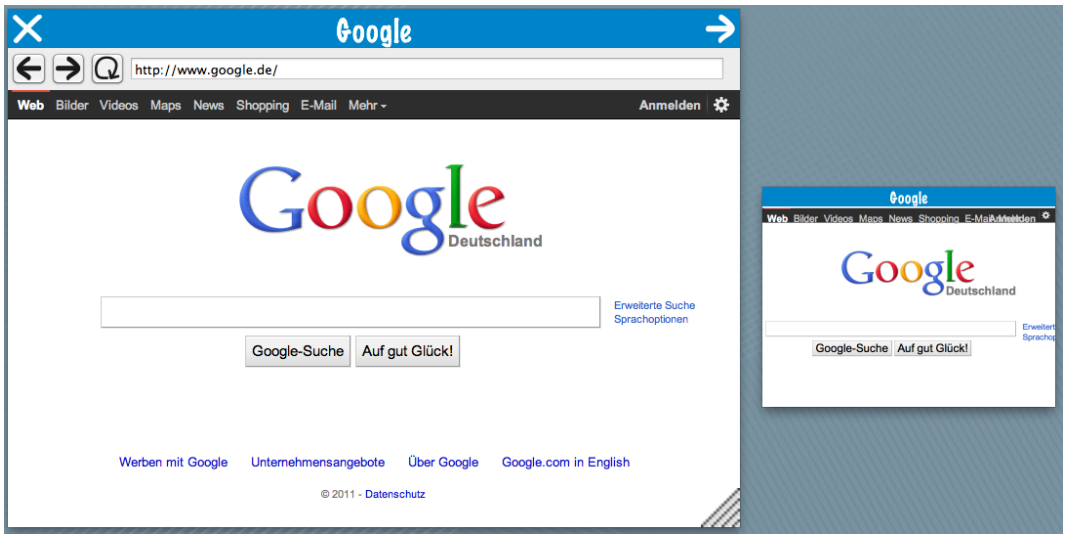

Figure 6.3: Comparison between a browser window in default and miniaturized mode

starts a resize operation and mouse drags will lead the view to resize its window accordingly.

The window solves correctly displaying both the titlebar and its content by enlarging itself by the size of the titlebar, as its `setContentView` method gets called. Both the titlebar view and the actual content will be added as subviews to the window's `contentView` property.

As decided after the paper prototype evaluation in 5.3, we can strip down the window controls to just the close and send functionality, both of which will find their place in the titlebar. Both of them are buttons. The close button is connected to the window's `performClose` action, while the send button is connected to the application delegate's `send` action.

Window Modes

Double-click scales window down

Once `WhiteBoardWindow` detects a double-click on the titlebar it calls the `toggleMinimize` action. In this action the window gets minimized to a fixed size of 300x225 pixels and the titlebar hides its controls. Additionally a *CoreAni-*

mation(CA) transformation on the windows `contentView` is performed. This transformation scales the contents of the window down by half their size.

Block all events on miniaturized windows

As a side effect of this transformation, mouse input on the window's view does not get redirected to the appropriate targets anymore. This does not pose a problem, however, as there should be no interactions with miniaturized windows in the first place. As a result, all events except for mouse-clicks and mouse-drags get blocked from them. Dragging moves a miniaturized window around, no matter where the initial mouse-press was performed. Double-clicking scales the window back to full size, resets the CA transformation and brings back all window controls, effectively putting the window back in default mode.

Delegates take care of their controls

Whenever `toggleMinimize` gets called, it also informs the `WhiteBoardWindowDelegate` of the new window mode. The delegate can now hide the controls of the window's content or perform any other operations if necessary.

6.3.2 Applications

Natively support browser, text editor and sketching

As with the desktop software, the board software natively supports a browser, a text editor and a sketching application. Support for other NSDocument-based applications will be provided as well but not as tightly integrated.

No own applications, everything integrated with NSDocuments

The natively supported types do not get their own applications, but are integrated into the NomadicBoard software. They use NSDocument-based storing and each of them has their own `NSWindowController` subclass to display the documents. They use the same application states as NomadicDesktop.

Browser

Stripped down browser using WebView

The `BrowserDocument` and its `WebController` provide a WhiteBoard window that displays a `WebView` alongside the usual browser controls: Back, forward, refresh and URL

bar. `WebView` is a native Cocoa subclass of `NSView` that uses *WebKit* to render HTML. It receives input from the controls and displays websites accordingly. On load of a website it calls the controller which in turn updates the window title and the URL bar. For a sample browser window see figure 6.2.

Text Editor

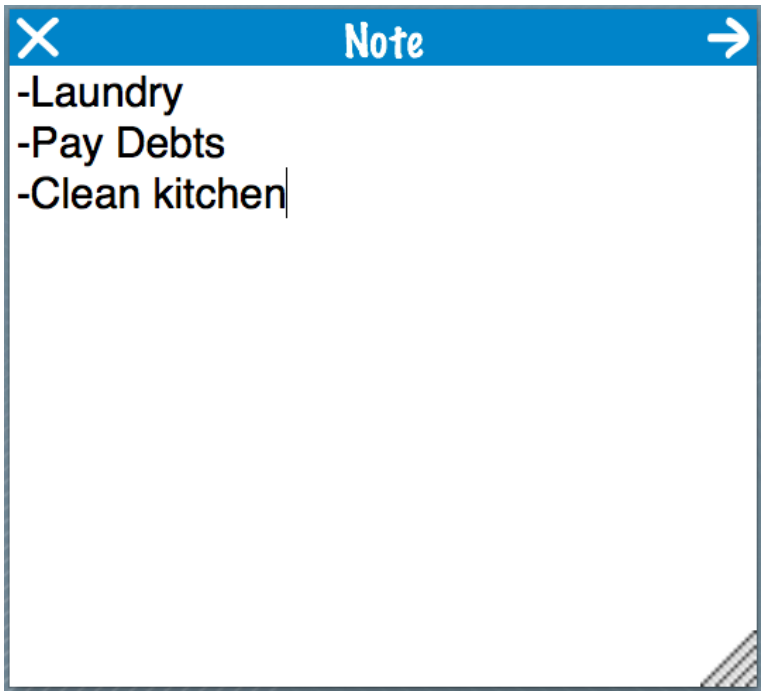

Figure 6.4: A sample Text Editor window in NomadicBoard.

Text View in a Whiteboard window

The `TextDocument` and its `TextController` provide a WhiteBoard window with an `NSTextView` element to display text in an oversized font. The controller provides no more functionality then getting and setting the text of the Text View.

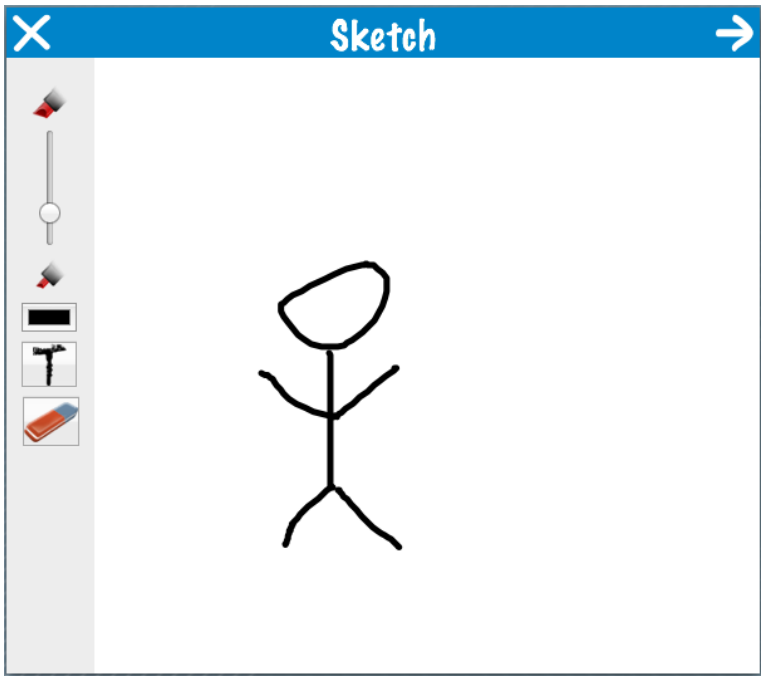

Figure 6.5: A sample Sketching window in NomadicBoard.

Sketching

Based on Sketch It! as well

The sketching part of NomadicBoard is based on *Sketch It!* by Fraikin [2011] as well. More specifically it uses Sketch It!'s model classes to store the data and its `SketchCanvasView` for display purposes.

`SketchDocument` and `SketchController` provide a WhiteBoard window with a `SketchCanvasView` to display sketches or draw on them via touch input. Four of Sketch It!'s drawing tools are available through a toolbar: A brush width slider, a color well, a text mode toggle and an eraser toggle. The miniaturized version of a sketching window hides the control bar.

Generic NSDocument-based Applications

Support documents without special features

The board software supports generic applications that store data with Apple's document system in the same way as the

desktop software (see 6.2.1). This means that any of these applications will appear in regular Mac OS X windows, having none of the special features of our WhiteBoardWindows.

This of course takes away a lot of the advantages that natively-supported applications have and should be seen as preliminary support only. While the tight integration in the system is missing, users can still use software not natively supported.

6.3.3 Annotations

The feature of annotating anywhere in the background posed two problems:

- As mentioned in 5.2.3, users got confused with the mixture of being able to draw in the sketching application and drawing on the background.
- Initial test runs of the software showed that windows (especially browser ones) in default mode take away rather large parts of the screen, effectively hiding most of the background

Skip on annotations, as they do not work well

This meant that annotations did not make a lot of sense in this system, when they are confined to the background. As users could always fall back to annotating with the sketching application, I decided to intentionally drop this feature.

6.3.4 Nomadic Operations

Offer Nomadic Operations in a panel, send to be called from window

As there is no need for a graphics toolbar anymore, a small panel in a corner of the screen that offers the receive operation and a ”new application”-function suffices. The send operation is invoked through the send button of each window.

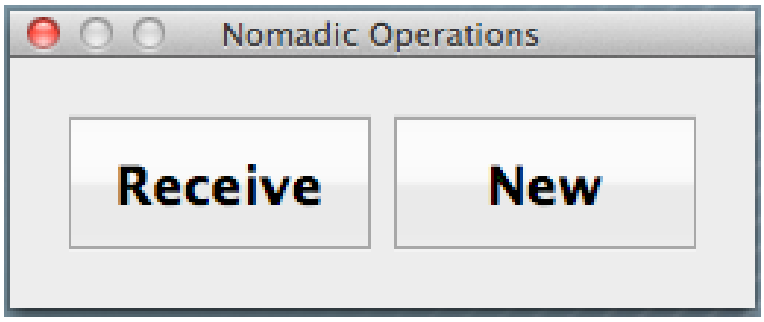

Figure 6.6: The Nomadic Operations panel in NomadicBoard

Operations implemented in app delegate

All Nomadic Operations are implemented in the application delegate. To determine a state to send, the delegate checks, if NomadicBoard itself is the current foreground application. If yes, it checks for the type of document of the foreground window and extracts the state from the appropriate document-class. If not, it fetches the currently opened file from the current foreground application in the same way NomadicDesktop does (see 6.2.1).

On receiving a state it checks for its type. If it is a generic document file, the appropriate application is launched to open the file. If it is one of the natively types, the `NSDocumentsController` creates a new document of that type and a window controller for that document. Afterwards the document gets its properties set, to display the state.

6.4 The Mobile Application

The mobile application called *NomadicPasteboard* is designed for mobile devices running Apple's iOS, with focus on the iPad. It is implemented using Apple's Cocoa Touch[7] framework and Objective-C as its programming language.

Temporary storage for states

As discussed in 4.1.3 it is designed as a short-term storage for application states to enable users to carry their applica-

[7] http://developer.apple.com/technologies/ios/cocoa-touch.html

Figure 6.7: A screenshot of NomadicPasteboard with a text document open

tions around with them. Its functionality is limited to the following four actions:

- Send and receive states using Nomadic Operation.
- Open up previously received states to take a look at them and send them.
- Open received websites in Safari
- Edit received text document

It uses *CoreData* to store all received states. They are listed using a `UITableView` in a `SplitViewController`. Whenever a state is selected from the table, the split view's `DetailViewController` loads an appropriate view as a subview and displays the state data in it:

Combine CoreData and SplitViewController to store and display state information

- A `UIWebView` is used to display websites. Choosing a website from the list also enables the "Open"-button, which will open the website in Safari.
- A `UIWebView` is used to display text documents. The documents are further editable and will be saved upon sending or choosing another item from the list.
- Sketches or generic documents are represented through the screenshot that was taken of them. A `UIImageView` will display the screenshot.

Store and retrieve states with CoreData

As with the other two applications, the application delegate is responsible for sending and receiving states. Upon receiving a state, it gets stored, using CoreData. The state's type and - if available - URL, text and screenshot will be stored in their respective field. All data that is not necessary to represent the state will be encoded as an `NSData` object and stored in the data field.

In return, if a state needs to be send, the data gets decoded again and stored in the state's associated fields, before the state is sent out.

6.5 NomadicApps Framework

The NomadicApps Framework is the heart of all three applications, as it provides the functionality for all Nomadic Operations. It was developed in conjunction with my tutor Jonathan Diehl.

NomadicApps Framework consists of two components:

- *AsyncNetwork* a library that provides access to commonly needed network functionality, using asynchronous connections
- *NomadicApps* a library, that uses AsyncNetwork to realize Nomadic Operations and provide an accessible interface for applications to use them

While I use the framework to implement a specific set of Nomadic Operations, it is not limited to these specifications and can easily be extended to be used with different operations in different applications.

6.5.1 AsyncNetwork

AsyncNetwork is a networking framework that provides easy access to commonly needed functionality like broadcasting messages over a network and transmitting data between two devices.

Use CocoaAsyncSocket for asynchronous connections

AsyncNetwork uses CocoaAsyncSocket[8] a network library that offers wrapper classes for both TCP and UDP sockets. It provides asynchronous, non-blocking operations and supports delegate methods to inform applications that implement it about completions and errors.

AsyncNetwork is built to support interaction between devices that are not previously known, by handling all negotiations with broadcasts. When negotiations are concluded a direct connection can be built between its `AsyncServer` and `AsyncLoader` classes to transmit data.

The framework consists of four connection classes:

- `AsyncServer` is responsible for creating sockets for incoming connections and providing an interface for all TCP methods.
- `AsyncConnectionHandler` stores all established connections for `AsyncServer` to access them.

[8] http://code.google.com/p/cocoaasyncsocket/

- `AsyncLoader` retrieves data that is know to be sent by an `AsyncServer`.
- `AsyncBroadcaster` is used for all sending and receiving of broadcasts.

In detail, the classes work like this:

AsyncServer

AsyncServer provides access to direct TCP connections

`AsyncServer` provides a general interface for accessing direct TCP connections.

It provides one `AsyncSocket` in the form of `listenSocket` that listens for new incoming connections. As a new socket connection gets created, `listenSocket` calls its delegate method `onSocket:didAcceptNewSocket:`. `AsyncServer` forwards the new socket to its `AsyncConnectionHandler`, which takes care of the connection from now on.

`AsyncServer` acts as the `AsyncConnectionHandler`'s delegate and caller for all send and receive operation, providing the interface for any application that builds upon it.

AsyncConnectionHandler

AsyncConnectionHandler stores and handles established TCP connections

`AsyncConnectionHandler` handles all established direct TCP socket connections. It receives new `AsyncSockets` from the `AsyncServer` upon their creation. These sockets are stored and the `AsyncConnectionHandler` will act as their delegate from this point on, receiving all incoming data.

If any data is received, `AsyncConnectionHandler` associates it with the stored `connectionId` and passes it back to `AsyncServer`. Furthermore it provides methods for the server, to send data over the connected sockets.

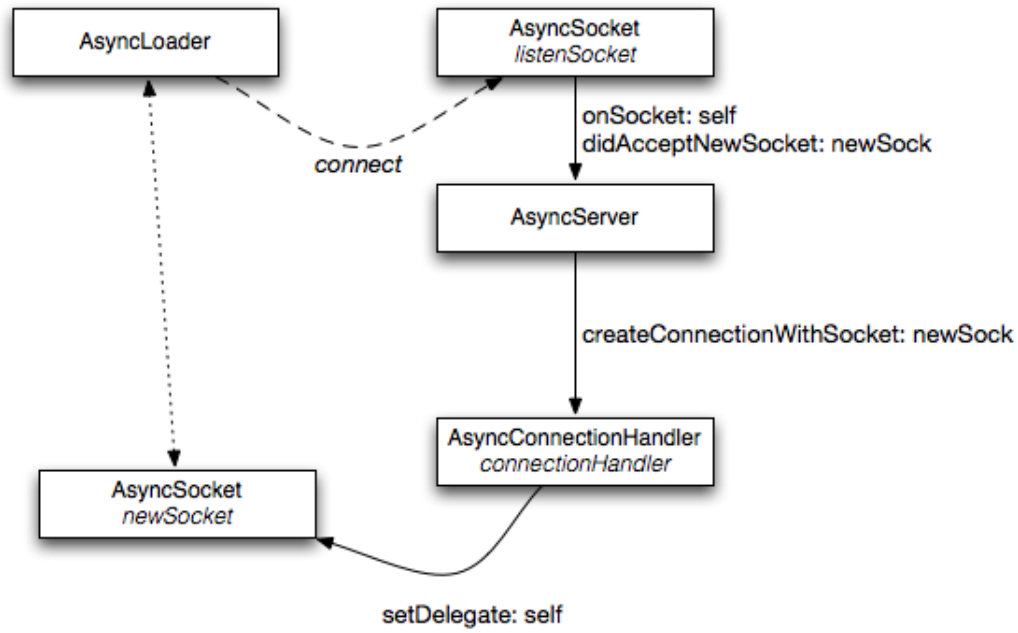

Figure 6.8: An example of how a new connection gets established. Interaction starts with AsyncLoader trying to connect

AsyncBroadcaster

AsyncBroadcaster sends and receives broadcasts over UPD sockets

The `AsyncBroadcaster` is responsible for sending and receiving broadcasts through the network. To do this it uses two `AsyncUDPSocket`s.

The `broadcastSocket` sends `NSData` objects as UDP packets to the whole subnet on a given port. The `listenSocket` receives incoming `NSData` on that port and redirects it to the broadcaster's delegate.

`AsyncBroadcaster` offers the method `broadcast` and the protocol `AsyncBroadcasterDelegate` for the `broadcasterDidReceiveObject` delegate method.

ASyncLoader

AsyncLoader retrieves data that an AsyncServer sends

`AsyncLoader` retrieves data that is known to get sent over a direct connection. In this case the sender on the other end will be an `AsyncServer`.

`AsyncLoader` will be instantiated at runtime with an `AsyncRequest` containing a message and the target host's IP address. It will create an AsyncSocket to directly connect to the target host. Upon connection it will send the request's message over the socket and wait for any incoming data.

Large data does not necessarily get sent at once, but gets split up in chunks. Because of this `AsyncLoader` first receives the length of the incoming data and then the data itself. This ensures all chunking of data to work correctly and only finishing a read operation, when everything was really received. Upon completion it returns itself to the caller with a pointer to the stored response.

ASyncRequest

`AsyncRequest` is a wrapper class to store the information `AsyncLoader` needs to initiate a request, namely being target host, network port and request message. It is designed to work similar to the system `NSURLRequest` in Cocoa uses.

6.5.2 NomadicApps

NomadicApps uses AsyncNetwork to realize Nomadic Operations

AsyncNetwork provides a wrapper for commonly needed network functions. NomadicApps now uses that wrapper to implement Nomadic Operations as I specified them and provide a simple interface for any application to use them.

AsyncNetwork has one main class `NAClient` that provides methods and a protocol to be implemented in a given application that should be extended with Nomadic Operations.

The two other classes - `NAState` and `NADevice` - are models to store the needed information.

NAClient

NAClient handles sending and receiving of states as well as device discovery

`NAClient` is the central class to the NomadicApps framework. It provides wrapper methods to use AsyncNetwork's functionality to send and receive states between devices and broadcast control messages that initiate the sending and retrieving of states. It is also responsible for discovering other Nomadic Application-enabled devices on the network and storing their information.

In all of my applications, the application delegate uses an `NAClient` object, and poses as the `NAClientDelegate` itself. This way the application delegate takes care of all interaction with the NomadicApps framework.

`NAClient` offers a number of options to the application using it, namely being:

- `sending`, boolean, used to initiate send operations.
- `accepting`, boolean, used to initiate accept operations.
- `autoAccept`, boolean, determines whether the client will accept direct send operations and advertise itself as a publicly available device.
- `prefetchState`, boolean, determines whether the client will fetch a state to send upon initiation of a send operation or when the operation actually gets executed.

Additionally it offers the following public methods:

- `setSendingOnce` initiates a single send operation.
- `setAcceptOnce` initiates a single accept operation.

- `sendToDevice` initiates a direct send operation to a known device
- `refreshClients` empties the list of known clients and initiates a new discovery.

Only one NAClient per device

All `NAClient`s use 50001 as the port for both network connections and broadcasts. Unfortunately this means that only one application using NAClient can run per device at a time, as all network traffic on that port will always be routed to the first started instance of `NAClient`.

Sending and Receiving States Sending and accepting states is initiated through "SEND" and "ACCEPT" command messages, broadcasted through the network via a client's `ASyncBroadcaster`.

Send operations initiated through broadcast of control messages

An `NAClient` that changes its `accepting` flag to `YES` sends out a broadcast with "ACCEPT" as its message. Similarly an `NAClient` that changes its `sending` flag to `YES` sends out a "SEND: ANY" message, with the "ANY"-component announcing that the send command is not directed at a specific device. This can cause two different scenarios to occur:

- A client that is set to receiving, gets a "SEND: ANY" message. It sends a "GET STATE" request using `AsyncLoader` directly to the client that broadcasted the send message.
- A client that is set to sending, gets an "ACCEPT" message. It rebroadcasts "SEND: ANY", causing the first case to occur

AsyncLoader retrieves state from AsyncServer

Next , the client that receives a "GET STATE" message uses its `ASyncServer` to directly send back a state encoded as `NSData` to the requesting client. The `AsyncLoader` that requested the state returns the state to the accepting `NAClient`, which finally delivers it to its delegate.

Clients can prefetch states

Determining the state to be sent works in two different ways, depending on the `prefetchState` flag of the client:

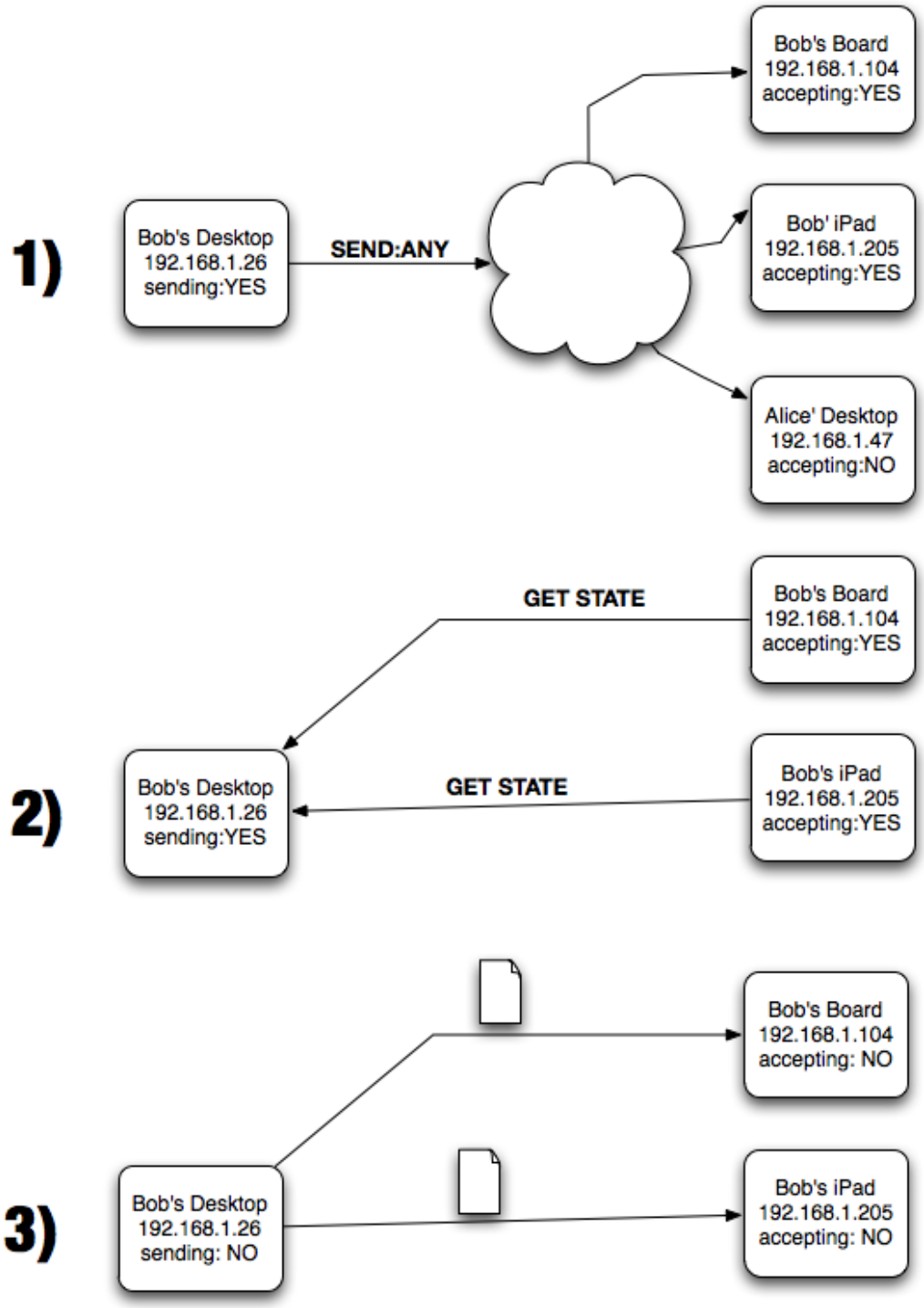

Figure 6.9: An example of sending and receiving. Bob's board and iPad are both set to receiving, when Bob's desktop broadcasts that it wants to send a state.

- Clients that are prefetching, retrieve a state from their delegate, when their `sending` flag changes from `NO` to `YES`
- Clients that are not prefetching, retrieve a state from their delegate, when they receive a "GET STATE" message

In case sending or accepting was invoked by a `setSendingOnce` or `setAcceptOnce` command, the respective flag of a client gets set to `NO` immediately, after communication ended.

States can be send directly to public clients

Directly Sending to Public Clients Clients are marked public, by setting their `autoAccept` flag to `YES`.

A client that has the information of another public client can send it to it directly, by broadcasting a "SEND: HOST", where HOST is the public client's IP address.

When an auto accepting client receives a "SEND: HOST" message, with HOST being its own IP address, it will request a state from the broadcasting device regardless of its `accepting` flag.

The following steps to request a state stay the same as in 6.5.2.

Discovery needed for direct connections

Device Discovery To be able to keep a list of devices that they can directly send to, clients need to first discover other devices that are on the network.

Initiated with "DISCOVER" and "OFFER" broadcasts

A client initiates a discovery by broadcasting a "DISCOVER" message upon first joining the network. This message gets accompanied by all necessary device information. Clients that receive the message, in turn broadcast an "OFFER" message with their own device information. This way a client that initiates discovery will get offers by all devices already in the network.

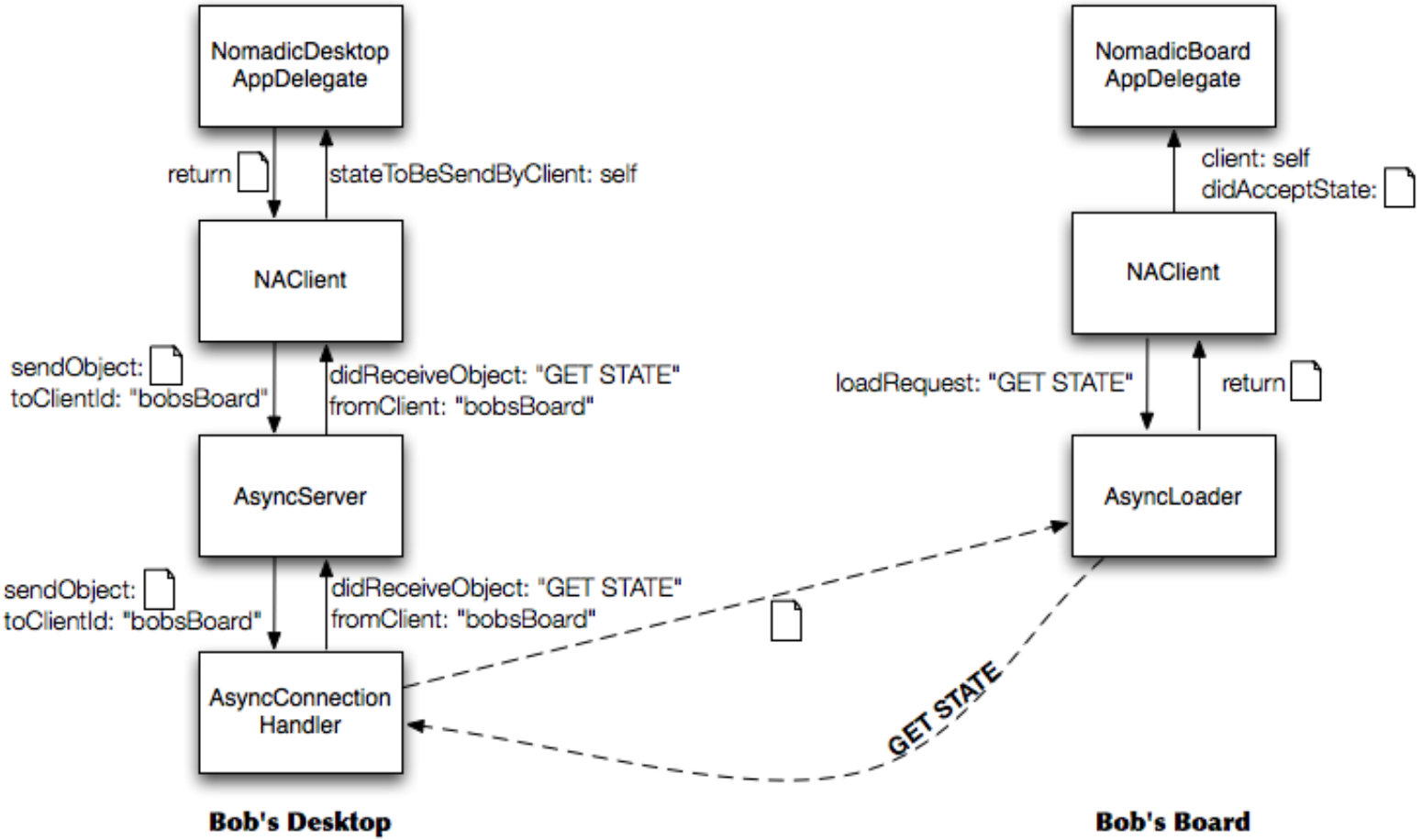

Figure 6.10: A simplified examination of sending/receiving on class-level. NomadicDesktop and NomadicBoard have both exchanged broadcast messages to start sending a state from the desktop to the board and a new socket connection has been established. The calls start with NAClient calling loadRequest from ASyncLoader

Only devices marked as public get tracked

This allows clients to track all devices in the network. However the system distinguishes between private and public devices. For that reason all "DISCOVER" and "OFFER" messages also have a flag that marks the given device as private or public. Upon receiving one of the two messages, a client will store any device's information in his `clients` array of `NADevice` objects, but only if it is flagged as public. Information of devices that are private or already in the array is discarded. Furthermore the client's delegate will have his `client:didFindDevice` method called, so he can use the device information. Applications that use the framework have no way of accessing the information of private devices.

Disconnection is problematic. Periodic refresh necessary

Devices that leave the network are supposed to broadcast a "REFUSE" message. However, due to the asynchronous nature of these network operations there is no way of ensuring the message gets broadcasted correctly upon program termination. As clients can not rely on any information about disconnects, their list of devices have to be refreshed periodically to keep them free of devices no longer in the network. This is implemented by using an `NSTimer` that periodically calls the `refreshClients` method, which removes all available clients and initiates a new discovery.

NADevice

NADevice stores and represents device information

The `NADevice` class is responsible for representing the information of a known device of the network. Specifically it stores the following information for a device:

- `deviceName`: Mac OS' designated computer name of the device
- `userName`: The full name of the user currently logged in on the device. Not available on devices running iOS.
- `operatingSystemName`: The name of the operating system running on the device

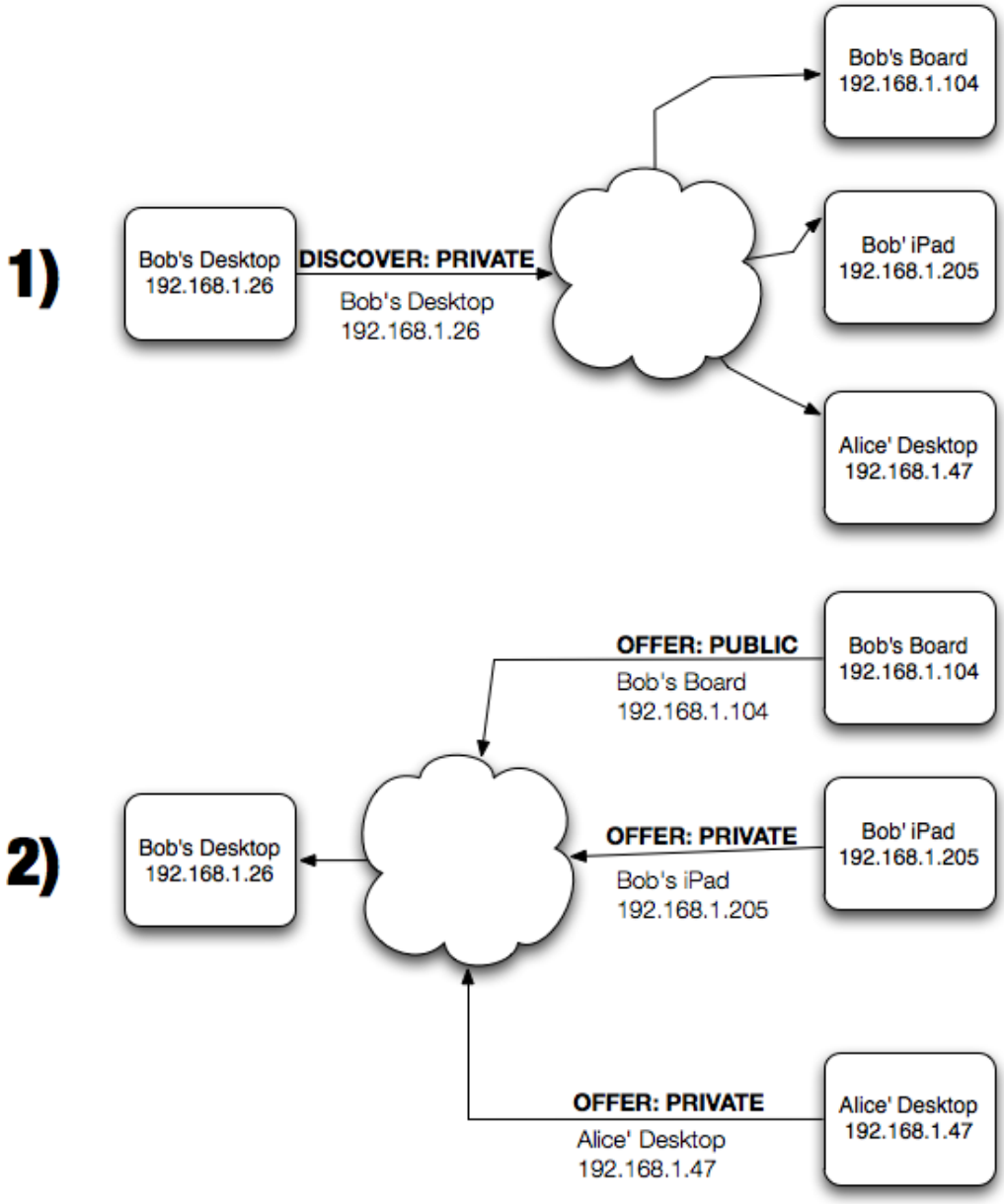

Figure 6.11: An example of device discovery. Bob's desktop joins the network and initiates the discovery. Note that all returned "OFFER" messages are actually broadcasts and will be received by any device in the network.

- `host`: The IP-address of the device

`NADevice` en- and decodes this information from identifier strings that are used in `NAClient`'s broadcast messages.

Additionally it offers the class method `currentDevice` to instantiate an object with the information for the device, the client is currently running on.

Provide current system's info

NAState

`NAState` objects are used to store the application states that are to be sent and received through the network.

As AsyncNetwork uses `NSData` representations for all objects that it send through the network, `NSCoding`-compliance has to be ensured for all state information that is to be exchanged. `NSCoding` is Cocoa's native way to provide object serialization. `NAState` fulfills this task by acting as a coding-compliant wrapper for an `NSDictionary`. It provides the methods `setValue:forKey:` and `valueForKey:` to store any coding-compliant objects in it.

NAState ensures coding-compliancy of state information

6.6 Another Small User Test

I used the nearly finished prototype to perform another small user test. The aim of this test was to see if there are still any general problems with any of the interactions and to possibly fix them.

Informal test with small group to check for usability problems

Again, the test was performed on a purely informal basis with no quantitive analysis and all findings taken from comments of the participants. 5 students of the fields computer science, physics and biology participated in the test.

6.6.1 Tasks

Same tasks as in previous test

As the tasks in 5.2.1 have proven successful in allowing me to evaluate the interactions, I used the same set of tasks again.

The tests starts with a Safari window open on the desktop computer, showing a page to order concert tickets and a miniaturized text window with a sample To-Do list open on the board.

6.6.2 Testing Process

Think-aloud test with full interaction for the users

As the software was fully functional but not completely stable during testing, there was no need for me to interact during this test, except when something unexpected happened.

Again, users were encouraged to voice their thoughts during task execution, to allow me to get into their thought process. If they got stuck on a task I would give them pointers in the right direction without solving it directly.

6.6.3 Findings & Software Updates

Findings could be solved in an updated prototype

This test gave some valuable feedback about the interactions with the software prototype. Some other problems showed themselves during testing phase, but not in actual user tests. After the test I tried to update the prototype with solutions to these problems, which I will also present here:

Send to: Specific device is the preferred interaction

- All users immediately solved the first task, by choosing to directly send the window to the board. This interaction seems to work well and does not need to be changed.

Send/Receive is still unclear

- In the last task, where participants needed to use the two-step send/receive interaction, confusion surfaced upon some of the participants. ”Send” still sug-

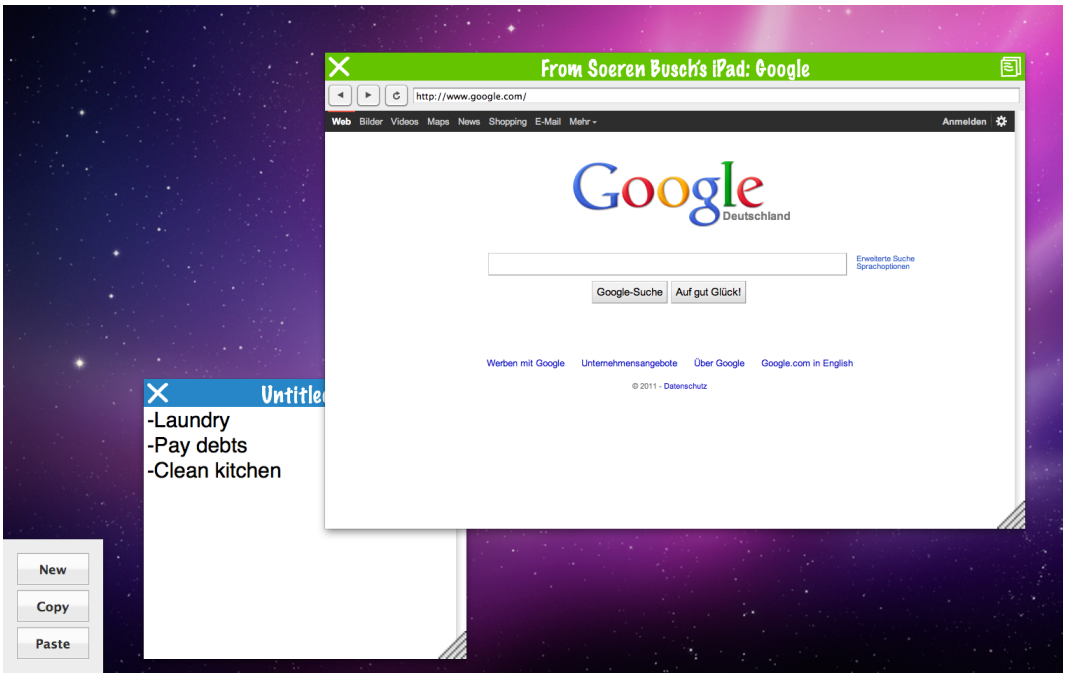

Figure 6.12: Screenshot of an updated version of NomadicBoard. Notable changes: 1) All Nomadic Operations have been renamed to "Copy/Paste" and the associated window button was changed. 2) The window that is marked to be sent gets highlighted in green. 3) A window shows its sender in the title. 4) The Nomadic Operations panel has received a global copy-button and the layout changed.

gests a direct operation and users expected their application to appear on the desktop immediately, when they pushed the button.

Solution: "Copy/Paste" is a known metaphor that needs users to perform two steps as well. Additionally in this system, the network resembles a clipboard, in which applications get copied and then pasted from. Renaming all instances of "Send" (but not "Send to:") to "Copy" and all instances of "Receive" to "Paste" should present users with a more appropriate metaphor and clear up confusion. The "Send"-button on WhiteBoard windows gets replaced with an icon resembling a usual "Copy"-icon.

Feedback between the steps is missing

- After explaining to users that there were two steps needed to complete the operation, some of them noted a lack of feedback on the board that a window was actually marked for sending. This was less

of a concern on the desktop, as users preferred to use "Send to:", which directly gave feedback with the window appearing on the other screen.

Solution: The window that is about to be sent should be highlighted in a clear way:

Enhance `WhiteBoardWindow` with the ability to highlight windows in a different color and unhighlight it again. When a copy operation is initiated, the application delegate saves the foreground window and calls its `higlightWindow` method. Upon completion `unhighlightWindow` is called upon the saved window.

Applications should show their sender

- While it did not pose a problem during the test, a common request of participants was that an application that came from another computer should show its sender, especially on boards, which are publicly available.

  Solution: The information about the sender of a state is known, it just needs to be displayed.

  Enhance `WhiteBoardWindowTitlebar` with a `sender` property and cause it to put "From: sender" in front of the window title, whenever the property is set.

  Enhance the `didAcceptState` method of `NAClientDelegate`s to have `fromDevice` as a parameter, forwarding the appropriate `NADevice` to it. Set the window's title bar's `sender` to the `NADevice`'s username, if available, or device name otherwise.

Receiving before sending causes confusion

- The opportunity to push the receive-button setting an application into receive mode until a state gets sent, posed two problems:

  – Users expected the interaction to be one-way only with choosing the source first and the target next.

  – More importantly it causes a big technical problem: All devices share one network "clipboard". So a user actually putting a device into receive mode, would cause another user in another place, who wants to initiate a send operation,

to send his application somewhere it is not supposed to go

Solution: Make the interaction one-way only: Send first and then receive.

This needs clients to not stay in receive mode, when a receive button is pushed. Use an `NSTimer`, so that an `NAClient`'s `accepting` property gets reset after two seconds of not receiving anything.

The board needs a global "Copy"-button

- As generic applications do not use `WhiteBoardWindow`s and such have no "Copy"-button, there is no way of getting them off the board again.

  Solution: Add a "Copy"-button to the Nomadic Operations panel. The underlying `send` method of the application delegate checks for the foreground window at runtime, so there is no need to change it.

Chapter 7

Evaluation

To evaluate the Nomadic Applications system, I conducted a qualitative user study. As the system needs special hardware I had to limit it to two users, both of them research assistants in computer science.

Two-week real-world scenario study

I tried to evaluate Nomadic Applications in a real-world scenario, as my focus of investigation was the adoption of the system. This resulted in a two-week study, in which the users were free to use the system in any way they wanted to, with digital whiteboards placed in their offices.

7.1 Study Setup

Two participants were provided with an NEC LCD4000 display connected to a computer running NomadicBoard. NomadicDesktop was installed to their primary work computers.

No tasks or restrictions after system introduction

I opened the study with a 20 minute introduction, demonstrating the capabilities of the system. I did not impose any tasks or restrictions upon them on how they should make use of the system and left them to work with it as they saw fit for two weeks.

After that period I conducted individual interviews with

them to gather their experiences with the system as well as points of criticism and ideas for improvement.

I will now present the results of those interviews case-by-case.

7.2 Case 1: Using the board for collaborative work

The first participant was supplied with a SMARTBoard-enhanced display. The display was mounted to a mobile rack and connected to a MacBook that was exclusively used for this task. The rack was put to the wall behind his desk, so people could gather around it but he had no direct view from his seat. He did not have an iPad and so could not use the mobile application.

NomadicBoard as an enhanced whiteboard to support transition between personal and collaborative work

He reported that he mainly used the system when he had to transition from his personal to collaborative work, e.g. discussing a project plan with an assistant. For these kinds of tasks he would usually use his traditional whiteboard to sketch drafts, a practice that does have its disadvantages:

> "You see, this board is full and in the past I used to take photos of it, as did [my student assistant]."

He noted that NomadicBoard would circumvent this issue by allowing him to just collect the needed sketches afterwards and have them in editable form. It would also enable him to perform any preparation work on his personal computer and transfer it to the board, when the time comes.

This resulted in sketching being the primary application of the system. The UI of the sketching tool of NomadicBoard was perceived as not being entirely appropriate, with some of the tools not being immediately recognizable. It also differs too much from the desktop UI of Sketch It! Drawing itself worked well, using the pens of the SMARTBoard system.

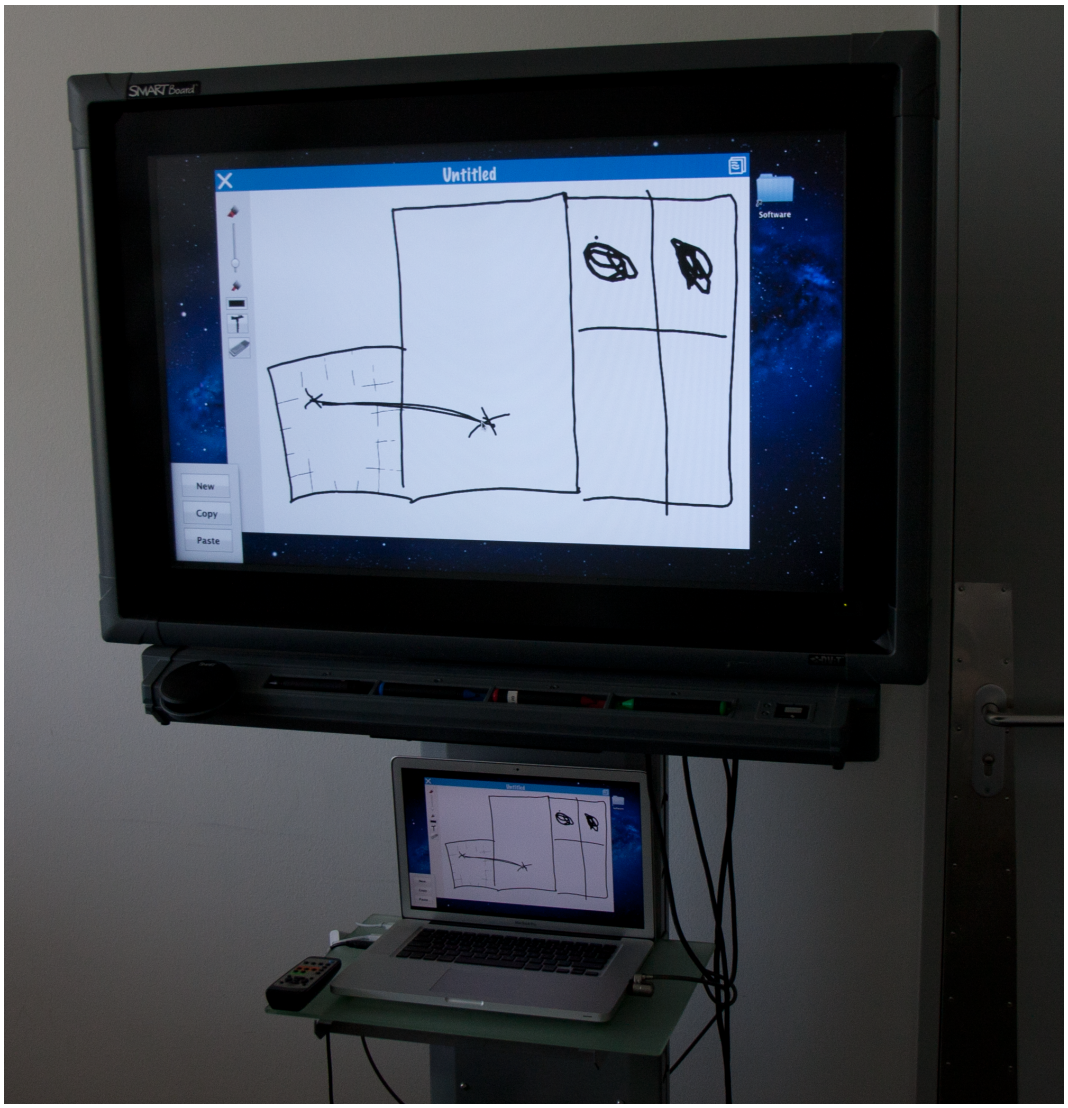

Figure 7.1: The board setup of the first study participant

WhiteBoard windows appropriate

The WhiteBoard window UI was received well and the lack of controls was not seen as a negative. Other people that tried using the board software immediately understood window mode switching. Nomadic Operations on the other hand did need explanations before using them. Having to use the laptop keyboard - situated half a meter below the screen - for text input was criticized for being impractical.

Mobile application potentially very interesting

He expressed interest in using the mobile aspect of the system, as he already uses his phone to have his synced notes available with him and would like to extend that functionality to other applications:

> "I like to use flowcharts created in OmniGraffle[1] to explain things to people. These are the things where I can imagine [using the system]. At the moment I have to print them out and carry them around."

In closing he states that he would like to include a system like this in his regular work, but sees the form factor of the hardware as a hindrance. If it could be reduced to the size of a traditional whiteboard he would definitely consider putting one in his office.

7.3 Case 2: Using the board as a personal storage

The second participant preferred to put his display on a spare table beside his desk. He chose to not use a SMART-Board and opt for a mouse instead. As an owner of an iPad he got the NomadicPasteboard software as well.

Board was used as context and reminder display

He mostly used the board to store his personal data. He would put applications on the board that he needed to stay in sight but out of focus. This way he could look at them, when needed and get them back to his computer to continue working with them.

The main applications he used were the browser and OmniOutliner[2] . In a typical setting the board would contain an OmniOutliner window for his current project plan and several browser windows in miniaturized mode with websites open that he plans on reading later on. This did lead to

[1] http://www.omnigroup.com/products/omnigraffle/
[2] http://www.omnigroup.com/products/omnioutliner/

Figure 7.2: The board setup of the second study participant

some problems, as the relatively low resolution of the display did not allow him to show as many applications as he wanted in a readable way.

He missed the option to fetch any applications from his board back directly similar to the "Send to" command. As a workaround, VNC was used to control the board without needing to get up.

Not satisfied with WhiteBoard windows

The boards WhiteBoard windows were not perceived as an advantage to Mac OS' default windows (partly because of not having to use a touchscreen). The different window modes were put to use, but miniaturized browser windows distorted the content too much and as a result were deemed problematic in a lot of cases.

Collaborative potential regarded high

Despite using the system for personal purposes only, he sees the system's strengths mainly in collaborative settings, especially when combined with mobile features:

> "If I would go to a meeting and didn't want to take my notebook, I could put my notes for that meeting on my iPad and in the meeting I could put them on one of their shared displays"

7.4 Conclusions

Despite being run on such a small scale, this study presented very interesting results:

NomadicDesktop fulfilled design requirements

The main design requirement for NomadicDesktop was, to extend a user's desktop applications with nomadic capabilities while not getting in his way. While the way Nomadic Operations were implemented in the system might not have been ideal, they can easily replaced in the software. Apart from that, the study showed no problems with this part of the software prototype, so we can consider the requirements fulfilled.

NomadicBoard has room for improvement, but proves to be flexible

While NomadicBoard proved to have its problems in both usability and display capabilities, the study showed that you can still successfully employ it in your work routine at this stage. The different ways of using it demonstrated well that the software prototype is very flexible, even with its limited features.

Nomadic features of the board ease the transition between individual and collaborative work

In the case of using the board as a collaborative tool, enhancing traditional whiteboard capabilities with simple data exchange manifested as the key feature. With its sketching application, NomadicBoard provided the strong primitives of a traditional whiteboard. At the same time, Nomadic Operations enable the user to access their work before and after the collaborative setting, rendering traditional workarounds like photographing unnecessary.

Interface of NomadicBoard increases possibilities in a context-display setting

The key features for using the board as a context-display were the nomadic enhancements of existing applications and the different options for window arrangement. Being able to migrate existing applications to the board ensures that users do not have to take any detours to make use of it. Displaying those applications on the board in stripped down versions with the option to miniaturize them, enables users to have more things, they can store in this context and provides them with more opportunities to use spatial arrangement for their organization.

NomadicPasteboard underused but good potential

NomadicPasteboard the mobile application has not seen a lot of use in the study. To be able to draw more use out of it, a wider spread of Nomadic Applications would have been necessary, as there is not much point in carrying around an application if you do not have a target. Still, both participants expressed great interest in having Nomadic Applications be available in a mobile setting and see this possibility of applications leaving the boundaries of one's office as one of the biggest potential strengths of the system.

Nomadic Operations sufficient, but not ideal

The Nomadic Operations were received with mixed feelings. While the ease of directly sending an application to the board was appreciated, an easy way to access the other direction was missed. While the two-step interaction of copying and sending did not pose any particular problems, apart from cases, where different users used it at the same time, the general consensus was that one-step interactions would be preferable for all kinds of operations.

Unused capabilities left

Unfortunately some other potential use cases of the system could not be examined in this study. Both participants only have one computer to regularly work with, so there was no reason for them to use the system for something like dividing one task across multiple devices.

Overall the concept of Nomadic Applications has successfully been tested in a real-life personal workspace scenario. The software prototype did show its limits and problems, but the findings indicate that personal workspaces can benefit from Nomadic Applications. Even with its limited support capabilities the software prototype was flexible enough to be put to use in two completely different ways.

Chapter 8

Summary and future work

8.1 Summary and contributions

In this thesis I presented the design and development of the three applications NomadicDesktop, NomadicBoard and NomadicPasteboard. They build upon the Nomadic Applications concept, in which applications can be freely moved between different devices, and combine it with the concept of digital whiteboards. The final implementation is a fully functional software prototype for different UbiComp device classes that can be employed in different ways.

Common practices and problems in personal workspaces were identified

I started my research by showing three different studies that dealt with different practices that are common in personal workspaces. Those practices were the usage of multiple devices, taking information scraps and using whiteboards in synchronous and asynchronous activities. In conclusion I presented ideas on how Nomadic Applications can be used to create a system that supports all of these practices.

Design requirements have been established

Next, I established an initial set of design requirements my system should meet. These requirements included the three classes of devices the system should support and the ways they should be supported. They also defined an approach

to Nomadic Operations.

The paper prototype showed an initial design. Interactions were evaluated

I created a paper prototype to fulfill these requirements. The prototype consisted of an interface for a desktop computer and a board that both support Nomadic Applications. I used the prototype as a starting point for my design, as well as evaluating it in a controlled study. With this prototype I could already find a number of problems with the interactions and start finding solutions to solve them.

Three applications built upon the NomadicApps Framework provide nomadic capabilities for different device classes

The initial design of the paper prototype was used to build a fully functional software prototype. This prototype consisted of three applications: NomadicDesktop extends existing applications with nomadic functionality via a status bar icon. NomadicBoard provides minimalistic versions of supported applications and displays them in board appropriate windows that can be used in two display modes. NomadicPasteboard acts as a mobile storage for application states. All of these applications use the NomadicApps Framework to realize their NomadicOperations. The software prototype was evaluated in another controlled study and more usability problems were eliminated.

The evaluation showed different ways in which the system can be used

The software prototype was evaluated in a two-week user study in which two participants were supplied with a board and could use the software in any way they wanted to. The proceeding interviews showed, how both used vastly different approaches to enhance their regular work with the software. One of them used it to enable him to better transition between personal and collaborative work. The other adopted it as a personal storage space for applications that he needed to keep in context or be reminded of. These case studies demonstrated the high potential of such a system.

My research contribution is the demonstration of flexibility Nomadic Applications provide for personal workspaces

The Contribution of this work is the confirmation that the Nomadic Applications concept provides a way to support users in their personal workspace. Users adopted to the system and proved its flexibility by using it successfully in completely different ways. The implementation of the system mostly fulfilled the design requirements and with its underlying framework can provide a basis on which similarly systems can easily be implemented. From these results I conclude that the research goals posed in 1.1 were

successfully achieved:

While limited in its applications, I showed the design and implementation of a system that enhances existing applications with nomadic functionalities. Placing the system in a work environment showed us different ways of users enhancing their workplace with Nomadic Applications, as well as points of improvement that can be fulfilled with future designs.

8.2 Future work

8.2.1 Enhanced Support

Support for both devices and applications was very limited in the prototype. To achieve a wider spread of Nomadic Applications, we need offer more options of what can be used in the system.

Find better ways to extract state information from applications

Application support in the prototype was accomplished by communication via AppleScript or - in the case of Sketch It! - my own implementation. This method is both limited in the number of applications that can be supported and the information that can be extracted. Better ways of extracting the state information from given applications need to be found to broaden the scope of Nomadic Applications.

Support more devices by using more generic protocols

Similarly support for systems that don't run Mac OS X or iOS respectively can widen the acceptance of Nomadic Applications. This should be less of an issue, as the communication between devices uses generic TCP and UDP sockets, which should be available in any given operating system. Encoding the information however needs to move away from the `NSCoding` protocol to allow for interoperability to happen. As a solution the NomadicApps Framework should replace `NSCoding` with a more generic approach to object serialization like XML or JSON[1] .

[1] http://www.json.org/

8.2.2 Additional Features

Increase state information that is being transferred

As mentioned before, the state information extracted from applications was very limited. It normally only included a file, a URL or a text string. However there is more to an application state then just the opened content. Interaction histories for example provide important information and are one of the features generally missed in todays' ways of information exchange between devices (see 2.1). If we can extract more information from running applications, we can also extend the state information that is being transferred and provide a better feeling of migrating a whole application, instead of moving files around between devices.

Allow more flexible options for arrangement on the board

Having windows that can switch between regular and miniaturized modes on the board provided a way for users to arrange a larger amount of applications on the board. It still only allowed for a limited number of applications to be stored side-by-side in a readable way and the fixed size of miniaturized windows limited the flexibility in arranging them. Additionally the implementation I used caused windows to have their content distorted in miniaturized mode, especially in conjunction with the browser application. We need to find better and more flexible ways to arrange a large number of windows on a board. A fully zoomable UI would be one possible approach to this problem.

Increase interaction on mobile devices

Being able to not only store application states on a mobile device but interact with them as well, would vastly increase the possible usage scenarios for mobile devices in the system. It would enable users to not only carry their applications around but perform work on them, e.g. while being on the road. To achieve this, we need to find suitable mobile applications and ways to integrate them in our system as well.

8.2.3 Nomadic Operations

The way Nomadic Operations were implemented in the system, proved to be far from ideal. The main problems that occurred:

- The copy/paste interaction uses one metaphorical clipboard per network. With only a very limited number of participants, situations in which states arrive at a wrong device, due to synchronous invocation of Nomadic Operations, already happened. The scope of this problem only increases, if more people use Nomadic Applications.

  Network clipboard too ambiguous

- Migrating applications to non-public devices needs more physical interaction than users would like to have. There should be easier ways to fetch applications back from the board.

  Too much physical interaction

An obvious solution would be, to extend the "Send to:"-device list to all available devices, needing confirmation on any private device when a state is incoming. This solution however would still not enable direct fetching of applications. It can also lead to a crowded device list very quickly.

Short term solution: Extended device list and confirmation

More research has to be done in this field to find an optimal way to implement Nomadic Operations that is flexible enough, does not have any problems with ambiguity and does not burden the user with any technical problems.

Techniques like the iconic map interface of ARIS (see 3.1.4) or Hyperdragging (see 3.3.2) could help with these issues, if we can find ways of implementing them in our system.

Another extension to Nomadic Operations that would benefit the system is input redirection. When used in a personal setting it would also decrease the physical interaction needed to use all capabilities of the board.

Input redirection would improve accessibility

8.2.4 Additional User Studies

While showing interesting results, the user study was performed on a very small scale. It is my belief that a study with more participants in a longer period of time, would not only refine those results, but also show other ways in which the system can be successfully used.

Larger scale studies can show other usage scenarios

Research the application of the concept in collaborative settings

My research was mainly focussed on supporting users in their personal workspaces. However Nomadic Applications should be able to enhance the experience of purely collaborative settings as well, as was already successfully demonstrated by Fraikin [2011]. Performing studies in meeting scenarios might lead to more interesting results.

Forego boards, focus on wide application

There is another approach to test the software on a larger scale without needing as much specialized hardware. Distributing only NomadicDesktop and NomadicPasteboard in a large work environment can show us, if a wider penetration of Nomadic Applications can lead to a wider adoption.

8.2.5 Better Form Factors

Virtual whiteboards should not take more space than traditional ones

The hardware used in my prototype was seen as too impractical to be permanently adopted in a personal workspace. If we want to establish a system like this as a real enhancement to traditional whiteboards, we need hardware with less space consumption and better display capabilities.

Multi-touch support enhances collaborative scenarios

Multi-touch displays are becoming more prevalent every day. The addition of multi-touch capabilities to the board software would vastly improve its possibilities, especially in collaborative settings.

Bibliography

Lionel Balme, Alexandre Demeure, Nicolas Barralon, Joëlle Coutaz, and Gaëlle Calvary. CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In *Ambient Intelligence*, pages 291–302, 2004. doi: 10.1007/978-3-540-30473-9_28. URL http://www.springerlink.com/content/p27tmcvqw24r2ljj.

Michael Bernstein, Max Van Kleek, David Karger, and M. C. Schraefel. Information scraps. *ACM Transactions on Information Systems*, 26(4):1–46, September 2008. ISSN 10468188. doi: 10.1145/1402256.1402263. URL http://portal.acm.org/citation.cfm?doid=1402256.1402263.

Jacob T Biehl and Brian P Bailey. ARIS: an interface for application relocation in an interactive space. In *Proceedings of Graphics Interface 2004*, GI '04, pages 107–116, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society. ISBN 1-56881-227-2. URL http://portal.acm.org/citation.cfm?id=1006058.1006072.

Jacob T. Biehl and Brian P. Bailey. Improving interfaces for managing applications in multiple-device environments. *Proceedings of the working conference on Advanced visual interfaces - AVI '06*, page 35, 2006. doi: 10.1145/1133265.1133273. URL http://portal.acm.org/citation.cfm?doid=1133265.1133273.

Jacob T Biehl, William T Baker, Brian P Bailey, Desney S Tan, Kori M Inkpen, and Mary Czerwinski. Impromptu: a new interaction framework for supporting collaboration

in multiple display environments and its field evaluation for co-located software development. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 939–948, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi: http://doi.acm.org/10.1145/1357054.1357200. URL http://doi.acm.org/10.1145/1357054.1357200.

David Dearman and Jeffrey S Pierce. It's on my other computer!: computing with multiple devices. *Writing*, pages 767–776, 2008.

Mario Fraikin. Collaborating with Tangible Windows - Idea Generation and Information Exchange in Small Groups. Diploma thesis, RWTH Aachen University, May 2011.

Michael Haller, Peter Brandl, Daniel Leithinger, Jakob Leitner, Thomas Seifried, and Mark Billinghurst. Shared Design Space: Sketching ideas using digital pens and a large augmented tabletop setup. *Advances in Artificial Reality and Tele-Existence*, pages 185–196, 2006. URL http://www.springerlink.com/index/2W34W2TR74218V39.pdf.

Michael Haller, Jakob Leitner, Thomas Seifried, and JR Wallace. The NICE discussion room: integrating paper and digital media to support co-located group meetings. In *CHI*, 2010. URL http://portal.acm.org/citation.cfm?id=1753418.

Brad Johanson, Shankar Ponnekanti, Caesar Sengupta, and Armando Fox. Multibrowsing: Moving web content across multiple displays. In *Ubicomp 2001: Ubiquitous Computing*, pages 346–353. Springer, 2001. URL http://www.springerlink.com/index/arg6ytayarbrdvyh.pdf.

Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Pervasive Computing Magazine Special Issue on Systems. In *Pervasive*, 2002.

Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 0125184050.

Antti Oulasvirta and Lauri Sumari. Mobile kits and laptop trays: managing multiple devices in mobile information work. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 1127–1136, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-593-9. doi: http://doi.acm.org/10.1145/1240624.1240795. URL `http://doi.acm.org/10.1145/1240624.1240795`.

Jun Rekimoto. Pick-and-drop: a direct manipulation technique for multiple computer environments. In *UIST*, page 39. ACM, 1997. ISBN 0897918819. URL `http://portal.acm.org/citation.cfm?id=263505`.

Jun Rekimoto and Masanori Saitoh. Augmented surfaces: a spatially continuous work space for hybrid computing environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 378–385, New York, NY, USA, 1999. ACM. ISBN 0-201-48559-1. doi: http://doi.acm.org/10.1145/302979.303113. URL `http://doi.acm.org/10.1145/302979.303113`.

Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H Campbell, and Klara Nahrstedt. Gaia : A Middleware Platform for Active Spaces. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):65–67, 2002.

Reinhard Sefelin, Manfred Tscheligi, and Verena Giller. Paper Prototyping - What is it good for ? A Comparison of Paper- and Computer-based Low-fidelity Prototyping. *New Horizons*, pages 778–779, 2003.

Norbert A Streitz, Jiirg Geibler, Torsten Holmer, Shin Konomi, Christian Miiller-tomfelde, Wolfgang Reischl, Petra Rexroth, Peter Seitz, and Ralf Steinmetz. i-LAND: an interactive landscape for creativity and innovation. In *CHI*, number May, 1999.

Anthony Tang, Joel Lanir, Saul Greenberg, and Sidney Fels. Supporting Transitions in Work : Informing Large Display Application Design by Understanding Whiteboard Use. In *GROUP*, 2009.

Mark Weiser. The Computer for the 21st Century. *Scientific American*, 1991. URL `http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html`.

Index

Typeset August 7, 2011