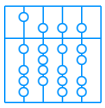


Institut für Informatik der Technischen Universität München

A Software Toolkit and Authoring Tools for User Interfaces in Ubiquitous Augmented Reality

Christian Sandor

TECHNISCHE UNIVERSITÄT MÜNCHEN

Institut für Informatik der Technischen Universität München

A Software Toolkit and Authoring Tools for User Interfaces in Ubiquitous Augmented Reality

Christian Sandor

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. Rüdiger Westermann
Prüfer der Dissertation:	1.	Univ.-Prof. Gudrun J. Klinker, Ph. D.
	2.	Prof. Steven Feiner, Ph. D. Columbia University New York/USA

Die Dissertation wurde am 7. 7. 2005 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12. 9. 2005 angenommen.

To my father
László Sándor
for teaching me so much.

A book should serve as the ax
for the frozen sea within us.

FRANZ KAFKA

Abstract

Ubiquitous augmented reality (UAR) is an emerging human-computer interaction technology, arising from the convergence of augmented reality and ubiquitous computing. In user interfaces in UAR, visualizations can augment the real world with digital information, interactions can follow a tangible metaphor and both should adapt according to the user's context and are distributed on a possibly changing set of devices. In the past years, the basic technology for UAR has become fairly mature, but important challenges remain. In my dissertation, I address the challenges in developing new user interfaces for UAR systems.

Since the standards for user interface elements for UAR are still immature, it is difficult to create a supporting software infrastructure. Another concern is the development process for new user interface elements. Evidence suggests that it is imperative to work in interdisciplinary teams. However, it is challenging to illustrate the design space for new user interface elements to the team members.

To address these two problems, my approach consists of two steps. First, I have created the component-based toolkit AVANTGUARDE, which enables the execution of user interfaces in UAR. AVANTGUARDE supports brainstorming sessions that use a running system, through rapid modifiability of user interface elements. The core components of AVANTGUARDE include a Petri net-based dialog control management system and a viewer for augmented reality scenes.

Second, on top of AVANTGUARDE, I have developed several tools that further ease the task of experimenting with new interaction elements. These tools employ a combination of traditional graphical user interfaces and user interfaces in UAR. To support their usage in brainstorming sessions, the main design goals were minimization of turnaround time and ease of use.

AVANTGUARDE was successfully used to build several research prototypes. Since AVANTGUARDE was developed through an iterative process, valuable insights regarding the components and overall architecture have been collected. The authoring tools on top of AVANTGUARDE have been successfully used by programmers and non-programmers to modify user interfaces during system runtime.

Zusammenfassung

Ubiquitous Augmented Reality (UAR) ist eine neue Form der Mensch-Maschine Interaktion, die aus der Konvergenz von Augmented Reality und Ubiquitous Computing entsteht. Bei UAR Benutzeroberflächen reichern Visualisierungen die reale Welt mit digitalen Informationen an, Interaktionen folgen einer anfassbaren Metapher und beide sollen sich anhand des momentanen Benutzerkontexts anpassen und sich auf eine, sich möglicherweise dynamisch ändernde, Menge von Geräten verteilen.

Während der letzten Jahre hat die zugrundeliegende Technologie für UAR einen ziemlich reifen Status erreicht, jedoch verbleiben immer noch wichtige Forschungsfragen. In meiner Dissertation behandle ich die Fragen im Themenbereich Mensch-Maschine Interaktion für UAR Systeme.

Weil die Standardisierung der Elemente einer Benutzeroberfläche für UAR noch unvollständig ist, ist es schwer eine Software Infrastruktur für diese zu bauen. Ein weiteres Problem ist der Entwicklungsprozess für neue Elemente einer Benutzeroberfläche. Der momentane Stand der Forschung legt nahe, das es unabdingbar ist in interdisziplinären Teams zu arbeiten. Es ist eine Herausforderung, den Mitgliedern eines solchen Teams die Möglichkeiten für neue Elemente einer Benutzeroberfläche anschaulich zu machen.

Um diese zwei Probleme anzugehen, verwende ich einen zweistufigen Ansatz. Erstens habe ich ein komponenten-basiertes Toolkit Avantguarde gebaut, das es erlaubt Benutzeroberflächen in UAR auszuführen. Avantguarde unterstützt interdisziplinäre Brainstormingsessions, die ein laufendendes System benutzen, durch die schnelle Veränderbarkeit von Elementen der Benutzeroberfläche. Die Kernkomponenten von Avantguarde sind ein Petrinetz-basiertes Dialogkontrollmanagementsystem und eine Anzeigekomponente für Augmented Reality Szenen.

Zweitens habe ich auf Avantguarde aufsetzend verschiedene Werkzeuge entwickelt, die das Experimentieren mit neuen Elementen einer Benutzeroberfläche weiter vereinfachen. Diese Werkzeuge benutzen eine Kombination von traditionellen Benutzeroberflächen und Benutzeroberflächen in UAR. Um deren Benutzung in Brainstormingsessions zu ermöglichen waren die wichtigsten Designziele die Minimierung von Änderungszeiten und eine einfache Bedienung.

Avantguarde wurde erfolgreich eingesetzt um mehrere Forschungsprototypen zu bauen. Avantguarde wurde mit einem iterativen Prozess entwickelt, so das interessante Einsichten bzgl. seiner Komponenten und Architektur gewonnen werden konnten. Die Entwicklungswerkzeuge, die auf Avantguarde aufsetzen, wurden erfolgreich von Programmierern und Nichtprogrammierern eingesetzt, um Benutzeroberflächen während deren Ausführung zu verändern.

Preface

When looking for advice on the internet on how to write a thesis, I have found a remarkable sentence: "Writing a thesis is a once in a lifetime experience". During the last six months when writing my thesis, I have realized the full consequences of this sentence—the good sides, but also the bad sides of this unique experience ... Many people helped me to make my thesis become what it is, I am extremely grateful to all of you!

I had the pleasure to work with two fantastic advisors: Gudrun Klinker and Steven Feiner. Gudrun has deeply impressed me with her metaphorical way of thinking. I will never forget her bubble metaphor for knowledge acquisition. Furthermore I think that Gudrun is the model of an inspiring researcher. Steven Feiner has the best intuition in judging novel ideas of all researchers I have ever met. His deep knowledge in scientific literature has astonished me several times—making me read scientific papers from the 1960s was something only he could make me do.

During the last weeks of completing this thesis, several reviewers helped me with their deep comments. Thank you, Martin Wagner, Martin Bauer and Otmar Hilliges. Several people directly influenced the visual appearance of my thesis. Thank you, Asa MacWilliams for supplying me with your fantastic LaTeX style. Thank you, Emmeline Oh for your beautiful drawings for this thesis that help me enormously to explain my ideas more concisely (also, you managed to make me laugh during the last weeks of writing this thesis). I would also like to thank all researchers who let me use their images in my thesis.

I had the pleasure to work with some great researchers during my Ph.D. studies. The biggest influence came from my fellow Ph.D. students in the DWARF team: Thomas Reicher, Asa MacWilliams, Martin "Mäh" Bauer, Martin "Zefix" Wagner, Marcus Tönnis and Daniel Pustka. I have learned a lot when working with you—thank you so much. Thomas Reicher also supervised my masters thesis. He made me realize that everything is not as simple as it seems by his deep understanding. Thinking back, he was one of the main reasons to go for a Ph.D. .

I supervised many students during my Ph.D. studies. They have helped me enormously by implementing my (sometimes crazy) ideas. I really enjoyed working with you! Thank you, Vinko Novak, Fabian Sturm, Markus Geipel, Maximilian Schwinger, Korbinian Schwinger, Peter Hallama, Nikolas "Ne Nico" Doerfler, Michael Riedel, Johannes Wöhler, Christian Kulas, Ming-Ju Lee and Marco Feuerstein. The student I have worked with the most was Otmar Hilliges. Thank you Otmar, it was a pleasure to work with you!

I also had the pleasure to work closely together with the members of the STUDIERSTUBE research group from Vienna, Austria. Their work was very influential for me. Thanks to all of you! During my stay in New York, I could work together with a lot of inspiring researchers at the Computer Graphics and User Interfaces Lab at Columbia University. It was a truly unique experience working with you!

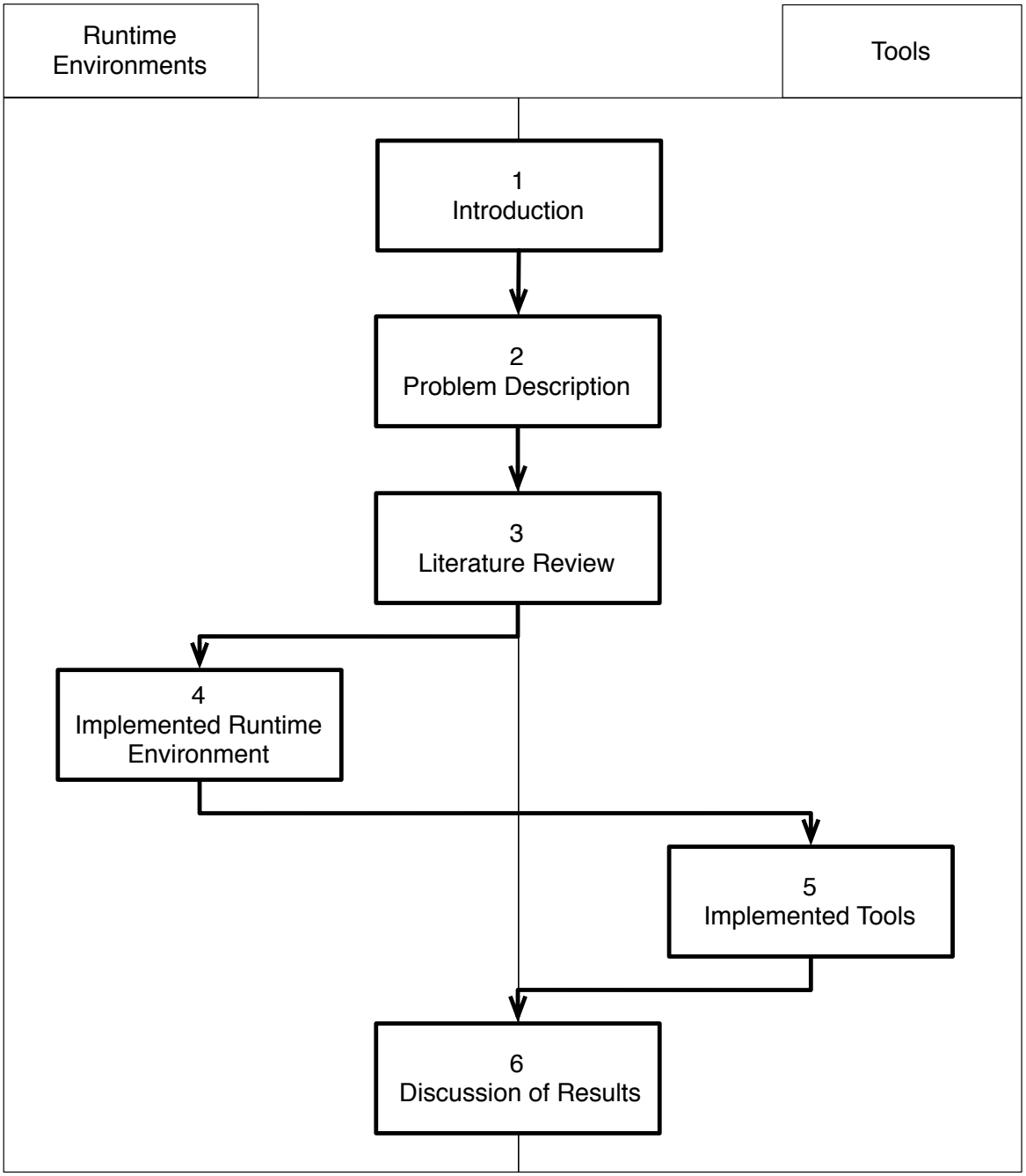
During the last couple of years I had delightful discussions with several outstanding researchers. Some of them had an especially deep influence on my thinking. Thank you, Alan Kay, Brad Myers, Randy Pausch, Ivan Poupyrev, Dieter Schmalstieg, Brygg Ullmer, Ben Bederson, François Guimbretiére, Nassir Navab, Blaine Bell and Alex Olwal. Several researchers have been very influential for my thinking, however I did not have the pleasure to talk to them deeply, yet. Thank you, Hiroshi Ishii, Mark Billinghurst and Jun Rekimoto.

To distract myself from the shallow waters of research, I am grateful to all of my friends who did an excellent job!

I would also like to thank all women in my life—thank you for your love and patience.

Finally, and most importantly, I would like to thank my family for their intense support during my whole life.

Graphical Overview



Overview

Contents

Introduction

Ubiquitous augmented reality (UAR) is the convergence of augmented reality and ubiquitous computing. This dissertation address the challenges of developing user interfaces for this emerging field.

Ubiquitous augmented reality (UAR) is an emerging human-computer interaction technology, arising from the convergence of augmented reality and ubiquitous computing. In the past years, the basic technology for UAR has become fairly mature, but important challenges remain.

This introductory chapter starts with an explanation of UAR (Section 1.1). Then, the challenges that are addressed within this thesis are presented (Section 1.2). After briefly explaining the proposed solution (Section 1.3), we proceed by stating the research hypothesis (Section 1.4). The approach I have used to realize my proposed solution is presented in Section 1.5. Section 1.6 gives an overview of the research contributions of this thesis. Finally, the structure of this document is outlined in Section 1.7.

1.1 What is Ubiquitous Augmented Reality?

The best way to predict the future is to invent it.

ALAN KAY

During the last decade, the way humans interact with computers has changed fundamentally. Several technological developments have made this change possible. According to Moore's law, every 18 months the number of transistors per square inch doubles. As a result, computers get continuously smaller and faster. In addition to the ever increasing number of computers that can be found in the environment, most people also carry more and more electronic devices on their body, such as personal digital assistants, cell phones, tablet PCs and mp3 players. Furthermore, wireless technologies are continuously improving and becoming more wide-spread. UMTS (Universal Mobile Telecommunications System), iMode and wireless access points are going to cover most urban areas soon. This means that computers will be interconnected almost everywhere, thus enabling new kinds of applications.

These rapid technological developments also introduce new problems. Most electronic devices have a very specific and complicated interface that a user has to learn. When connecting

different devices, things can get even more complicated. As a result, several research groups have been working on new interaction paradigms that improve the access to digital information.

The paradigm that my dissertation addresses is called ubiquitous augmented reality (UAR). It has been first defined by Asa MacWilliams in his Ph.D. thesis [101] as a unification of ubiquitous computing and augmented reality (AR). He has explored the implications of developing software for UAR systems from a *software engineering* perspective, whereas my thesis focusses on the implications of developing *user interfaces* for UAR.

In the remainder of this introductory section on UAR, I will first explain the contributing user interface paradigms, then present illustrative examples that show how these can be combined into UAR and finally conclude by summing up the main characteristics of user interfaces in UAR.

Contributing User Interface Paradigms

Most user interfaces today are driven by the WIMP (Windows, Icons, Menus, Pointer) paradigm that was developed in the 1970s at Xerox PARC. Current research suggests that WIMP is well suited for working with a stationary computer, whereas it has shortcomings for mobile user interfaces. This fact has inspired a variety of other user interface paradigms. The contributing user interface paradigms for user interfaces in UAR are: AR, ubiquitous computing, wearable computing, hybrid user interfaces and tangible user interfaces. They are being investigated by many scientific researchers world-wide. This section presents very short explanations of them.

Augmented reality complements or *augments* the user's reality by adding virtual objects to it, in effect enhancing the user's senses. Figure 1.1(a) shows a user wearing a head-mounted display that shows a video image of the real world enhanced with a virtual figurine. A more general term for these kinds of user interfaces is mixed reality (MR). Section 2.1.2 will explain the difference between AR and MR in detail.

Ubiquitous computing is driven by the vision that computers are everywhere, but that the user is not consciously aware of their presence [181]. Processors and wireless network facilities are embedded in everyday items such as pens, glasses and light bulbs. The example in Figure 1.1(b) shows a water glass that reminds its owner to drink, because it has detected that he is in danger of becoming dehydrated. This example illustrates an important characteristic of a ubiquitous computing system: it has to react according to the context of the user and not only to direct input by the user.

Wearable computing focusses on computers that are worn on the user's body (see Figure 1.1(c)). Compared to ubiquitous computing, it introduces new research problems: for example, efficiency regarding power consumption, or convenient form factors for wearable devices.

Tangible user interfaces An important input paradigm for ubiquitous computing is tangible user interfaces, which were pioneered by Hiroshi Ishii and the Tangible Media Group at MIT.

(a) Augmented reality [68] (b) Ubiquitous computing [22] (c) Wearable computing [93]

Figure 1.1: User interface paradigms related to UAR.

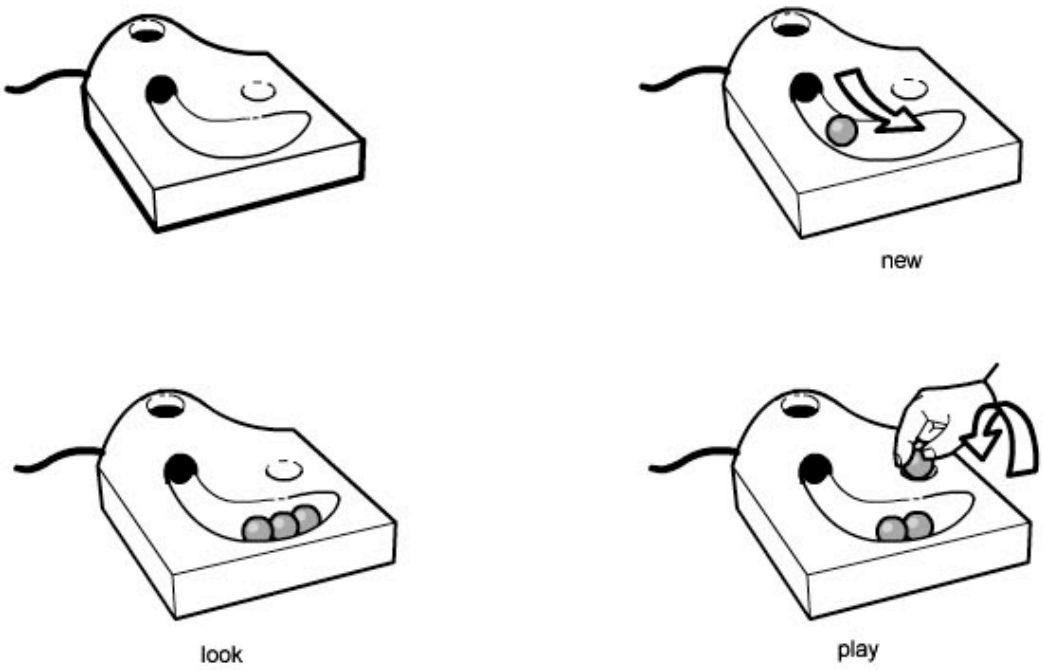
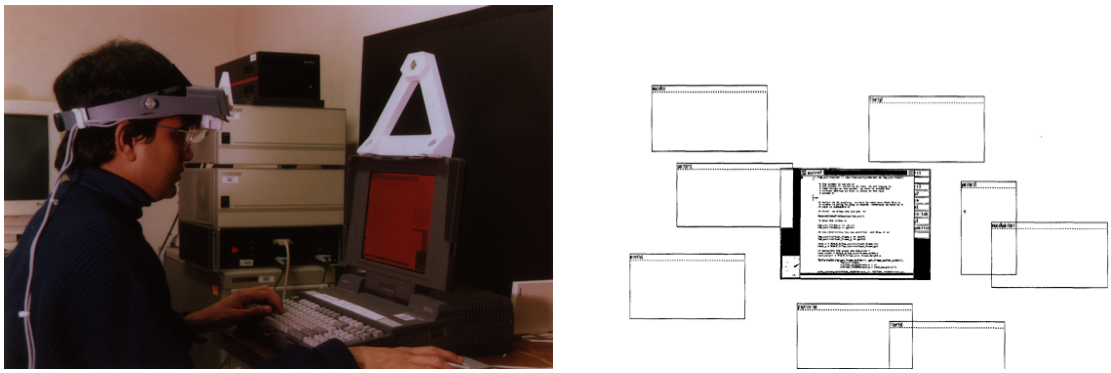


Figure 1.2: Example of a tangible user interface: the marble answering machine (courtesy of Durrell Bishop).



(a) User wearing a head-mounted display sitting in front of a workstation.

(b) The user's view.

Figure 1.3: First example of a hybrid user interface [47].

Their core idea is to use everyday items as input and output simultaneously. For example, the tangible answering machine (see Figure 1.2) represents voice messages by marbles that are put on a tray. By putting them into a slot, the corresponding message will be played back.

Hybrid user interfaces Feiner and Shamash [47] coined the term *hybrid user interfaces* in 1991. Their original example (see Figure 1.3) was a window manager that extends screen space by the combined use of a classical 2D screen and an augmented reality view of windows that are overlaid on the real world. This original example illustrates the goal of hybrid user interfaces: multiple displays with different characteristics are combined into a more powerful visualization than each display would be able to provide on its own.

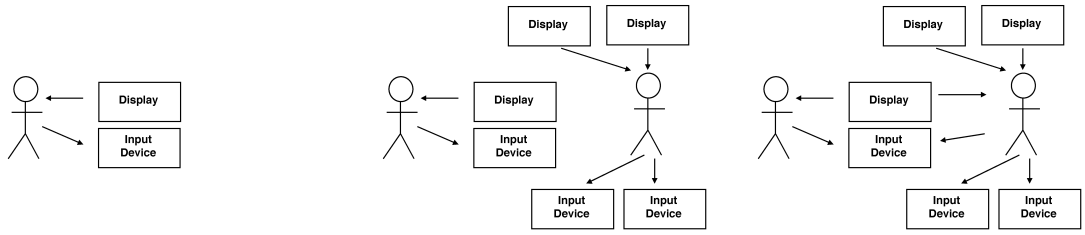
Illustrative Examples

To explain what is considered a user interface in UAR within this thesis, we briefly present two examples (some nice examples can also be found in [45]). The first example points out, in an abstract manner, the feature of multiple displays and input devices that can be employed dynamically by several users. The second example illustrates this feature by an example from one of our systems.

The first example is shown in Figure 1.4. The flow of events for this example:

(a) A user uses one input device and one display.
(b) Another user is approaching, using two input devices and two displays.
(c) The second user now additionally uses the input device and display of the first user.

The displays might be conventional displays, but could also be head-mounted displays or projectors that display virtual content directly on the real world objects that should be augmented. Similarly the input devices could be regular mice and keyboards, but also tangible



(a) (b) (c)

Figure 1.4: Example illustrating the dynamic behaviour of a user interface in UAR.

input devices, six-degrees-of-freedom (6DOF: three degrees for translation and three degrees for rotation) input devices or other exotic input devices.

An actual example of more complex I/O devices is shown in Figure 1.5. There are four displays involved: a projection on the table, a tracked laptop as a window to the virtual world, the head-mounted display the user is wearing and finally the head-mounted display through which this image was taken. The virtual sheep on the user's hand is an AR overlay. The user's hand is tracked, because he is wearing a bracelet with markers for a tracking system. The wooden sheep in the other user's hand is an example of a tangible input device that is tracked in 6DOF.

Summary

The main characteristics for user interfaces in UAR can be deduced from the union of the requirements for the contributing research paradigms:

1. Multiple displays, input devices and users
2. Mixed reality displays
3. Tangible interactions
4. Context-awareness

1.2 Challenges in Developing User Interfaces for Ubiquitous Augmented Reality

Development and deployment of user interfaces in UAR pose a variety of hard problems that are unique to UAR. The two main challenges addressed in this thesis are:

1. What does a software infrastructure have to provide to run user interfaces in UAR?
2. How can these user interfaces be designed and modified intuitively?

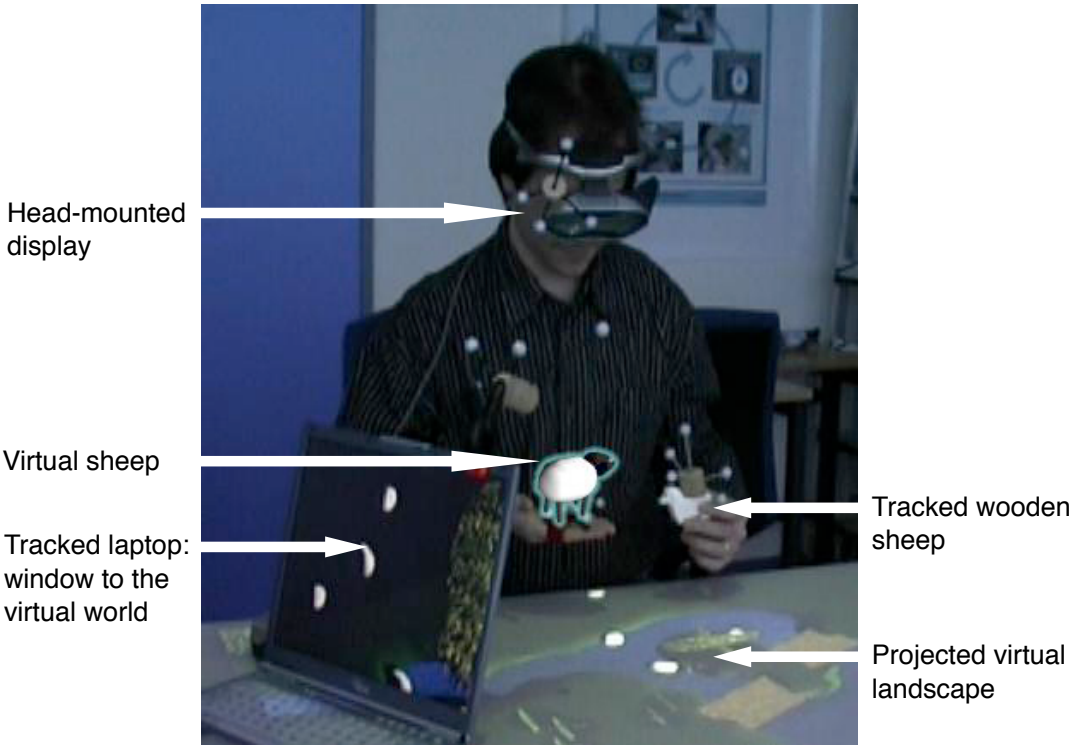


Figure 1.5: Example UAR user interface from the SHEEP game [100]. Videomixed view through another user's tracked, see-through, head-mounted display.

First, providing a runtime environment for user interfaces in UAR is not trivial. The expressiveness of UAR is powerful, since it combines a variety of user interface paradigms: ubiquitous computing suggests multiple users, input devices, and displays, as well as bidirectional multimodality. AR mixes the real and virtual world in realtime. It is interesting to note that the real world also includes content displayed on other displays. Tangible user interfaces support a wide range of input devices that might even provide input with 6DOF. Additionally, the highly dynamic nature of user interfaces in UAR introduces new problems: The processors controlling the I/O devices have to be connected via a network. These processors might be different and could run different operating systems. Furthermore, the dataflow connections between the processors can change dynamically during runtime and should require no or at least minimal user intervention.

Second, as opposed to WIMP user interfaces, the interaction elements in UAR have not yet been formalized. The development of these interaction elements requires interdisciplinary teams (psychologists, human-factors researchers, human-computer interaction experts, mechanical and electrical engineers, computer scientists, etc.). An experimentation environment

for collaboration in these interdisciplinary teams is desirable. What should such an experimentation environment look like?

1.3 Proposed Solution

My approach to these two problems is to first build a software toolkit to build and execute user interfaces in UAR and then to build several authoring tools on top of it to be able to quickly modify these user interfaces.

The software toolkit AVANTGUARDE[1]that I have created is based on DWARF. DWARF connects a set of software components that can be arbitrarily distributed across a set of machines. AVANTGUARDE is composed of DWARF components. Its architecture and components address the specific requirements for user interfaces in UAR.

On top of AVANTGUARDE, I have created several prototypical tools for interaction development for UAR user interfaces. Some of these tools follow the traditional WIMP metaphor, while others have a UAR user interface themselves. They enable adjusting a running UAR user interface executed in AVANTGUARDE; for example, to configure dataflow networks, to specify dialog control, or to create context-aware [151] animations.

Context of my Research

This research was conducted within a team of four other Ph.D. students. We have jointly developed the software infrastructure DWARF, which was used as a glue between our research products. As a result of this team effort, I had a sophisticated middleware and tracking infrastructure at hand. Both of these research products helped me enormously when developing my proposed solution.

Middleware and architecture. Thomas Reicher has developed most of the architectural concepts for DWARF. Details can be found in his thesis [137]. Asa MacWilliams implemented and refined these concepts. As a result, he has produced the middleware for DWARF. His thesis [101] contains more details.

Tracking. Martin Wagner applied DWARF's capabilities to the problem domain of tracking in UAR. His research results can also be found in his thesis [180]. Martin Bauer is also working in the area of tracking for UAR. However, compared to Martin Wagner, he is more concerned with the algorithms involved, than with the overall concept for tracking in UAR.

[1]The name was influenced by two ideas:

1. `http://m-w.com` defines avant-garde as: an intelligentsia that develops new or experimental concepts especially in the arts.
2. I wanted a name that contains the string UAR.

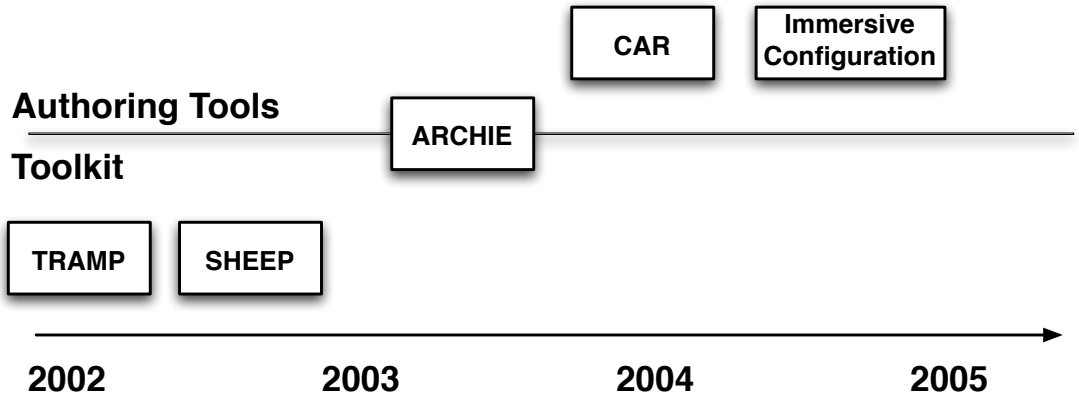


Figure 1.6: Timeline of projects realizing my proposed solution.

Development process. Together with Asa MacWilliams, I have worked on a development process called *development at runtime*. It is described in his thesis [101] in more detail. However, he has only applied it on a software engineering level; My thesis deals with its application to user interface development for UAR.

1.4 Hypothesis

My research hypothesis is:

> It is possible to develop a flexible, extensible and reusable software toolkit that can be used by programmers to quickly create user interfaces in UAR. To ease the task of developing new interaction elements for user interaces in UAR, authoring environments that allow the modification of a running user interface and facilitate the abilities of UAR can help to produce results even faster. As a proof of concept, I have developed the component-based toolkit Avantguarde, which provides a runtime environment for user interfaces in UAR and several authoring tools that are built on top of Avantguarde.

1.5 Approach

My proposed solution was realized within five projects. Figure 1.6 gives an overview of the goals of these projects and their timeline. More details about these and other projects implemented with Avantguarde can be found on the thesis' webpage [146].

1.6 Research Contributions

This thesis makes research contributions to the fields of software engineering, ubiquitous computing, augmented reality and human-computer interaction. These contributions can be divided into three groups:

1. This thesis defines and explores the new user interface paradigm UAR. We present, model, and reflect upon the challenges in developing and executing user interfaces for UAR. During this discourse, two novel models are developed.

2. This thesis describes the design and implementation of AVANTGUARDE. AVANTGUARDE is a reusable, extensible and flexible infrastructure, addressing the novel challenges of UAR user interfaces. AVANTGUARDE has been used to implement a wide range of UAR applications using a new development process.

3. This thesis presents a variety of prototypical tools for programmers and non-programmers to quickly build and adjust user interfaces in UAR while they are running. These lightweight, flexible tools can be combined on demand to support authors of UAR user interfaces.

1.6.1 Definition and Exploration of the UAR User Interface Paradigm

The initial definition of UAR from Section 1.1 will be refined in Section 2.1; furthermore, the novel challenges in UAR user interfaces are discussed. In the process of this discussion, two new models will be introduced that are applicable to user interfaces in UAR, ubiquitous computing, and augmented reality: Spheres of Influence and Superimposed Lenses.

The Spheres of Influence Model (Section 2.1.1) can be used to describe highly dynamical multi-device user interfaces. It applies ideas from the Spatial Model of Interaction [61, 21, 44] to the domain of UAR. Any software infrastructure that addresses this area has to be able to model Spheres of Influence.

The Model of Superimposed Lenses (Section 2.1.2) describes mixed reality in a multi-view environment. The well-known taxonomy of Milgram and Kishino [110] for isolated mixed reality views is put into a more general perspective. Since in UAR, several views according to their taxonomy can be superimposed, a new model is required.

1.6.2 Design and Implementation of the AVANTGUARDE Toolkit

To enable the execution of user interfaces in UAR, I have designed the component-based toolkit AVANTGUARDE [148] (Chapter 4). So far, it has been successfully used to build nine research prototypes (development time for each prototype: five to seven months) with and without my participation. This proves that it is an extensible and reusable software infrastructure.

The novel combination of a distributed framework (DWARF) with several well-known techniques (dataflow programming, user interface management systems and context-aware architectures) enables the creation of UAR user interfaces. This makes AVANTGUARDE the first infrastructure addressing UAR in an overarching fashion.

This thesis consequently applies the development at runtime process to user interface development for UAR, which embodies another contribution. Several decisions regarding the design of AVANTGUARDE have taken this process into account, since development at runtime requires a high degree of flexibility from a supporting software infrastructure.

The first system built with AVANTGUARDE and the development at runtime process is SHEEP [103]. Until now, no software infrastructure has supported the development of such a system. Additionally, the benefits of development at runtime for user interfaces in UAR has been verified with this system.

1.6.3 Design and Implementation of Authoring Tools

To ease the development of UAR user interfaces, several tools have been developed (Chapter 5). All of these tools apply the development at runtime idea. These tools are highly experimental, since they build on an experimental infrastructure and an experimental development process. We have not conducted any usability studies, yet. This makes it impossible to make claims regarding the usability of the tools—although we have anecdotal evidence about the benefits of using these tools.

The contribution of these tools is the novel approach that motivated their development: they create a toolbox of lightweight, flexible tools that can be combined by the user interface author as necessary. For every task that is addressed by these tools, it is possible to develop a tool following the WIMP paradigm, or a tool that uses the UAR paradigm—each with its own benefits and limitations. The appropriate combination of these tools maximizes the benefits for the author. This approach is novel to the domain of UAR, since most authoring tools for UAR are either completely WIMP-based (e.g., DART [96]), or completely in augmented reality (e.g., Lee and colleagues' tool [91]).

A combination of tools with different user interface paradigms has already been applied to authoring virtual environments in the SAVE system [66]. Depending on the task that the author wants to fulfill, he can choose between a WIMP tool, or a tool in virtual reality. These tools can even be used collaboratively by two authors. This basic idea has been extended by the work presented in this thesis in several ways: first, we apply it to UAR instead of only virtual environments. Second, our collection of tools covers a much larger area of tasks than the SAVE system.

The tasks our tools address are:

Monitoring the user. In developing user interfaces, it is beneficial to have feedback about how users employ it, as soon as possible. Two tools have been developed for this purpose: first, a WIMP tool that can collect and evaluate usability data during system runtime [87]. Second, a tool using a novel augmented reality visualization is presented [114] that makes

it possible to clearly see the visual attention of a user with a combination of head- and eyetracking.

Configuring dataflow networks. For adjusting dataflow networks, the WIMP tool DIVE will be presented. We have also created an immersive visual programming environment [147, 150] that addresses the same task. It extends the work presented by Lee and colleagues [91] in important ways: it can configure visualizations on multiple displays and is also able to interact with real-world objects.

Adjusting dialog control. To specify dialog control, the User Interface Controller Editor [64] is explained. It embodies a visual front-end for specifying dialog control—much in the spirit of a User Interface Management System [115]. However, it is the first user interface management system for UAR.

Creating context-aware animations Finally, a set of tools to experiment with context-aware mobile augmented reality user interfaces (Section 5.5) is presented. Especially interesting about these tools is that one WIMP tool is combined with several immersive tools, enabling a novel way of specifying context-aware animations.

1.7 Structure of This Document

Chapter 2 presents the inherent difficulties that stakeholders and programmers face when developing user interfaces in UAR. Chapter 3 discusses the approaches that have been used by other research groups to address these problems. AVANTGUARDE was developed to address these problems and is described in Chapter 4. The authoring tools that I have built on top of AVANTGUARDE are presented in Chapter 5. Their design space ranges from conventional graphical user interfaces to user interfaces in UAR. In Chapter 6, the results achieved by using my approach are discussed.

Additionally there is an accompanying webpage for this thesis [146]. It contains a full list of systems that have been built with AVANTGUARDE and the components that make up AVANTGUARDE. Additionally several movies, illustrating the presented user interfaces are provided. It is highly recommended to readers not familiar with these movies, to first watch them before reading on.

Challenges in Developing User Interfaces for Ubiquitous Augmented Reality

User interfaces in UAR provide ad-hoc distributed, hybrid visualizations and interactions. This chapter presents the inherent difficulties that stakeholders and programmers face when developing them.

This chapter explains the specific problems that arise in developing user interfaces for UAR. According to their main characteristics that have already been presented in Chapter 1.1, we discuss the implications for a runtime environment (Section 2.1) and for tools (Section 2.2) that support the development of novel interaction elements.

The contents of this chapter, summarized in Section 2.3, provide the foundation for the literature review in Chapter 3, which discusses how other researchers have addressed these challenges.

2.1 Runtime Environments for User Interfaces in Ubiquitous Augmented Reality

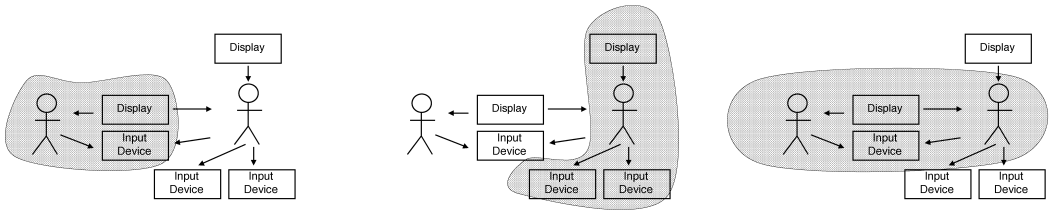
The nets are vast and infinite.

MASAMUNE SHIROW, *Ghost in the Shell*

In this section, we analyze in more depth the consequences for a supporting runtime environment that addresses the characteristics of user interfaces in UAR:

1. Support for multiple displays, input devices and users: Section 2.1.1.
2. Mixed Reality displays: Section 2.1.2.
3. Tangible interactions: Section 2.1.3.
4. Context-awareness: Section 2.1.4.

At the end of each section, we sum up the resulting requirements. They are divided into functional and technical requirements. Functional requirements refer to a black-box view of the system; for example, saying that a system should be able to add integers, would be a functional requirement. Technical requirements refer to a white-box view of the system; for example, saying that the system should use a stack to do this, would be a technical requirement.



(a) (b) (c)

Figure 2.1: Example illustrating dynamic behaviour of a user interface in UAR. Spheres of influence are shown as shaded ovals.

2.1.1 Multiple Displays, Input Devices and Users

User interfaces in UAR assume that users are surrounded by several input and output devices. A user can choose a subset of these devices to perform an interaction. In this section, I introduce the formal *Spheres of Influence* model to describe this potentially changing set of devices employed by a user. Its spirit is similar to the awareness model used in the MASSIVE system [61, 21, 44]. Finally, I sum up the resulting requirements.

The illustrative example from Section 1.1 can be seen as a dynamically changing tuple that contains three sets: ({users}, {input devices}, {displays}) that are involved in an interaction. I call these dynamically changing tuples *Spheres of Influence*. Figure 2.1 shows the Spheres of Influence for the illustrative example. To give another example for a Sphere of Influence, let us model my current interaction when writing this sentence. The tuple would be: ({Christian}, {Keyboard, Mouse}, {Eizo screen, Laptop screen}); however, this is a highly static example.

It is interesting to compare the Spheres of Influence model with the *Spatial Model of Interaction* [61, 21, 44]. In the Spatial Model of Interaction, each object has a *nimbus* and a *focus* that describes a medium-specific (e.g., graphics or sound) subspace. The focus refers to a subspace that an object is aware of, whereas the nimbus refers to a subspace within which other objects can be aware of it. Object A is aware of object B if and only if: B is in A's focus and A is in B's nimbus.

Since foci and nimbi are medium-specific, objects can have several nimbi and foci. For example, the visual focus of a user is cone-shaped, whereas the aural focus of a user looks like a distorted sphere. An object behind this user with sphere-shaped nimbi could lead to the user being aware of it through the audio medium; however, he is not aware of it through the visual medium.

There are several differences between the Spheres of Influence model and the Spatial Model of Interaction. First, the Spatial Model of Interaction was developed for pure virtual environments. Second, its main goal was to express the mutual awareness of objects, which is not an

issue in the Spheres of Influence Model. The main difference, however, lies in the mapping of objects to subspaces: an object in the Spatial Model has only one nimbus and one focus per medium, whereas the Spheres of Influence model allows one object to be simultaneously in several spheres. For example, a user who performs two different interactions at the same time (e.g., tactile and gesture input) would be in two spheres. To sum up, the Spatial Model is about *potential* interactions, whereas the Spheres of Influence model is about *actual* interactions.

To support these interactions, several functional requirements for a software infrastructure can be deduced:

Support for dynamically changing Spheres of Influence A software infrastructure for user interfaces in UAR has to address the concept of dynamically changing Spheres of Influence by allowing ad-hoc connectivity and automatic selection of appropriate input devices and displays.

Combination of mobile and stationary devices Resources in the environment have to be connected dynamically to resources a user might carry around (e.g., palmtop computers or wearables such as MITHRIL [38]). This implies a highly modular architecture, whose components should be dynamically reconnectable.

Multi-channel communication To address multimodality, a system has to be able to deal with several input channels. The user's intention has to be extracted from the input that is received over these channels (input multiplexing). Similarly, a multimedia based system has to have a coordination instance that distributes the content to be presented to the user, leveraging the available output channels (output demultiplexing).

These functional requirements imply a set of technical requirements that are desirable properties for a supporting runtime environment:

Flexible architecture Because of the fine granularity of components and the loose coupling between them, a high degree of flexibility is necessary.

Distributed components The components that compose a user interface in UAR can be a combination of local and remote devices. Distribution should be transparent to the components.

Adaptivity of dataflow networks With the inherent options for ad-hoc connections and reconfiguration of components, an inherently adaptive infrastructure is necessary.

Operating System independent To allow deployment among a variety of devices, an infrastructure should be independent of a specific operating system.

Programming language independent To foster the benefits of different hardware platforms, programmers should not be forced to use a certain programming language. For example on a small device, programming in an efficient, but low-level, languages like C may be desirable. On the other hand, on a stationary computer, higher level languages such as Python or Java may be more desirable. Although they are slower, they are much easier for programmers to use [135].

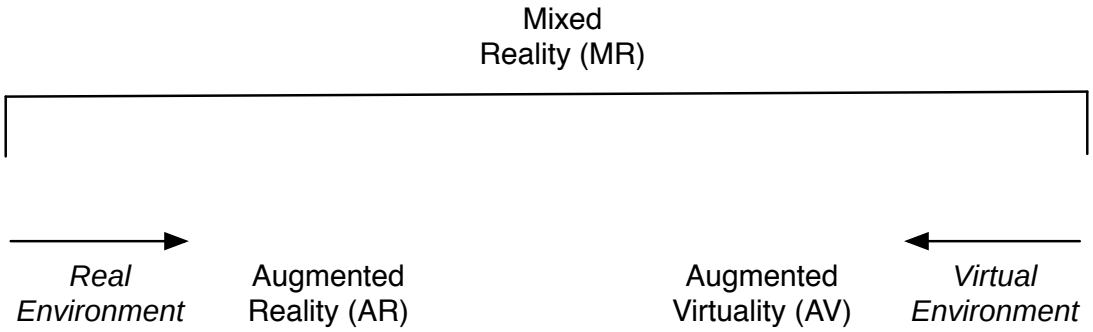


Figure 2.2: Reality-virtuality continuum according to Milgram and Kishino [110].

2.1.2 Mixed Reality Displays

In this section, we first discuss the relation between augmented reality, mixed reality and UAR. Then, we introduce the Model of Superimposed Lenses, which describes mixed reality in the context of UAR. Next, we discuss reference frames for augmentations. Finally, we sum up the requirements for mixed reality displays in the context of UAR.

Most people are more familiar with virtual reality (VR), where a user is completely enclosed in a purely virtual scene, (e.g., for flight simulation) than with augmented reality. Due to their nature, many of these systems operate in some sort of CAVE (Cave Automatic Virtual Environment: projectors are directed to four, five or six of the walls of a room-sized cube), or use head-worn displays tethered to stationary computers, thereby restricting the movement range of their user considerably.

Augmented reality systems, on the other hand, that *augment* the user's reality by adding virtual objects to it. Reality that is augmented in this sense conveys additional information, in effect enhancing the user's senses (e.g., to see a future building at the real, still empty construction site) whereas VR would only be able to show the future building but not in the context of reality. Different forms of augmentation are possible, including visual, audio and tangible.

A survey of augmented reality applications by Azuma [10] concluded in the following definition:

> AR systems have the following three characteristics:
>
> 1. Combines real and virtual
> 2. Interactive in realtime
> 3. Registered in 3–D

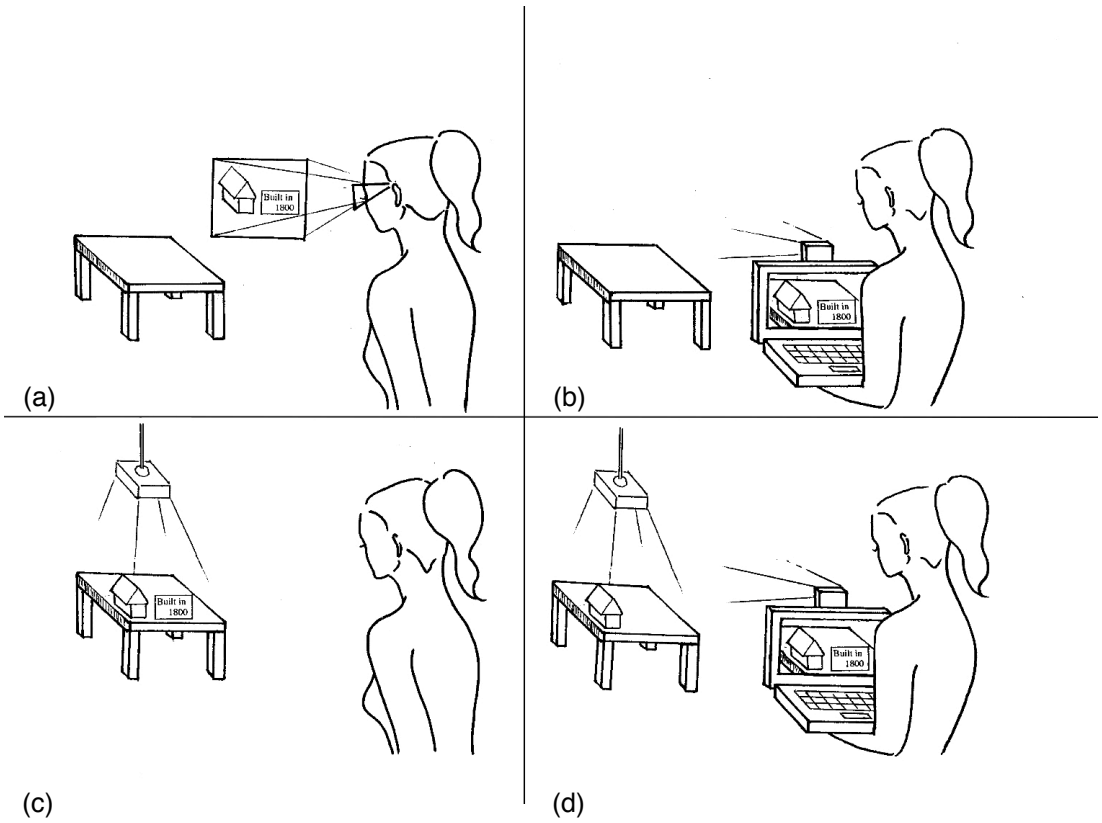


Figure 2.3: Mixed Reality in the context of UAR: a composition of lenses. House and annotation are both displayed on a: (a) head-mounted display, (b) see-through laptop, and (c) projector. (d) When two lenses are involved, augmentations can be distributed among them (i.e., the house is a projection and the annotation is on the see-through laptop).

Milgram and Kishino have identified the 'reality-virtuality continuum' (Figure 2.2, [110]) which connects completely real environments to completely virtual ones. Mixed Reality (MR), which includes augmented reality refers to everything in between fully virtual and fully real environments. Moving from left to right the amount of virtual imagery increases and the connection with reality weakens. 'Augmented Virtuality' refers to mostly virtual display environments, either completely immersive, partially immersive, or otherwise, to virtual environments to which some amount of (video or texture mapped) 'reality' has been added. An example visual augmentation is shown in Figure 2.4—the figurine on the marker held by the woman is a virtual character.

In user interfaces in UAR, several views can be superimposed. I introduce the *Model of*

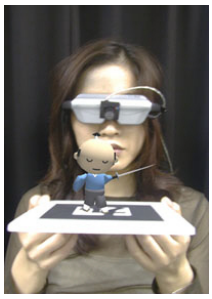


Figure 2.4: An Example of augmented reality [68] with an object-fixed reference frame: the virtual figurine sticks to the marker.

Superimposed Lenses (inspired by the theoretical discussion in [45] and examples in [18]). I claim that a more precise definition of MR in the context of UAR is that it is a composition of lenses. Figure 2.3 illustrates this model. The user looking at the table is involved with three lenses that provide augmentations: the head-mounted display, the see-through laptop showing a video image of the scene on the table with additional augmentations and finally the projector that can also provide augmentations. Figure 2.3(d) shows that the augmentations can be distributed among the lenses—the house is augmented by the projector, the descriptive text by the see-through laptop.

An important issue is the reference frame for augmentations. Feiner and colleagues [46] identify four reference frames (in my terminology: world-fixed, head-fixed, body-fixed and object-fixed), but other reference frames are possible (e.g., hand-fixed or gaze-fixed reference frames). Figures 2.4 to 2.9 on the following pages give examples for six possible reference frames. However, these are just illustrative examples; the number of reference frames is potentially unlimited, as virtual content can be attached in any relation to real objects:

Object-fixed reference frame. Augmentations are attached to freely movable objects (Figure 2.4).

World-fixed reference frame. Augmentations are at a constant position relative to the real world. This type of augmentation is typically used for annotating real-world objects with a fixed position (Figure 2.5).

Body-fixed reference frame. Augmentations move with the body of the user. Typically this type of augmentation is used for task-independent tools that the user carries around and accesses on demand. In virtual environments, this concept is called physical mnemonics [111]. Figure 2.6 highlights this idea. Additionally, it introduces another idea: the user's head gaze triggers a zoomed display of the currently observed item.

Hand-fixed reference frame. Augmentations move with the hand of the user. Figure 2.7 shows a user discussing a virtual map with another user. To observe the map from different

angles, he can pick it up from the body-fixed toolchest around his belt and put it in his hand.

Head-fixed reference frame. Augmentations move with the head orientation of the user. This type of augmentation is typically used for information that is relevant to the user's current task. Figure 2.8 shows a map of the environment that supports a navigational task.

Gaze-fixed reference frame. Augmentations move with the user's eye gaze. Its applicability is similar to the head-fixed reference frame—however, it is meant for information that has to be even more in the user's focus of attention. Figure 2.9 shows a virtual compass that is constantly displayed in the center of visual attention.

To sum up, here is a list of the functional requirements for providing mixed reality visualizations within an UAR environment:

Mixing real and virtual The virtual content augments the real world. This can be achieved by a variety of techniques, (e.g., video see-through or optical see-through head-mounted displays or projections).

Support for real-virtual continuum It is required to adjust the degree of augmentation along Milgram and Kishino's continuum.

Support of handling of superimposed lenses Several displays should be superimposable.

Different reference frames for augmentations The virtual content should be attachable to arbitrary reference frames.

On the technical level, the requirements are:

Realtime Azuma's definition requires realtime to contrast it to offline augmentations that can often be seen in Hollywood movies. Within this thesis, a user interface is considered realtime, when it provides an update rate of at at least 25 Hz.

6DOF tracking To provide correctly spatially registered augmentations, the user's viewpoint and relevant objects have to be tracked in 6DOF.

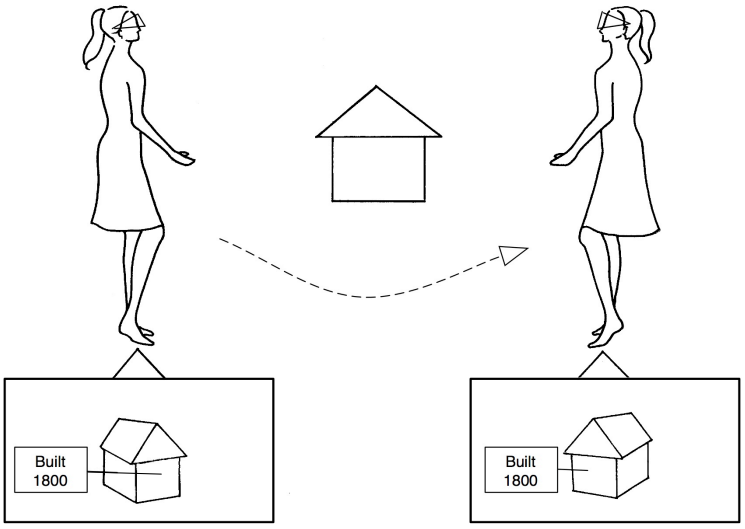


Figure 2.5: World-fixed reference frame: Augmentations are at a constant position relative to the real world. The example shows the annotation of a building.

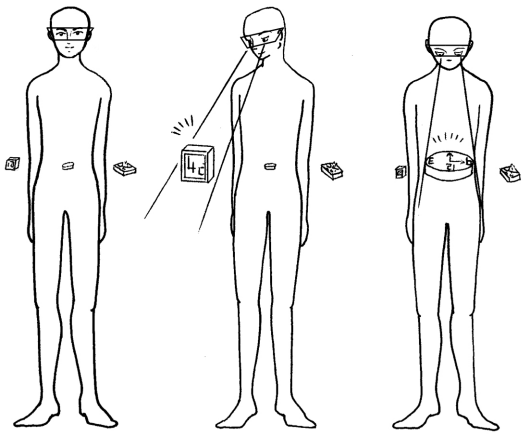


Figure 2.6: Body-fixed reference frame: Augmentations move with the body of the user. This drawing also introduces another idea: the user's head gaze triggers a zoomed display of the currently observed item.

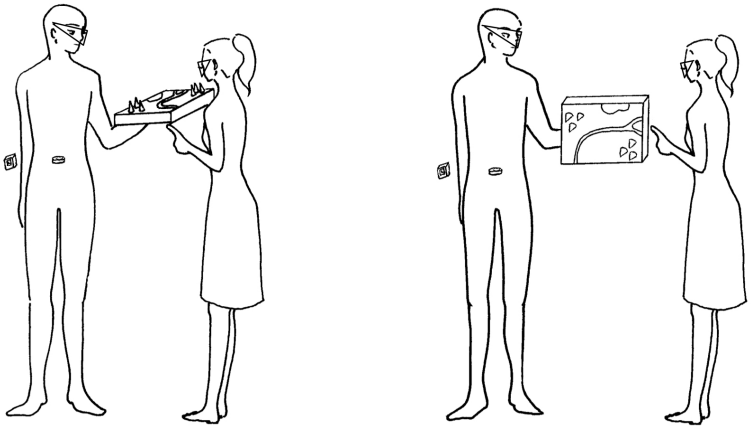


Figure 2.7: Hand-fixed reference frame: Augmentations move with the hand of the user. This example shows a user discussing a virtual map with another user. To observe the map from different angles, he can pick it up from the body-fixed toolchest around his belt and put it in his hand.

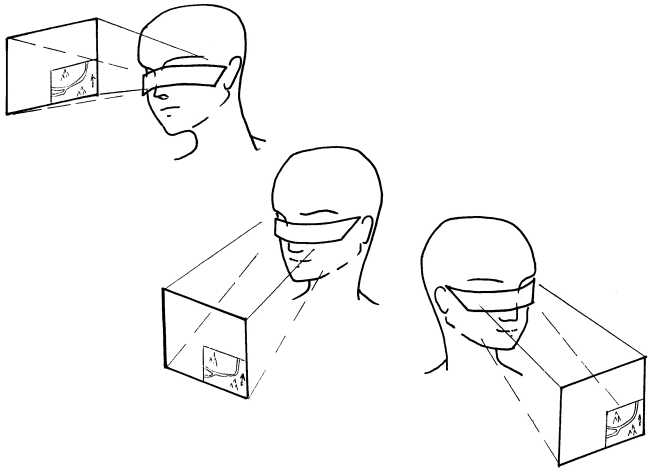


Figure 2.8: Head-fixed reference frame: Augmentations move with the head gaze of the user. The example shows a map of the environment that supports a navigational task.

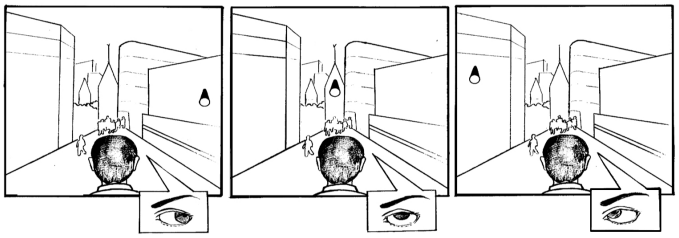


Figure 2.9: Gaze-fixed reference frame: Augmentations move with the user's eye gaze. The example shows a virtual compass that is constantly displayed in the center of visual attention.

2.1.3 Tangible Interactions

Next-generation interfaces like tangible user interfaces take a very different approach than that of traditional graphical user interfaces. Ishii coined the term *tangible user interfaces*, which he later formalized in the MCRpd TUI interaction model [177] (see Figure 2.10).

Traditional graphical user interfaces share a clear distinction between the visual representation (or *view*) provided by the graphical display and the *control* capacity mediated by the mouse and keyboard of the UI. A standard design pattern for implementing the coupling of view and control in traditional GUIs is the MVC pattern (Model-View-Controller pattern [52]).

This border between view and control blurs for tangible user interfaces as expressed in the model-control-representation (physical and digital), or MCRpd. The *view* notion has here been replaced with the notion of a physical representation (*rep-p*) and a digital one (*rep-d*), highlighting the TUI's integration of physical representation and control.

An example of these user interfaces is 'Illuminating Clay', a system for real-time computational analysis of landscape models, as seen in Figure 2.11. Here, users alter the topography of a clay landscape with their hands, which changes the projected result of a landscape analysis function in realtime [127].

To sum up, the requirements for tangible user interfaces in the context of UAR are:

Physical application domain objects act as tactile input The constraint to physical application domain objects is important, otherwise mouse input could be considered tangible input.

These objects have to be tracked in three dimensions Since mixed reality already requires objects to be tracked, tangible interactions come as a natural supplement.

These objects are coupled to digital representations The original MCRpd model requires the objects to be coupled to digital representations—in our context, the digital representations are augmentations most of the time.

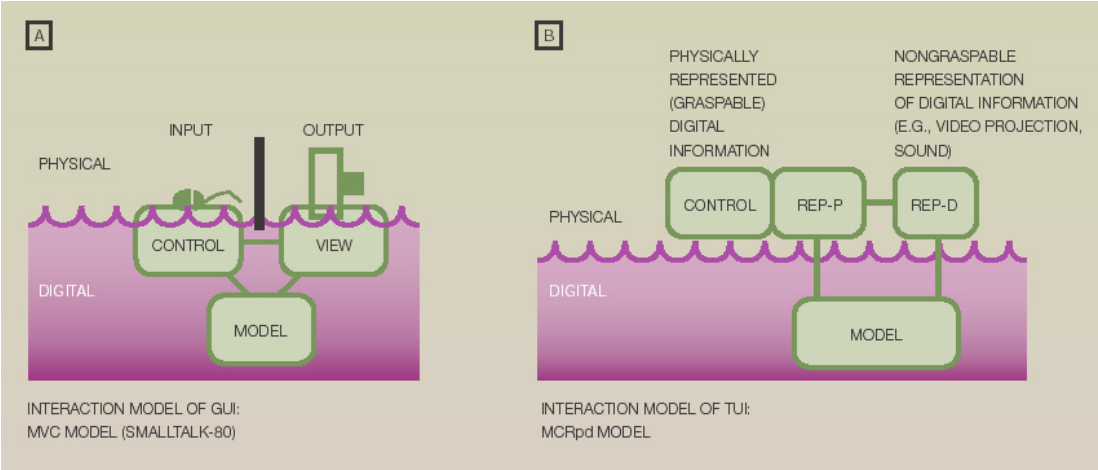


Figure 2.10: Tangible MCRpd Model [177].

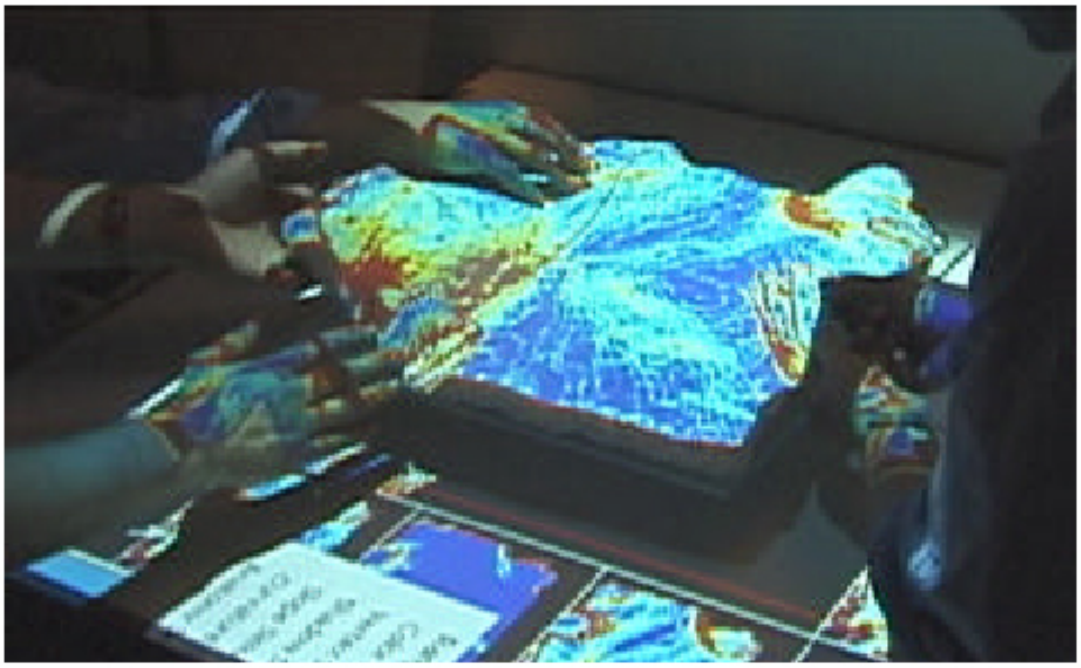


Figure 2.11: Illuminating clay: Users' hands interacting with physical and digital representation [127].

2.1.4 Context-Awareness

Context-awareness [151] is one of the key issues in ubiquitous computing research. To support users in their everyday tasks, the inference of the user's context is a desirable goal. For example, how can the current task of the user be determined without actually asking her? Several research prototypes have demonstrated that such inference is possible and desirable. As a result, the 'calm computing' paradigm that underlies the vision of ubiquitous computing [153] seems feasible. What are context and context-awareness exactly? Dey [39] defines them as follows:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.
> A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

We can conclude that context can be basically anything, depending on the user's task. If this information is used by an application, to change the way information is presented to the user, the application is context-aware.

Context-aware applications can be structured in three layers [180]: First, sensors measure properties of the user's environment. Second, these measurements are interpreted to provide contextual information. Finally, this information is used to trigger actions.

In the context of this thesis, we focus on user interfaces. Thus, our focus is on the way actions are triggered according to changes in the user's environment. A very important prerequisite for experimenting with context-aware user interfaces is the ability to easily simulate these changes in context. However, we are not concerned in our work with how context is derived from the measurements of the sensors. To sum up, the functional requirements for an infrastructure supporting context-awareness are:

Simulation of context data Because we do not want to deal with the tedious task of extracting context data from sensor measurements, it is sufficient to simulate the contextual data.

Interpretation of context data Once the measurements are interpreted, context data is available. This context data has to be interpreted again to trigger actions in the user interface.

User Interface should be influenced by context data When changing context trigger actions, they should cause changes to the user interface.

2.2 Tools for Developing User Interfaces in Ubiquitous Augmented Reality

Design like an analyst,
analyze like a designer.

ANONYMOUS

Before we discuss how tools should support the typical tasks involved in developing user interfaces for UAR (Sections 2.2.1 to 2.2.4), we first discuss the context in which these tools are supposed to be used. The *moving target problem* [112] was the starting point for our thoughts, which ultimately led to the invention of a new development process: *development at runtime* (also known as *Jam Sessions*[1]).

The moving target problem states, that inflexible software infrastructures for experimental user interfaces often fail. Since the user interfaces to be created are not known in advance, flexibility is a key issue for a software infrastructure supporting experimentation. User interfaces in UAR are especially prone to that problem, because it is such a young field.

The remainder of this section gives an overview of development at runtime. More details on this approach (including a comparison to standard methodologies) can be found in Asa MacWilliams' thesis [101].

Because UAR is still a young field, one of the main activities in user interface development for UAR is experimentation with different interaction elements. These elements have to be designed, implemented and evaluated. An important research issue here is to establish a development methodology that covers these three sub-activities and links them together more closely.

The main groups of developers participating in the development of user interfaces in UAR are (Figure 2.12) :

Programmer The programmer changes the implementation of the user interface by writing and editing low level code.

3D Designer This type of designer is concerned with creating 3D interaction and presentation elements which appear in the user interface.

Screen Designer The focus of this designer is the actual screen layout, that is what is presented to the user at which location on screen. However, in the domain of UAR, the activities for a 2D screen designer are clearly limited to content that is independent of spatial relations between the user and her environment (e.g., head-fixed content). Other content (e.g., world-fixed content) requires knowledge in 3D geometry, since what will be displayed in a head-mounted display is merely a projection of the 3D object to the image plane.

Interaction Designer This designer fine-tunes the interactions the user can experience. She will set which multimodal inputs trigger which actions.

[1]The name Jam Sessions was inspired by the spontaneous collaboration of Jazz musicians that is also named Jam Sessions. However, in our case we collaborate on user interface elements, instead of music.

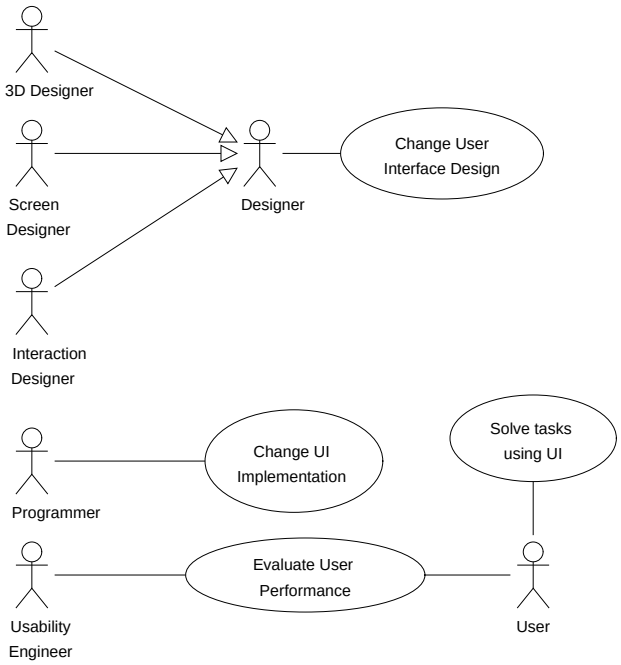


Figure 2.12: Use cases of all basic participating groups in the process (UML Use case diagram).

Usability Engineer The quality assurance of the usability of the user interface is the main point of interest of the usability engineer. This person combines all the roles of conducting usability studies in one. He selects, briefs, and debriefs the users for the study, prepares the evaluation materials, conducts and logs the actual study and finally analyzes and evaluates the results. However, for practical reasons, the tasks of the usability engineer are often addressed by a group of usability engineers.

User The user actually uses the user interface by navigating through it in an attempt to accomplish certain tasks. For example she might want to place a roof on a building she is constructing within an architectural augmented reality application.

In old-fashioned user interface development, a waterfall or an extended waterfall process (Figure 2.13) is applied [105]. Starting with the phase design, it proceeds to the phase implementation and finally a phase evaluation. These phases are run sequentially with no or little room for feedback or dependencies. This type of process can work well if the user interface design space is well known.

For traditional desktop software, vast amounts of knowledge on usability data exist, which created extensive standard guidelines [155] that limit the design space to a known usable and working subset. Since user interfaces in UAR are a comparably new development, such a

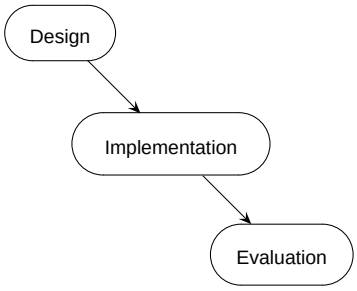


Figure 2.13: Basic waterfall process (UML Activity diagram).

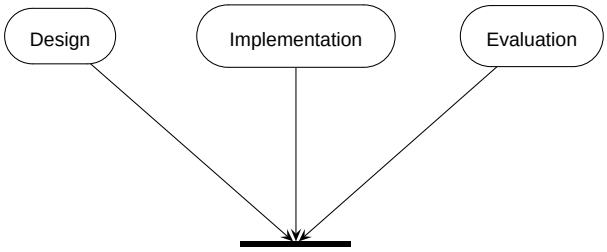


Figure 2.14: Proposed parallel process: development at runtime. (UML Activity diagram).

knowledge base is still to be built. So, for now, we are confronted with a vast design space and an uncertainty about which designs will work and which will not.

Traditional user interfaces can also benefit from clear nonfunctional requirements. For example, a web-site has to be navigable, which is defined in web-style guides together with all other non-functional requirements that are proven to be sufficient. But which non-functional requirements do we have to impose on user interfaces in UAR? The graphical portions of the user interface should probably be concise, but what exactly does this mean?

To make up for these uncertainities, a better process offering much more feedback between phases is necessary. In fact we believe that only maximizing this feedback can offer us enough efficiency until our knowledge base is large enough to allow older, slower paced, sequential processes.

To maximize this feedback, we propose to run all three phases (design, implementation and evaluation) in parallel (Figure 2.14). We have already learned a few valuable lessons regarding the process within the earlier SHEEP project [103]. In *Jam Sessions*, development takes place at system runtime. This allows a group to work with peer code reviews and on-the-fly insertion of altered system components, for quick integration tests. These sessions proved to increase the development speed significantly. This process also allows playful exploration, because sub-components that have an impact on the user experience can be swapped during run-time, enabling quick assessment of different variations. Iterative, continuous development

is an implication of this. This process is only feasible with a highly flexible infrastructure (e.g., our DWARF framework).

We are now revisiting the functional requirements presented in Section 2.1 to explain typical development tasks that are ideally supported by yet to be developed tools.

2.2.1 Multiple Displays, Input Devices and Users

To develop user interfaces for UAR in the context of multiple devices, displays and users, an interaction designer and later a programmer have to deal with specific problems.

Which input devices and displays When designing interactions with multiple devices it is crucial to think about which combination of input devices and displays to use. Ideally, the developer can quickly try out how an interaction feels using different input devices and how it looks like on different screens.

Orchestration of devices and displays Once the set of input devices and displays is determined, the next question concerns the *orchestration* of these devices. In multimedia output design and multimodal input design, it is crucial to find the best way to have all involved devices work together. For example, if an interaction requires inputs A and B to happen one after another, the question for the timeout interval is crucial: how long after A can B happen? We think, it is best tried out, while actually performing the interaction.

2.2.2 Mixed Reality Displays

The coordinated display of mixed reality content on several screens requires the developers to deal with very specific questions:

Look of virtual objects A crucial question for virtual content with which the world should be augmented is their visual appearance. This is different from WIMP user interfaces, because of the novel display devices used in UAR user interfaces (head-mounted displays, retina displays, see-through laptops and projectors)—the virtual objects might have to look different on different devices. For example, projections are often color-corrected, since the color of the surface on which it is projected, has to be taken into account. This is not the case for head-mounted displays that are typically only calibrated for distortions.

Behaviour of virtual objects The dynamic behavior of virtual objects is another difficult question. Apart from regular animations, there can be dynamic changes to an object's reference frame or the device it is shown on.

Types of lenses As already explained in section 2.1.2, display devices for augmentations can be seen as lenses that change the impression of the world seen through them. It is very difficult to decide which spot on the real-virtual continuum should be chosen for a visualization. Often it is dependent on the user's task how much immersiveness is acceptable for him. For example while driving a car, at high speeds, augmentations that consume little screen space are necessary; whereas at low speeds or while the car stands still, augmentations consuming more screen space might be applicable.

Mapping of virtual objects to devices When a user looks at a real world object and several lenses are in his line of sight, it is an interesting question on which lenses to put the augmentations. This can be dependent on many factors (e.g., resolution of the devices or tracking accuracy for them).

Allocation of augmentations to reference frames In augmented reality applications, objects are registered in 3D [10]. Therefore, after completing a screen design, using a 2D tool, the designer usually needs to map the screen's objects to 3D. She might have decided to keep a presentation component, keeping the user up-to-date on an important variable in close reach, head-fixed [46] in the left right corner, all the time. Currently, the screen designer has to recreate her earlier 2D design in 3D using a completely different tool for mapping 3D objects. It would be much more efficient if she could instead import her 2D design into a 3D registering application. But this is not possible without much better inter-tool integration.

2.2.3 Tangible Interactions

When designing tangible interactions, there are two areas of concern:

Types of physical objects To provide users with efficient and intuitive ways to interact with a user interface, the physical appearance of tangible input devices is very important. Human factors researchers and industrial designers are concerned with the design of everyday things. It would be very convenient to allow these researchers to quickly try out different physical form factors of tangible objects.

Function for coupling When tangible objects are coupled with virtual objects, an important question is what the coupling function should be. For example, Poupyrev has experimented with non-isomorphic coupling functions [131, 132]. Additionally, it is common practice already in consumer electronics to provide non-isomorphic input mappings. Two examples are: first, using the iPod's scroll wheel [6] (the longer the wheel is turned, the faster the selection moves); second, when setting the time on a modern digital alarm clock, the longer you keep the button pressed, the faster the time advances.

2.2.4 Context-Awareness

For context-aware user interface design, two questions have to be dealt with by the designers:

Implications of changing context to user interface The calm computing paradigm supposes that changes in context should imply subtle changes in the user interface. To make these changes unobtrusive and transparent to the users is a challenging task that requires the interaction designers to be able to quickly try out different alternatives for how the user interface is modified in response to changes in the user's context.

Simulation of context data To actually carry out these experiments, a convenient way to simulate contextual data is required.

2.3 Reflections

In this chapter, we have presented the requirements for an infrastructure for user interfaces in UAR and the challenges involved in developing these user interfaces. Both of these will serve as a foundation for the literature review in the next chapter. The infrastructural challenges (summarized in Figures 2.15 and 2.16) will also be useful to understand the decisions in developing our infrastructure, AVANTGUARDE, which will be described in detail in Chapter 4. Similarly the challenges regarding tools (see Figure 2.17) will serve as a foundation for the description of the tools in Chapter 5. Before we conclude this chapter by listing open questions regarding our experimental development process, we would like to present some meta requirements for a runtime environment and authoring tools for user interfaces in UAR.

In addition to the specific requirements for a runtime environment listed in Figures 2.15 and 2.16, the typical software engineering requirements should not be forgotten:

Reusability Developed user interface elements should be easy to reuse in other systems.
Modifiability It should be easy to change the properties of a running user interface.

Figure 2.17 lists the functional requirements and maps them to typical tasks a developer has to do. We propose to support developers doing these tasks by realizing these key ideas:

Authoring during system runtime. Immediate feedback helps to minimize uncertainties.
Immersive authoring. Since we want to develop user interfaces for UAR, it is naturally to consider tools that have an UAR user interface themselves.
Providing tools for non-expert users. As there are many different user groups with varying backgrounds involved, we would like to provide them with easy to use tools.
Collaborative development. The various groups involved in developing a user interface can produce better results by collaborating.
Minimization of turnaround times. The less time it takes to change a user interface, the faster results can be produced.

Open Questions

There are a number of open research issues regarding tools that support building user interfaces in UAR:

Which tools There are numerous paths to take in supporting the three main user groups, resulting in a large design space for tools. It is a challenge to gain clarity regarding which type of tools will have the largest benefit.
Tool integration By integrating tools with each other, a much better work-flow between these tools can be achieved, resulting in better results. Where are the limits to integration and which integrations are reasonable at all?
Tool mapping Some tools might be useful to more than one user group thanks to a high level of integration. The presentation of multiple tools simultaneously to certain user groups

- Support for dynamically changing Spheres of Influence
- Combination of mobile and stationary devices
- Multi-channel communication
- Component for input multiplexing
- Component for output demultiplexing
- Mixing real and virtual
- Support of real-virtual continuum
- Support for handling of superimposed lenses
- Different reference frames for augmentations
- Physical application domain objects act as tactile input
- These objects are coupled to digital representations
- Interpretation of context data
- Simulation of context data
- User interface should be influenced by context data

Figure 2.15: Functional requirements for a runtime environment for user interfaces in UAR.

- Flexible architecture
- Distributed components
- Adaptivity of dataflow networks
- Operating System indepent
- Programming language independent
- Realtime
- 6DOF tracking

Figure 2.16: Technical requirements for a runtime environment for user interfaces in UAR.

Functional Requirements	**Parameters to change during development**
Support for dynamically changing Spheres of Influence	Which input devices and displays
Support for dynamically changing Spheres of Influence	Orchestration of devices and displays
Mixing real and virtual	Look of virtual objects
Mixing real and virtual	Behaviour of virtual objects
Supporting real-virtual continuum	Types of lenses
Supporting handling of superimposed lenses	Mapping of virtual objects to devices
Different reference frames for augmentations	Allocation of augmentations to reference frames
Physical application domain objects act as tactile input	Types of physical objects
These objects are coupled to digital representations	Function for coupling
Interpretation of context data	Implications of changing context to user interface
Simulation of context data	Actual context data

Figure 2.17: Typical tasks for developers trying to change the functionality of a user interface in UAR.

might have more value than the sum of each single tool on its own merit. It is a challenge to figure out which tool combinations map best to which user groups.

Tool automation The more knowledge on UI design is accumulated, the more ideas for automation features in tools can be generated. For example, basic clear-cut design principles that have been shown to apply in certain scenarios could be enforced in design tools. Since we still lack knowledge in this area, it is unclear which automations will be indeed feasible in the future. Instead of testing against known usability problems, there have been interesting approaches in web interface development, such as WebRemUSINE [122], which try to automatically identify new usability problems. This is done by looking at the level of correspondence between how users actually perform tasks and the intended system task model. This idea might also be applicable to AR user interfaces.

Similarly, on the process side, there are some open issues that remain to be answered:

Limit to parallelism By conducting multiple different development phases at the same time much better feedback can be attained. But how parallel can the process get without losing reasonability? The different phases of the process undeniably have certain dependencies that will probably not allow totally parallel execution.

Formal process Only by obeying a formal process similar to Extreme Programming [16], built on reasonable rules and process patterns [54], can highly parallel development be accomplished. But which practices are best for AR user interface development?

Persistence of UI experiments It is in the nature of rapid proto-typing to experiment with different variations of the UI in quick succession. However, after testing a number of UIs, it is very desirable to be able to roll-back into a previously evaluated UI iteration since it may have turned out to be best suited after all. It is a challenge to build the process in a way to ensure that the results of these prior experiments are not lost.

Literature Review

Software infrastructures and authoring tools ease the task of developing user interfaces for UAR. Approaches used by other research groups are presented and discussed.

This chapter gives an overview of related work that has been done in the area of runtime environments and authoring tools for user interfaces in UAR. These two research areas are closely coupled, since every authoring environment needs a runtime infrastructure to execute the authored user interfaces. Similarly, runtime environments often come with authoring mechanisms. The requirements presented in Chapter 2 serve as a conceptual framework for the discussion in this chapter.

In Section 3.1, the building blocks for runtime environments for user interfaces in UAR are discussed. These range from heavyweight approaches, such as user interface management systems, to lightweight approaches, such as class libraries. Several research projects have tried to combine several of these building blocks into an overarching infrastructure, since these different approaches are not mutually exclusive at all, but can instead complement each other.

In Section 3.2, the building blocks for an authoring environment for user interfaces in UAR are examined. They are classified according to the type of user interface they employ to address the authoring challenge. First, we discuss tools that use a classical desktop user interface, then we proceed to tools employing a tangible interaction metaphor, and finally we present tools that fully immerse the user.

Finally, in Section 3.3, the results of the literature review are summarized, and a detailed explanation of their relation to my approach is given. This lays the foundation for the explanation of our runtime environment (Chapter 4) and authoring tools (Chapter 5).

3.1 Runtime Environments

We start the examination of runtime environments by introducing a generic functional decomposition of user interfaces in UAR. After explaining the functionalities of its parts, we map these to several technical approaches that can be used to address these functionalities and discuss the relation between these approaches. The remainder of this section presents related work that employs these approaches:

1. Distributed frameworks: Section 3.1.1.

2. Dataflow architectures: Section 3.1.2.
3. User interface management systems: Section 3.1.3.
4. Component-based frameworks: Section 3.1.4.
5. Class libraries : Section 3.1.5.
6. Scripting languages : Section 3.1.6.

We propose a generic functional decomposition of the functional requirements presented in Section 2.1 for a runtime environment for UAR user interfaces. A large number of augmented reality frameworks have recently been analyzed (see [138]). As these frameworks partly support UAR, the findings made in that analysis also support our generic functional decomposition.

Figure 3.1 (inspired by [104]) shows the relevant subsystems and components within them. We explain the functionality of the subsystems and their components, starting from the top left corner and proceeding clockwise. It is important to note that the subsystems are general purpose and generic. However the presented components are just examples.

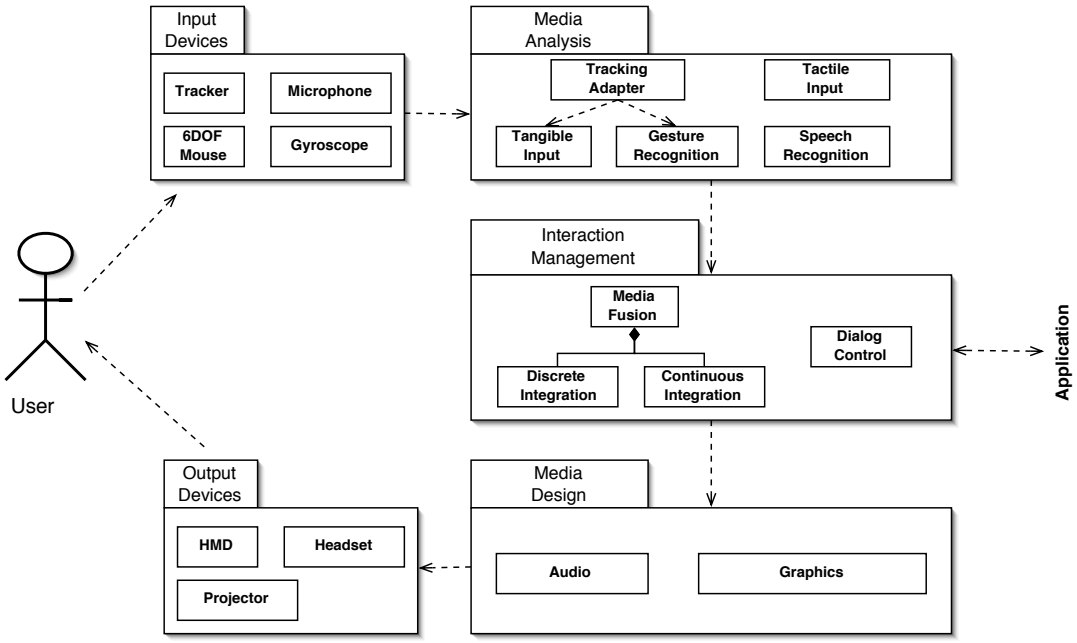


Figure 3.1: A generic functional decomposition of user interfaces in UAR.

The *Input Devices* subsystem contains input devices that are used to receive commands from the user. Each of these devices offers an individual input modality to be evaluated by the multimodal user interface.

Media Analysis is the process of turning physical user input into abstract tokens [119] handed over to the subsequent parts of the system – this can be compared to the task performed by

the lexical analysis of a compiler. Separate classes, such as gesture analysis, speech analysis, and tangible input analysis deal with the specific properties of different input modalities of the input devices.

The *Interaction Management* subsystem determines which output is presented to the user. Current challenges for interaction management are performance, flexibility, adaptivity, usability and efficient error management. The *Media Fusion* component takes the tokens of several input channels and infers user intention from them. In this component, two different ways for combining different input channels under respective boundary conditions are considered. *Continuous Integration* combines tokens that can take real values in a certain range. For example a rotation around one axis can take an infinite number of different values between 0 and 360 degrees. Example input devices that deliver these kinds of tokens are mice, trackers and gyroscopes. *Discrete Integration* refers to the integration of devices such as certain speech recognition systems that deliver only discrete values like the word that was recognized. Finally, the *Dialog Control* component selects the presentation medium and what to present in it.

The software components that present content to the user over any of the cognitive channels, (e.g., visual and aural) are contained within the *Media Design* subsystem.

The *Output Devices* subsystem renders the signal on the specified output devices. For multimedia based systems, several output devices are used at the same time.

The mapping of the technical approaches, that are presented in the remainder of this section, to the generic parts is:

Distributed frameworks The data exchange between the various parts of a user interface in UAR is often handled by a distributed framework.

Dataflow architectures A dataflow architecture is a specialization of a distributed framework. It is not aimed at connecting arbitrary components, but instead eases the combination of a set of lightweight filters. The continuous integration subsystem requires the setup of dataflow networks to couple real objects to virtual objects in a complex manner (e.g., through non-linear input processing [129, 161]).

User interface management systems The discrete integration subsystem and the dialog control subsystem can be handled by user interface management systems [115]. This approach tries to enable experimentation with user interfaces by providing a high-level abstraction, similar to the way Database Management Systems provide an abstraction to data.

Component-based frameworks A popular way to realize the overall runtime environment is to split the functionality into components. The components can be reused between applications, but they have to be configured differently according to the needs of the application.

Class libraries The generic parts of a user interface in UAR are often written with the help of class libraries that encapsulate reoccurring functionality. Class Libraries are the lowest level approach discussed in this section, since they encapsulate functionality on a much more fine grained level than component-based frameworks.

Scripting languages At various points in our reference architecture, scripting languages can be employed. Similar to class libraries, they are a very general approach and can be used in a wide range of tasks. Since scripting languages are interpreted languages, they minimize

turnaround times. Also they are easier to learn and use than compiled languages and code tends to be shorter. They prove to be a sensible extension to the approaches discussed before.

3.1.1 Distributed Frameworks

As we have already pointed out in Chapter 2, the various parts of a user interface in UAR are often distributed over several machines. The dataflow between them can be handled by a distributed framework. The core questions that we examine in this section are: How do the distributed parts find each other? How do they communicate with each other once the communication is established? Is the distribution flexible (i.e., can ad-hoc connections to new devices in reach be established dynamically)?

The simplest approach is to use a *client-server* solution. Most functionality is kept on a server, the clients connect explicitly to that server and use the desired functionality on the server. Examples of this approach are the Bat system [113], ARVIKA [183], or Piekarski's framework [126]. The advantage of this approach is its simplicity; however, flexibility becomes a big concern. In addition, direct communication between clients is not possible. This can lead to server overload and unnecessary latency.

A more flexible approach is to use a *blackboard* architecture, or its successor, a *whiteboard* [25]. Their core idea is, that several producers write their data onto a shared, distributed memory (the board), and consumers interested in that data pull the data from the board. For whiteboards, multiple processes operate in parallel on the server's data. This differs from blackboard systems where, typically, a blackboard manager directs control to access the board. Examples of these architectures are MIThril [38] (whiteboard) and BEACH [169] (blackboard; based on Coast [32]).

A related idea is a *tuplespace*, as used in Stanford's iRoom [82] project. Producers can write data into the tuplespace, which acts as a distributed shared memory. Then, clients choose which objects they want to read by using template matching. A tuple can be thought of as a set of attributes that are used as a template for matching. An example of a query is: `(type=PoseData, object=Christian's head)`.

Another approach is to split the communication of distributed components into two phases: first, connections are set up; then, the relevant data is passed along these connections. For setting up the connections, the appropriate producers have to be selected by the consumers. One mechanism to do this is to use *attributes* to specify the capabilities of a producer. Then, consumers can select the appropriate consumer with *predicates*. This approach can be found in Fluidum [42].

Last, we discuss a completely different approach: *distributed scenegraphs*. Scenegraphs are a fundamental data structure in computer graphics that store a set of objects to be rendered in a graph structure. Frameworks that are scenegraph-based typically compute the changes to the scenegraph on one machine and then replicate these changes to the other scenegraphs on other machines. Examples of this approach are the replication mechanisms in Repo-3D [95], and Distributed Open Inventor from Studierstube [152].

3.1.2 Dataflow Architectures

A dataflow architecture is a specialization of a distributed framework. It is not aimed at connecting arbitrary components, but instead makes it easy to combine a set of lightweight filters. The first formal definition of dataflow programming was given by Dennis [36]. A more practical definition is given by Wikipedia [182]:

> Dataflow architectures are based on the idea that changing the value of a variable should automatically force recalculation of the values of other variables. Spreadsheets are perhaps the most widespread embodiment of dataflow. For example, in a spreadsheet you can specify a cell formula which depends on other cells; then when any of those cells is updated the first cell's value is automatically recalculated. It's possible for one change to initiate a whole sequence of changes, if one cell depends on another cell which depends on yet another cell, and so on.

There are a variety of visual programming environments for dataflow programming on a single machine (e.g., AVS [3] for scientific data visualization or Shake [7] for special effects in movies). The application of dataflow programming to user interfaces in UAR is a relatively widespread technique, and several architectures are based on that principle (e.g., Unit [116], OpenTracker [141], VRIB [136] and VRPN [145]).

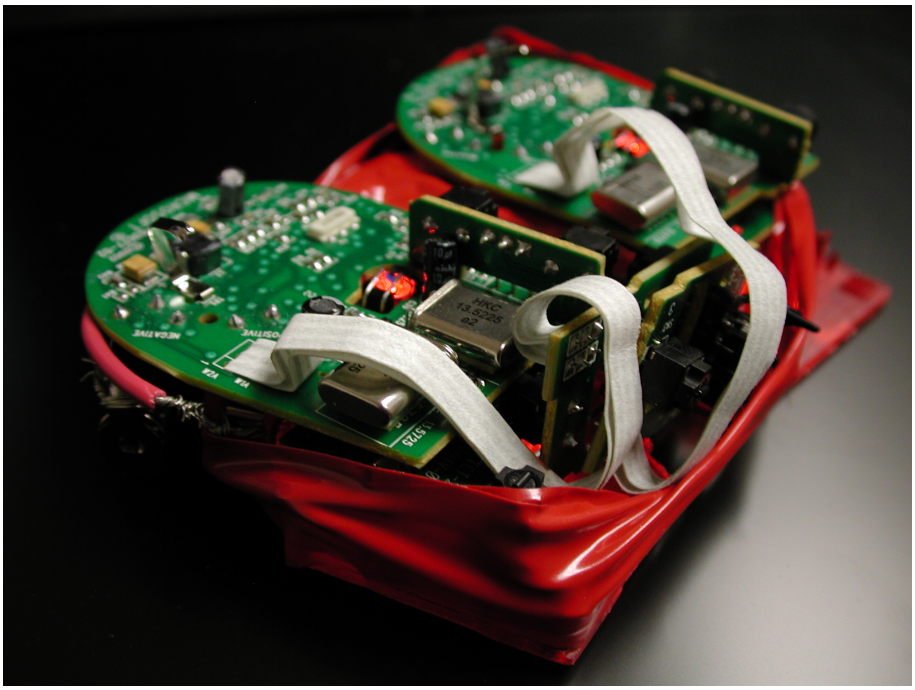
OpenTracker and VRPN use dataflow programming to integrate tracking data. They both allow the user to declare how a set of prefabricated nodes are connected to each other. For example, the outputs of one node that emits positions and one node that emits rotations can be connected to the inputs of a combination node that finally emits the complete 6DOF pose data. Other facilities in these frameworks include transformation of coordinate systems, Kalman filters and constraining of translations along fewer than 3DOF.

VRIB and Unit focus instead on the modeling of interactions. They apply the same principles as in OpenTracker and VRPN, but their data sources are not only limited to trackers, but they additionally support a wide range of input devices. An example realized within Unit is to combine two mice that are attached to each other (see Figure 3.2(a)) into a 3DOF input device (measuring translation in a plane and rotation). The corresponding dataflow network is shown in Figure 3.2(b).

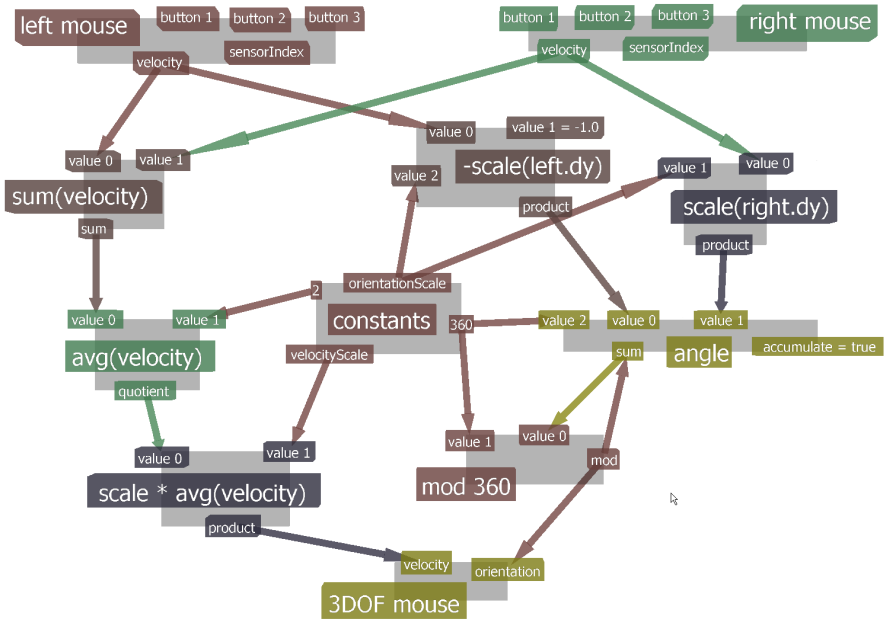
3.1.3 User Interface Management Systems

User Interface Management Systems (UIMS) were first defined in the workshop on UIMS in Seeheim (1983) and exhaustively described by Olsen [115]. A compact explanation of UIMS is given in [112] by Myers and colleagues:

> The term "user interface management system" was coined to suggest an analogy to database management systems. Database management systems implement a much higher and more usable abstraction on top of low-level concepts such as disks and files. User interface management systems were to abstract the details of input and output devices, providing standard or automatically generated implementations



(a) Two mice attached to each other to create a 3DOF input device.



(b) The corresponding dataflow network.

Figure 3.2: Dataflow programming with Unit [118].

of interfaces and generally allowing interfaces to be specified at a higher level of abstraction.

Typical high-level abstractions used for specification in UIMS are event languages and state machines. During the 1980s, many UIMS for WIMP user interfaces were developed (e.g., HyperTalk (part of Apple's HyperCard, first presented in 1987) and Sassafras [62]). However, they have become unpopular, according to Myers and colleagues [112], because the standardization of the user interface elements in the late 1980s made the need for abstractions from the input devices unnecessary. Furthermore, the slower execution speed of UIMS user interfaces compared to low-level programmed user interfaces might have been another problem.

It is my belief that UIMS are well suited for prototyping user interfaces in UAR. Since the slower execution speed is not an issue anymore with processors becoming continuously faster, the slow execution argument is probably not valid anymore. Additionally, while user interface elements have mostly been standardized for WIMP user interfaces, this is not the case for user interfaces in UAR.

Jacobs and colleagues [77] provide a recent example for a UIMS-based approach. They uses a SGML-based language for specifying user interfaces for virtual environments. The underlying components are implemented in C++. The state machines that define the behavior of user interface elements can be observed during runtime in a window.

3.1.4 Component-based Frameworks

A paradigm for software reuse is component-based software engineering [107]. A component typically is an autonomous unit within a system or subsystem. It has one or more provided and required interfaces, and its internals, other than as provided by its interfaces are hidden and inaccessible. Although it may be dependent on other elements in terms of interfaces that are required, a component is encapsulated and it can be treated as independently as possible. As a result, components and subsystems can be flexibly reused and reconnected via their interfaces. Due to these advantages of component-based software engineering over pure object-oriented software engineering, several frameworks for user interfaces in UAR are component-based.

One way to distinguish among component-based frameworks is by the granularity of their components. Most frameworks choose *services* as their components. A service is a very high-level component (e.g., Collision Detection or Renderer). Examples of this approach are Amire [41], Fluidum [42], Gaia [30] and ARVIKA [183].

Older frameworks that are graphics-driven are based on a scenegraph architecture and use *scenegraph nodes* as their components. Examples are Repo-3D [95] and Studierstube [152]. The advantage of this approach is that applications tend to be more compact, since the complete application is contained within one scenegraph.

3.1.5 Class Libraries

A class library encapsulates reusable functionality into a set of classes. For non-object oriented languages like C, the corresponding term is an API (Application Programming Interface).

A variety of class libraries and APIs exist that ease the development of user interfaces in UAR. Fundamental functionality is provided by libraries like OpenGL [8](rendering), Phidgets [125](tangible interactions), OpenCV [73](computer vision), and the Open Dynamics Engine(ODE) [160](physics simulation).

The big advantage of class libraries is that they can easily be integrated into programs or infrastructures. For example Opencroquet [159, 158, 106] uses ODE for physics simulation, while ARToolkit [67] uses OpenGL for rendering.

One of the best known libraries for UAR is probably the MR Platform [174], which covers a lot of functionality for mixed reality (tracking, rendering and calibration).

3.1.6 Scripting Languages

Scripting languages are an attempt to create languages that are extremely easy to use. Many languages for this purpose have been designed with these typical properties: they favor rapid development over efficiency of execution; they are often interpreted rather than compiled; and they are strong at communication with program components written in other languages.

Since the interaction atoms for user interfaces in UAR are unclear, a highly flexible and easy way of changing them is desirable. Scripting languages prove to be a good match for this requirement. It is not surprising that several frameworks for user interfaces in UAR employ scripting languages. Several frameworks come with their own, custom scripting language (e.g., Repo-3D [95] (COTERIE [94]) or LuaOrb [29] (Gaia [30])). Other frameworks use standard scripting languages, e.g Tcl/Tk [170] (ImageTclAR [121]), Python [134] (Studierstube [152], Panda3D [156]), Smalltalk [157] (Squeak [72]).

Python is currently one of the most popular scripting languages. It has a very modern design and a big developer community. Additionally Python bindings for a lot of class libraries exist (e.g., PyOpenGL, PyODE etc.). Class libraries can easily be wrapped by automatic wrapper generators (e.g., SWIG [167]) or using C code mixed in Python code (e.g., Pyrex [43]).

3.2 Authoring Tools

In this section we discuss several approaches to authoring user interfaces in UAR. We can divide the approaches into three different classes: First, most authoring tools are meant to be used in a conventional desktop environment (Section 3.2.1). Second, the benefits of tangible interfaces such as better collaboration and intuitive usage, have led to a variety of authoring tools based on tangible interactions (Section 3.2.2). Third, immersive authoring (Section 3.2.3) is also a relatively old concept that has been applied extensively to the authoring of virtual environments. However, in the context of user interfaces in UAR, there are only a few relatively new projects. Our focus is on tools for novices. Since we would like to support a wide range of users including non-programmers, an ideal tool would be usable without any knowledge of computer science or mathematics.

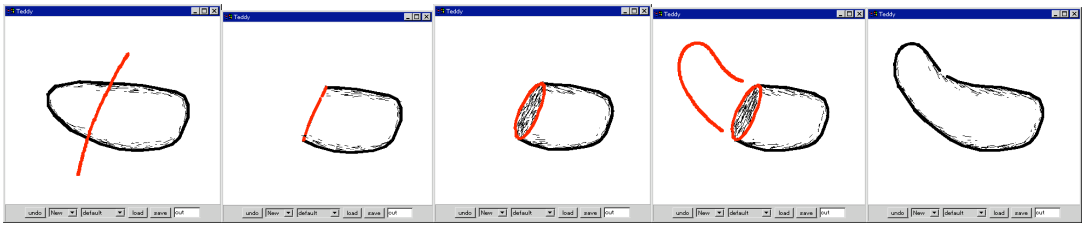


Figure 3.3: Example of culling and extrusion operations with Teddy [69].

3.2.1 Desktop Tools

Desktop environments have proven to be efficient for a wide range of tasks. So far, no dedicated desktop tool for user interfaces in UAR has been implemented, but a variety of tools for subproblems of user interfaces in UAR exists. Our examination of desktop tools starts from the most generic tools for 3D animation and 3D modeling and ends with specialized tools for augmented reality and tangible user interfaces that address several requirements for user interfaces in UAR already.

3D Modeling

Generic tasks that are necessary for all 3D user interfaces are creation and animation of 3D models. There exist several full-blown 3D environments like 3D Studio Max [40], Maya [4] and Softimage [9]. However these tools are very difficult to learn. Several projects tried to address this problem. Probably the best known examples of 3D modeling tools for novices have been developed by Igarashi (e.g., Teddy [69]). With basic operations like culling, extruding and bending (see Figure 3.3), organic life-forms can be modeled with ease.

3D Animations

For animating 3D models, there has been done an exhaustive amount of work. A very early example is Baecker's Genesys system [11], which makes it possible to sketch graphs that specify the animated behavior of 3D models. This idea has been further refined by Motion Doodles [172]. Users can directly draw the trajectory of an object to be animated into a scene. Motion Doodles also has a specialized vocabulary of strokes that specifies the motions a human-like character should perform (walking, running, tip-toeing, etc.). Earlier work by this research group [88] used freely mappable keys on the keyboard to specify the parameters of a physics simulation for knowledge-based animations.

Another more technical approach to animating 3D models is the usage of scripting languages. The same benefits as for sketching-based approaches apply, such as immediate feedback and ease of use. On the one hand these approaches provide an even greater degree of expressiveness, while they are more difficult to learn. One of the best known examples of this approach is

probably Alice [34]. In Alice, users can enter commands in a simplified version of Python. E.g. the command `bunny.move(forward)` makes a virtual bunny move forward.

Even more complex results can be achieved by environments that provide full featured scripting languages. Squeak [72], Opencroquet [159, 158, 106], Panda3D [156] are examples of this category. Panda3D is based on Python and C++. In C++ powerful components are encapsulated, while Python serves as an easy to learn scripting language to use these components. Squeak uses Smalltalk as scripting language and provides to the user both programming tools like a debugger and a class browser and easy to use graphical ways of scripting that are usable by children [179]—however only in 2D. The follow-up project Opencroquet takes these ideas to 3D. The interesting point about these projects is that runtime environment and authoring environment have been collapsed into one.

Configuring Dataflow Networks

Several tools exist for configuring dataflow networks. VR Juggler [23] and VRIB [154] provide the user with a graphical interface to specify these connections. These environments focus exclusively on tracking data. The same kind of dataflow networks for tracking data can be specified in OpenTracker [140], however not using a direct manipulation user interface, but instead by editing an XML file. For specifying dataflow networks for MIDI input devices, there exist a variety of tools (e.g., Max/MSP [35]). Its user interface is depicted in Figure 3.4(image taken from their webpage).

Workflow Description Languages

To specify high-level user interface behaviour, a variety of tools has been developed. The ARVIKA [183] project uses a graphical workflow editor that can be used to specify the way user interface elements are added, removed or changed during runtime. The workflow descriptions are based on finite state automatons. A similar approach, however without a graphical user interface for specification can be found in April [89]. A Petri net based approach for specifying the behaviour of user interface elements in virtual environments has been implemented by Jacobs and colleagues [77] (see Figure 3.5).

Tools for Augmented Reality

Several desktop tools for authoring augmented reality content have been developed. PowerSpace [58] is integrated as a plugin for Microsoft's Powerpoint. Non-programmers can create augmented reality content in Powerpoint, export an XML description of it and load it into a runtime environment and execute it. The benefit of this approach is that Powerpoint is a widely known tool, however it is unclear whether the facilities offered in Powerpoint really address the core questions of authoring augmented reality.

Timeline-based tools (see Figure 3.6) like DART [96] and Güven's and Feiner's MARS authoring tool [57] embody a more promising approach.

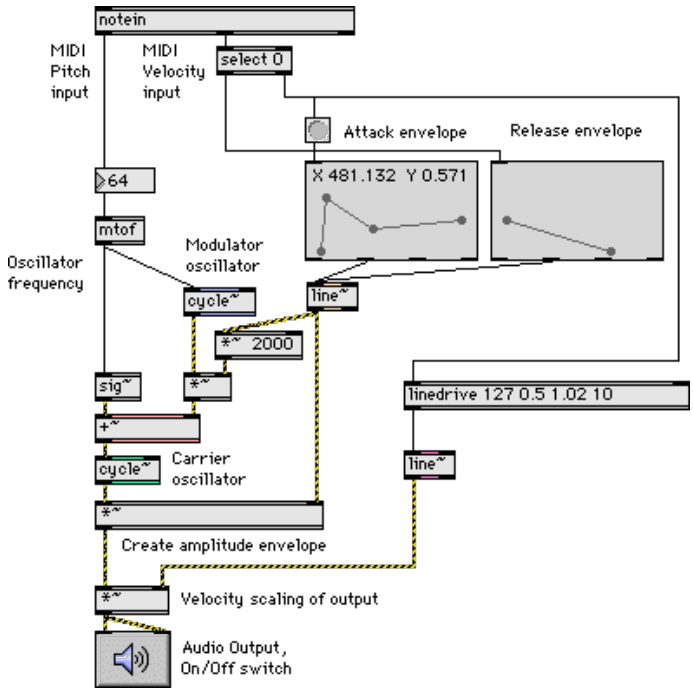


Figure 3.4: Reconfiguring dataflow networks with Max [35].

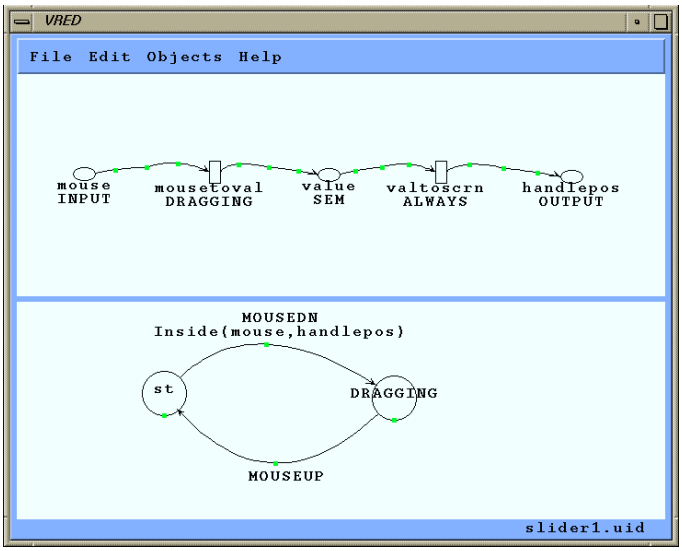
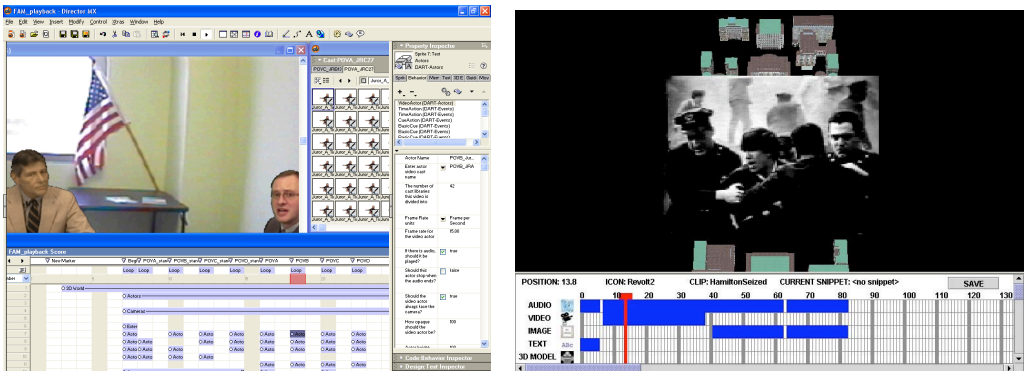


Figure 3.5: Petri nets are used by Jacobs and colleagues [77] to specify the dynamic behavior of user interface elements.



(a) DART – The Designer's Augmented Reality Toolkit [96]

(b) MARS authoring environment [57]

Figure 3.6: Timeline-based authoring tools for augmented reality.

DART has been implemented as a set of plugins for Macromedia's Director. This approach is similar to PowerSpace—extend an existing and well known tool. DART's key features are integration of trackers via VRPN [145], realistic physics through the usage of the Havok physics engine [60], augmentation of captured video and support for different reference frames for the augmented content.

The MARS authoring tool on the other hand has been implemented from scratch. It allows users to add hyperlinked content to a virtual or augmented scene. A very interesting idea in this tool is that the real world can be captured by using an omni-directional camera (see also [65]). Then when authoring in a desktop environment, the augmentations can be seen in front of the captured surround image of the real world.

Tools for Building Tangible User Interfaces

Two low level libraries help in building tangible user interfaces: Phidgets [125, 55] and i-CubeX [71]. They both come with custom hardware devices like sensors and actuators. Additionally software drivers for input handling are provided. A first step towards an IDE for building tangible user interfaces is Papier-Mâché [84] (Figure 3.7). It contains support for dealing with RFID (Radio Frequency Identification) and computer vision input. Programming development facilities include support for monitoring events and a high-level class library.

3.2.2 Tangible Authoring

Tangible user interfaces have proven to be among the most intuitive user interfaces [176]. In this section we first examine cases when tangible user interfaces were used to perform modeling tasks. Then we take a look at how tangible user interfaces can be used to build programs.

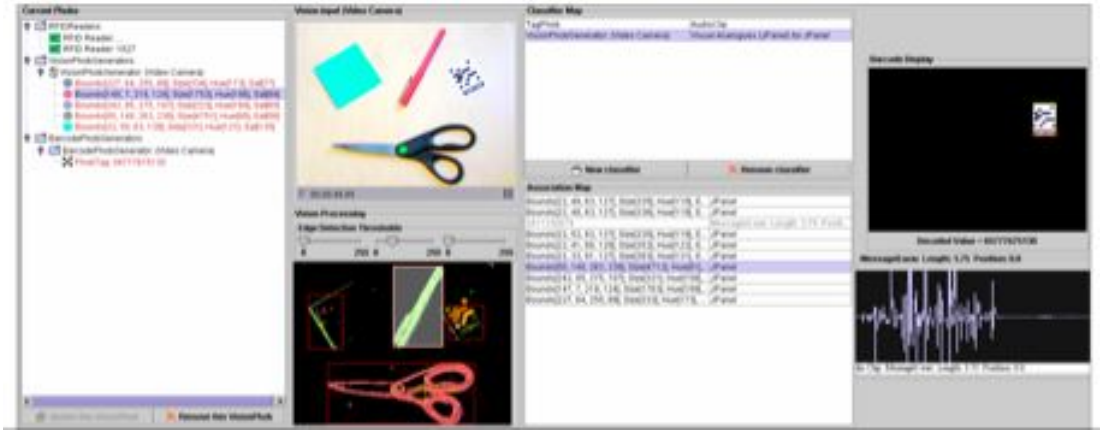
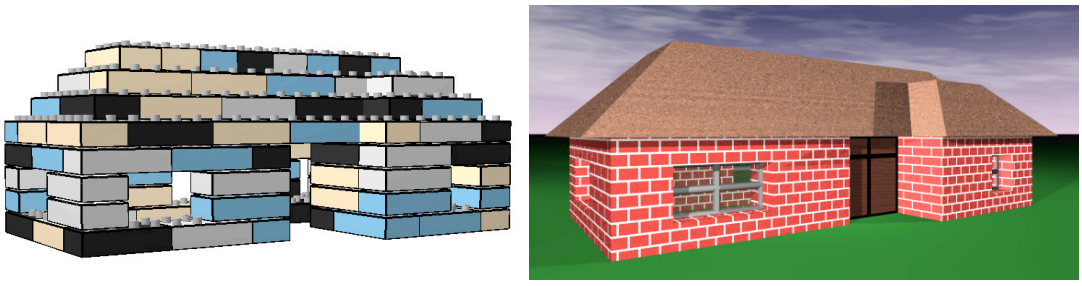


Figure 3.7: The user interface of Papier-Mâché [84].

3D Modeling

The idea of building 3D models by attaching physical building blocks to one another has been explored in [5]. Results were encouraging as quite complex models could be built (see Figure 3.8) and usage proved to be intuitve.

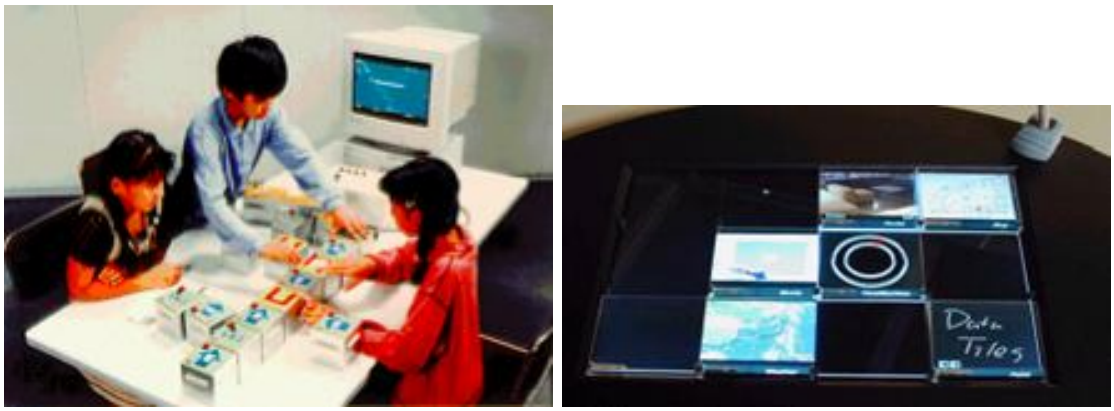


(a) Real bricks (b) Corresponding virtual model

Figure 3.8: Using tangible user interfaces to model 3D objects [5].

Programming with tangible user interfaces

An early example of the intuitive authoring capabilities of tangible user interfaces is the Algo-Block [166] system (see Figure 3.9(a)). Children could compose Logo programs out of tangible building blocks. Each block corresponds to an instruction in the program.



(a) Logo programming with a tangible user interface [166]

(b) DataTiles system [143]

Figure 3.9: Programming with tangible user interfaces.

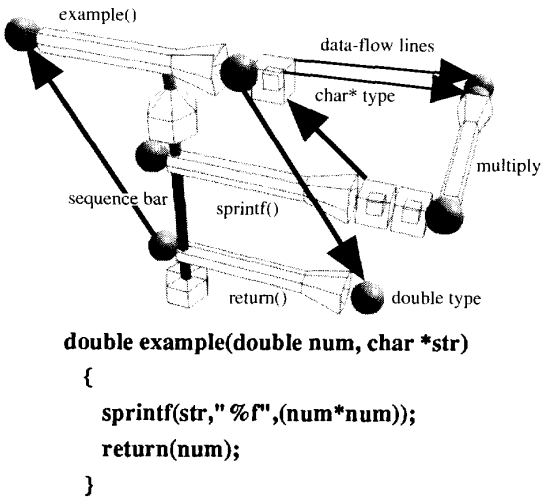
Data Tiles [143], takes a roughly similar approach. Each tangible tile (depicted in Figure 3.9(b)) has a certain function. More complex functionality can be achieved by placing the tiles adjacent to each other. For example, an image browser tile can be used to look at several images. Placing a time machine tile next to it allows the creation of simple animations by playing back the sequence of images observed previously.

3.2.3 Immersive Authoring

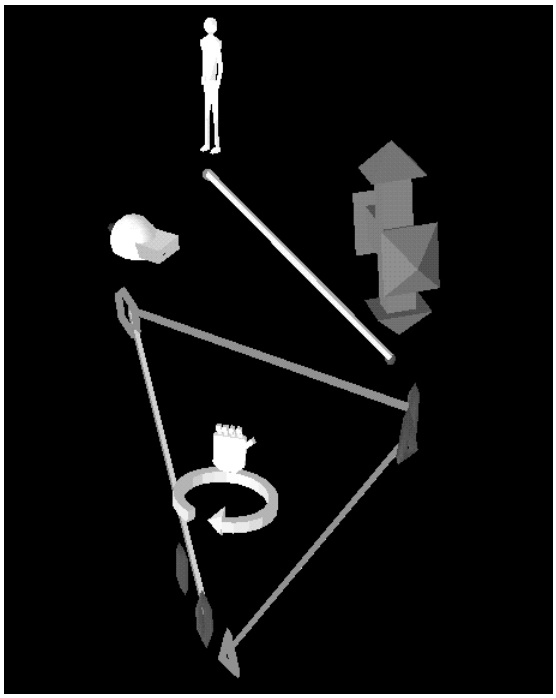
Whereas tangible user interfaces try to support interactions in an everyday setting, virtual environments strive to immerse the user in them. Similarly augmented environments strive to immerse the user in a virtual world embedded into the real world. An interesting idea regarding authoring is immersive authoring. The rationale is: if a strong infrastructure is present that can execute an immersive environment, why not use the same infrastructure for building new virtual environments? The related work presented in this section proceeds historically from virtual to augmented environments. Finally hybrid approaches that combine immersive with desktop authoring are presented.

Virtual Environments

The two earliest systems employing the idea of immersive authoring are from Steed and Slater [163], and Stiles' and Pontecorvo's Lingua Graphica [164]. They both follow the older idea to represent user interface logic in a graphical language. The user can directly manipulate the visual representation of user interface logic. These changes are propagated to the underlying user interface. Figure 3.10 shows both graphical representations. It seems that Steed's and



(a) Representation of a simple program in Lingua Graphica [164]. That representation appears to be very low-level.



(b) Representation of a dataflow network in Steed and Slater [163]. The programmer is shielded from low level-details by applying a visual language that contains higher-level elements.

Figure 3.10: Graphical representations of programs within a virtual environment.

Slater's representation is better suited, since it shields the user from low level programming oddities like type casts.

Recent examples of this approach include the PiP [90] system and 3DM [26]. Although they both are immersive approaches, they differ from Lingua Graphica and Steed's and Slater's work in one important way: they do not present to the user a dataflow network. PiP uses programming-by-example techniques to specify interaction logic, whereas 3DM has a completely different scope: it is used to model 3D objects immersively.

Augmented Environments

For augmented reality several systems exist that follow the immersive authoring approach. Probably the earliest example is the Tiles system [130]. With basic operations like clone, move, drop and delete an augmented version of a airplane cockpit can be assembled. A more complex example is the recent system of Lee and colleagues [91]. In addition to supporting the functionality of Tiles, it allows the user to build a dataflow network that is represented

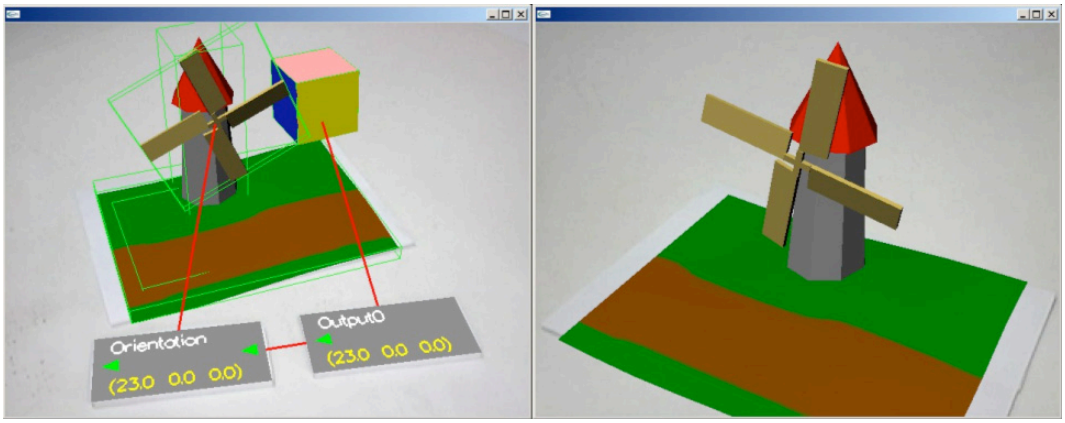


Figure 3.11: Immersive authoring of augmented reality user interfaces [91]. Dataflow networks are represented visually.

visually (see Figure 3.11). With this technique, quite complex behaviors can be built. Finally, Piekarski [126] describes a mobile augmented reality system that can be used to capture the geometries of real objects.

Hybrid Approaches

Another group of approaches combines immersive authoring with desktop authoring. Amire [41] is a component-based runtime environment for Augmented Environments with desktop configuration capabilities similar to VR Juggler [23] and VRIB [154]. All three of these systems provide authoring capabilities based on authoring by demonstration within the Augmented Environment. These capabilities were demonstrated by creating furniture assembly instructions [184]. The SAVE [66] system similarly combines authoring by a user immersed in a virtual environment with another user sitting at a workstation. Figure 3.13 depicts a scene viewed by the immersed and by the desktop user. A very interesting idea in SAVE is the *Greek God Metaphor*:

> To build a bridge between the two metaphors of interaction (immersive VR and desktop point and click) and moderate the collaboration between the two users, we have created a Greek god metaphor. The simulation side user who is immersed in the virtual environment is the hero, whereas the editor side user is the Olympian god. An avatar represents the hero in the 3D editor. All of the heros movements and actions are synchronously performed by the avatar. The god can pick up the avatar and drop the hero at any position in the scenario. The hero experiences the god in the form of a gigantic hand which moves according to the mouse pointer of the editor. Every time the god clicks, the hand reaches down to grab or touch

the object intersecting the mouse pointer. Even floor control is determined by this metaphor: The god has all the power, so his/her actions always overrule the actions of the hero.

This approach clearly goes in line with mine. The right tools should be used for the right task. Fine-grained tasks, like adjusting the position of a virtual object, should be done directly in the virtual environment – since the objects should finally be perceived in the virtual environment when the authoring is completed, it is perfectly sensible to perform the authoring task within the virtual environment. On the other hand, a desktop user interface can offer capabilities that cannot be matched by a virtual environment (e.g., the higher screen resolution of a desktop computer make it better suited for overview tasks).

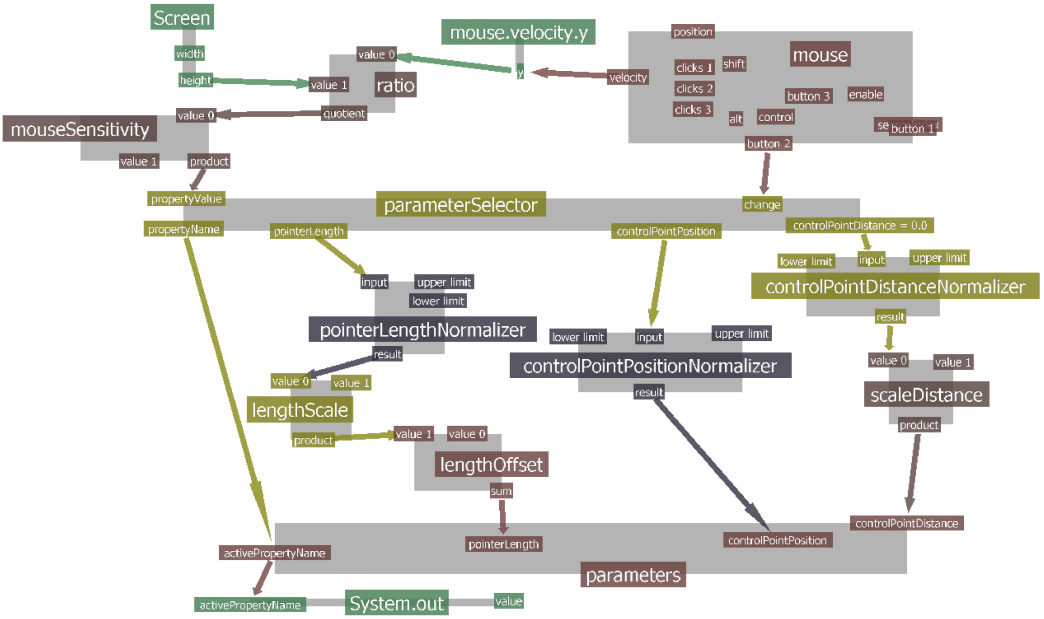


Figure 3.12: A desktop user interface to the Unit [116, 118] framework.

The Unit [116, 118] environment also provides a desktop tool for configuring dataflow networks (see Figure 3.12). The dataflow networks can be adjusted by changing values of nodes that determine the low-level behavior of interaction techniques, or by exchanging subgraphs of the network, thus switching between completely different interaction techniques. A very interesting idea in Unit is that its inherent modular architecture makes it possible to additionally use arbitrary input devices to perform these actions. In this fashion, interaction techniques can be tweaked immersively or at the desktop.

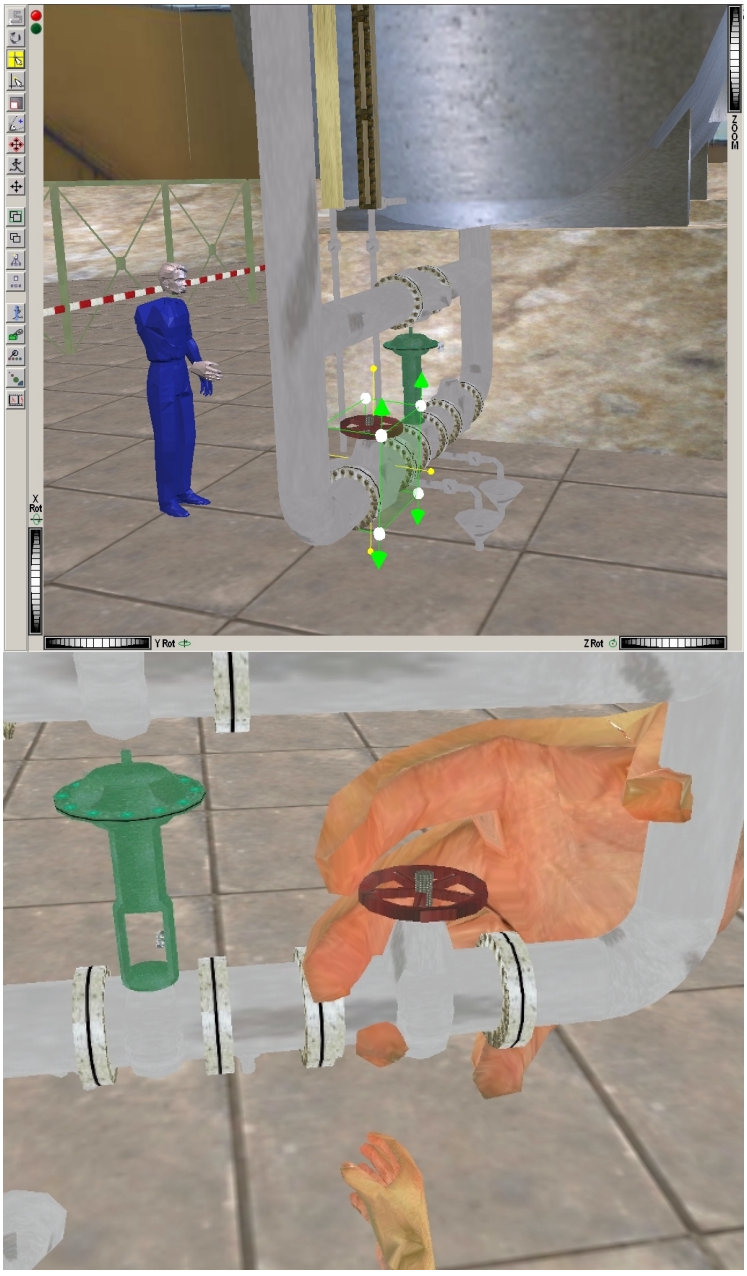


Figure 3.13: Views of the same scene by a desktop user (top) and an immersive user (bottom) in the SAVE system [66].

3.3 Reflections

This concluding section of the literature review describes how the research projects presented in this chapter relate to my research. First, I discuss the relation of the runtime environments that have been presented in Section 3.1 to my approach, which will be presented in Chapter 4. Second, I talk about the relation of the authoring tools (Section 3.2) to my authoring approach, which will be presented in Chapter 5.

Runtime Environments

My runtime environment consists of two layers: the toolkit AVANTGUARDE that I have created on top of DWARF provides a runtime environment for user interfaces in UAR. To discuss the relation of this approach to the projects presented in this chapter, we start with an outline of DWARF and AVANTGUARDE. Then we contrast the overall concept to previous work. The discussion of the overall concept is most important to highlight my contribution, since the combination of AVANTGUARDE and DWARF is what makes my approach unique.

DWARF supports sets of components that are able to connect to one another across a dynamically configurable peer-to-peer network of distributed processes. The connectivity structure is not fixed at startup time. In fact, it can be changed arbitrarily at runtime. Connections between processes are established via pre-defined interfaces, called *needs and abilities*, defining sources and sinks of information flow between components. One source can be connected to many sinks and vice versa. The DWARF middleware thus enables the development of highly dynamic, flexible system arrangements. DWARF can be classified under these technical approaches:

Component-based framework The atomic units of DWARF are components.
Distributed framework These components can be arbitrarily distributed across a set of machines.
Scripting support DWARF is based on CORBA. Thus, all languages supported by CORBA can used in DWARF. So far, we have used JAVA, Python and C++ to create components. The scripting language Python turned out to decrease development times significantly.

Since AVANTGUARDE is based on DWARF, it inherits all the characteristics of DWARF. It is a collection of components that address the specific requirements for user interfaces in UAR. The following approaches were applied when developing AVANTGUARDE:

Dataflow architecture The continuous integration subsystem of AVANTGUARDE is a dataflow architecture. It can be used to create dataflow networks for the interpretation of user input, tracking or contextual data.
User interface management system The User Interface Controller component of AVANTGUARDE is very similar to a UIMS. It can be used to specify dialog control, based on the formal model of Petri nets.

The combination of the flexible, distributed communication infrastructure DWARF with the technical approaches of AVANTGUARDE (dataflow programming and user interface management system), and the specific components of AVANTGUARDE (3D Viewer, lightweight filters and the Petri net-based User Interface Controller) makes it possible to build UAR systems, which are not easy to build with other runtime environments. For example, the SHEEP application (see section 4.3) would be hard to build with any other runtime environment that I am aware of.

Why is UAR not addressed by other runtime environments? The reason could be that the design goals of ubiquitous computing and augmented reality runtime environments are, to some extent, contradictory. For augmented reality, one of the most important goals is to have precise augmentations. This requires a good tracking infrastructure and fast rendering. On the other hand, for ubiquitous computing, the focus is more on the dynamic coupling of many devices, not on a high-end graphical presentation. To illustrate this point, we will now examine two ubiquitous computing frameworks (iRoom and MIThril) and highlight their shortcomings for building augmented reality applications. Then, we describe the problems that many augmented reality frameworks (Tinmith, ARVIKA, BAT, STUDIERSTUBE and COTERIE) have in building ubiquitous computing systems.

A well-known runtime environment for ubiquitous computing is the iRoom [82] project from Stanford University. Their work shares some common concerns. They have built a highly dynamic software infrastructure for mobile and distributed user interfaces based on tuplespaces [81]. However they do not address tangible interactions and augmented reality interfaces. Instead they focus on more conventional input like pen-based interactions, whereas the output they are mainly concerned with is wall-sized displays. As a result, iRoom does not couple real world objects to visualizations and thus does not provide a data flow framework for continuous integration. MIThril [38] is based on a similarly flexible infrastructure. Its design goals have however been totally different than the ones for an augmented reality infrastructure: cheap hardware with a small and lightweight form-factor and low-fidelity user interfaces (following the calm computing paradigm for ubiquitous environments)—MIThril even uses subliminal information display [37].

Tinmith [126] is a distributed augmented reality framework that shares data in a central repository, called Object Store. Clients access the Object Store by explicitly addressing elements, similar to accessing a file system. For two reasons, this approach is not feasible for highly dynamical UAR systems: first, a large number of clients would lead to an overload of this central repository. Second, since the elements have to be addressed with an explicit id, the ad-hoc combination of components can be a big challenge.

ARVIKA [183] and BAT [113] are both client-server architectures. This makes it impossible for clients to communicate directly with each other. In multi-user environments (typical for ubiquitous computing), this makes it very hard to implement collaborative interactions, since sharing data efficiently becomes very difficult.

STUDIERSTUBE [152] is one of the most prominent distributed augmented reality frameworks. All components for an application are stored in a scenegraph: objects to be rendered, trackers, input components and application logic. This approach is very elegant on the one hand; on the other hand, it introduces a big problem: how can dataflow networks between distributed

trackers and input devices be modeled? Since the STUDIERSTUBE scenegraph cannot be split up into smaller parts that are distributed among several machines, this is very difficult to achieve. To address this problem, the input and tracking components were removed from the scenegraph, and have been put in a separate dataflow network: OpenTracker [141]. This solves the distribution problem, since OpenTracker dataflow networks can contain networked nodes. However, the OpenTracker graph cannot be changed during runtime. This makes it impossible to implement dynamic spheres of influence with it.

Probably the best-suited augmented reality framework for implementing ubiquitous computing systems is Repo-3D [95] (built on top of the distributed objects system COTERIE [94]). Although it also uses a scenegraph-based approach, one of its design goals was to treat all data uniformly—from a distribution perspective, there is no distinction between scenegraph objects, tracking data or user inputs. This approach closely resembles the communication mechanisms in DWARF, that similarly treats all data uniformly. However, Repo-3D has one drawback: its learning curve is rather steep [95]. Additionally, it is based on Modula-3, a language that is not used widely, anymore. Compared to my approach, there are two more important differences: first, it does not have mechanisms for service discovery unlike DWARF. Second, Repo-3D is a general purpose language that makes it possible to do distributed graphics programming. However, compared to AVANTGUARDE, it does not come with explicit mechanisms for modeling dataflow networks or dialog control.

To conclude our discussion of related runtime environments, we discuss two approaches that are very similar to AVANTGUARDE: Unit [116] and the system by Jacobs and colleagues [77].

A user interface in Unit is composed of several small units that can also be networked. This makes Unit, in principle, usable for building flexible, distributed UAR systems. However Unit has three problems that make it hard to achieve this: first, Unit uses UDP for communication,which requires a sender to explicitly address the receiver by its IP address (UDP broadcasts do not have this constraint; however, network congestion will be a serious problem when taking this approach). The missing network transparency makes it impossible to build highly flexible systems (e.g., when communication partners are not known in advance). Second, Unit lacks specification possibilities for dialog control. AVANTGUARDE on the other hand has dedicated facilities (i.e., the User Interface Controller) for this. Third, there are no standard interfaces for the small units that compose a Unit system—the basic prerequisite for component-based approaches. Reuse of units between different applications is very challenging without standard interfaces.

Jacobs and colleagues' system [77] makes it possible to build dataflow networks, to specify dialog management logic in Petri nets, and to display 3D content—all of these facilities can also be found in AVANTGUARDE. However, several points are not addressed: because they focus exclusively on virtual reality, where input hardware is, to a certain degree, standardized and known in advance, dynamic device exchange is not really an issue. In contrast, we are concerned with flexible exchange of I/O devices, and distribution of software components over several hosts. Additionally, we consider output demultiplexing and control of multiple output components to be an issue.

Authoring Tools

To ease the development of UAR user interfaces, we have developed several tools (Chapter 5). All of these tools follow the development at runtime idea. We start by discussing the overall concept behind the tools, which embodies their main contribution. Then, we discuss each tool on its own.

The core idea of these tools is, to create a toolbox of lightweight, flexible tools that can be combined by the user interface author as necessary. This idea resembles the plugin mechanism that can be found in many programs (e.g., Adobe Photoshop). Although plugins can serve very different purposes, no plugins have been developed, that use different user interface paradigms to accomplish their task. The tools that I have developed use a large variety of user interface paradigms: tangible user interfaces, augmented reality and conventional WIMP user interfaces. Each of these paradigms has its own benefits and limitations. However, with the appropriate combination of tools, the maximum benefit for the user interface author can be achieved.

Almost all tools presented in this chapter strictly follow one user interface paradigm to accomplish their tasks. Exceptions are the SAVE system [66] and Unit [116]. The SAVE system allows authors to choose between a WIMP tool, or a tool in virtual reality. Since arbitrary input devices can be used in Unit to change properties of the user interface to be developed, it also makes it possible to use different user interface paradigms to address different tasks. However, my toolbox addresses a wider range of tasks than Unit or SAVE.

Additionally to the novel idea of a toolbox of cross-paradigm tools, some of these tools have contributions, when examined on their own:

Monitoring the user. We are going to present a tool using a novel augmented reality visualization [114] that makes it possible to clearly see the visual attention of a user with a combination of head-tracking and eyetracking. Probably the most similar work has been done in the SenseShapes project [117]. Olwal and colleagues have used the user's head-gaze to support more precise object selection. They also visualize that by AR overlays that show the user's field of view. Our work differs in two ways: first, we have additionally used eyetracking to get an even more precise idea about the user's visual attention. Second, our primary concern is not to select objects, but a more general one: to provide help to designers of attentive user interfaces (i.e., user interfaces that take the user's visual attention into account).

Configuring dataflow networks. In joint work with Alex Olwal, Blaine Bell, Nick Temiyabutr and Steven Feiner, we have developed an immersive visual programming environment [147, 150], which makes it possible to reconfigure dataflow networks by direct manipulation. For example, to change a source in a dataflow network, the user can touch the respective input device with a tracked wand. This extends the work presented by Lee and colleagues [91] in important ways: it can configure visualizations on multiple displays and is also able to interact with real-world objects. Similarly, the work presented in Section 3.2.3 (e.g., Steed and Slater [163], and Stiles' and Pontecorvo's Lingua Graphica [164]) is limited to virtual environments, as opposed to our work that addresses the more general idea of UAR environments.

Within this chapter, we have examined a variety of WIMP-based dataflow programming environments (see Section 3.2.1). Current usability research suggests that the more direct a manipulation is, the easier it is to use. In our immersive authoring tool, the user does not have to make a mental transformation (e.g., 'Box A on the screen refers to input device X'), but instead can directly interact with the input device X. This constitutes a more direct manipulation.

Specifying dialog control. To specify dialog control, the User Interface Controller Editor [64] is explained. It embodies a visual front-end for specifying dialog control, similar to Jacob's and colleagues' system [77]. However, since the User Interface Controller Editor is built on top of AVANTGUARDE, it can control UAR user interfaces, instead of just virtual environments.

Creating context-aware animations A set of tools to experiment with context-aware mobile augmented reality user interfaces (Section 5.5) is presented. What is especially interesting about these tools is that one WIMP tool is combined with several immersive tools, enabling a novel way of specifying context-aware animations. These tools use several concepts presented in this chapter. The WIMP tool allows to sketch animations, similar in spirit to Genesys [11] or Motion Doodles [172]. Three tangible interfaces (applying principles presented in Section 3.2.2) can be used to perform the following tasks: to simulate context, experiment with different reference frames, and set parameters for animations.

The AVANTGUARDE Toolkit

AVANTGUARDE is a software toolkit for rapid prototyping of user interfaces in UAR. It is a collection of components that are connected using the DWARF middleware. The example application SHEEP highlights the benefits of AVANTGUARDE.

The main characteristics for UAR user interfaces were presented in Chapter 2. Section 3.1 mapped these characteristics to technical approaches that are used to address these characteristics. This chapter describes details of the AVANTGUARDE toolkit.

First, the fundamental infrastructure DWARF is described. DWARF [13, 14] (Section 4.1) is the underlying infrastructure for AVANTGUARDE. It connects a set of components that can be arbitrarily distributed across a set of machines. Thus, it can be used to implement Spheres of Influence.

Second, the AVANTGUARDE toolkit itself is explained. I have developed the software toolkit AVANTGUARDE [148] (Section 4.2) that is composed out of DWARF components. Its architecture and components address the specific requirements for user interfaces in UAR.

Third, I present the example application SHEEP [103, 149] (Section 4.3) to illustrate the usage of the toolkit.

4.1 DWARF

The AVANTGUARDE toolkit builds upon the ad-hoc connectivity and the peer-to-peer-based architecture of the Distributed Wearable Augmented Reality Framework [13, 14] (DWARF) that has been developed in our group. This section gives an overview of the capabilities of DWARF.

Some content of this section has been jointly developed and written with Thomas Reicher, Martin Bauer, Martin Wagner and AsaMacWilliams. More details about the capabilities of DWARF can be found in the theses of my colleagues:

Architecture and middleware Thomas Reicher has developed most of the architectural concepts for DWARF [137]. Asa MacWilliams has implemented and refined these concepts. As a result, he has produced the middleware [101] for DWARF.

Tracking Martin Wagner has applied DWARF's capabilities to the problem domain of tracking in UAR [180]. Martin Bauer is also working in the area of tracking for UAR. However

compared to Martin Wagner, he is more concerned with the algorithms involved, than with the overall concept for tracking in UAR. His thesis is supposed to be finished 2006.

I was the only team member addressing user interfaces. Thus, the main result of my work in the context of DWARF is AVANTGUARDE, described in Section 4.2. However, I have contributed to the work presented in this section in three ways: first, I have been involved in the overall conception of DWARF. Second, I have continuously articulated user interface requirements that have led to adjustments of the DWARF middleware. For example, the extension of DWARF to do wild-card matching of attributes (described in [101]) was needed to allow the dynamic reconfiguration of the TouchGlove input device (see Figure 4.5). Third, together with Asa MacWilliams, we have investigated the development at runtime process.

DWARF supports sets of components that are able to connect to one another across a dynamically configurable peer-to-peer network of distributed processes. The connectivity structure is not fixed at startup time. In fact, it can be changed arbitrarily at runtime. Connections between processes are established via pre-defined interfaces called *needs and abilities* defining sources and sinks of information flow between components. One source can be connected to many sinks and vice versa. The DWARF middleware thus enables the development of highly dynamic, flexible system arrangements. More specifically, the following features of DWARF support a developer:

Flexible architecture Because of the fine granularity of components and the loose coupling between them, DWARF systems are highly flexible.

Fast Several communication protocols are implemented for the communication between components. Some of them are especially well suited for real-time applications (e.g., shared memory and CORBA (Common Object Request Broker Architecture) events).

Distributed The components that form a DWARF system can be a combination of local and remote devices. Distribution is completely transparent to the components.

Adaptivity With ad-hoc connectivity and reconfigurability of components, DWARF systems are also inherently adaptive.

Operating System independent To allow deployment among a variety of devices, DWARF has been designed to be independent of a specific operating system. We have successfully implemented DWARF systems on BSD, Linux, Windows and Mac OS X platforms.

Programming language independent Similarly, DWARF supports three programming languages so far: JAVA, C++ and Python[1].

In this section we present the rationale and details of DWARF's design. We start by presenting the design goals that we had in mind when designing DWARF (Section 4.1.1). Then we present the key concepts in DWARF that were facilitated to achieve these goals (Section 4.1.2). Finally, we explain, how Spheres of Influence (Section 2.1.1) can be implemented with DWARF (Section 4.1.3).

[1] Many thanks to Joseph Newman for his work on DWARF, which resulted in the addition of Python to the list of supported languages!

4.1.1 Design Goals

The major design goals of DWARF were a high degree of reusability on various levels and support for both ubiquitous computing environments and rapid prototyping of augmented reality user interfaces.

Reusability

A study on software architectures for augmented reality systems in 2002 [138] revealed that most existing augmented reality systems were developed to test and demonstrate specific techniques. Only very few systems employed a reusable framework that allowed the development of several augmented reality applications on the same base.

A paradigm for software reuse is component-based software engineering [107]. A component can always be considered an autonomous unit within a system or subsystem. It has one or more provided and required interfaces, and its internals are hidden and inaccessible other than as provided by its interfaces. Although it may be dependent on other elements in terms of interfaces that are required, a component is encapsulated and it can be treated as independently as possible. As a result, components and subsystems can be flexibly reused and reconnected via their interfaces. Due to these advantages of component-based software engineering over pure object-oriented software engineering, we chose to use component technology as the basis for the development of the DWARF framework.

Ubiquitous Environment Support

In ubiquitous computing environments, a large number of highly distributed computing devices should interact. The capabilities, range of operation and availability of these devices is not foreseeable at development time. Thus, DWARF has to cope with a dynamically changing infrastructure. Failure of parts of the overall system and reduced or unavailable networking capabilities have to be tolerated. In addition, it should be possible to reconfigure parts of the system at runtime, to enable mobile devices to interact with previously unknown stationary environments.

Rapid Prototyping Support

A major challenge for user interface research in augmented reality and ubiquitous computing is that interaction idioms and metaphors are not established yet or even not known—in contrast to classical WIMP user interfaces. To establish the WIMP paradigm with its idioms like *Drag and Drop* or *Point and Click*, a lot of research and usability evaluations have been done. It is clear that for augmented reality, such experiments and evaluations still have to be carried out.

For experimenting with new concepts it is necessary to develop prototypes quickly. In traditional WIMP usability engineering, it is common practice to create mockups, sometimes just a sketch on a piece of paper, as a basis for discussion. However, in our domain this would not be applicable, as 3D sketches illustrating dynamic aspects are difficult to draw and

understand. To shorten the development time for illustrative prototypes, a framework for augmented reality user interfaces that supports rapid prototyping is necessary.

4.1.2 Key Concepts

The design goals of DWARF necessitate an architecture that is both flexible in order to support ubiquitous computing environments and well structured for a clear separation of tasks in rapid prototyping applications.

Component Paradigm

A DWARF system consists of a set of loosely coupled distributed components, called DWARF *services*. A service is composed of typed *needs* and *abilities*, i.e. data it needs as input and data it is able to offer. *Connectors* specify how data is exchanged; to date, we have implemented event-based, RPC-based and shared memory communication facilities. Distributed *service managers* maintain descriptions of available services and provide connectors matching the need and ability descriptions of services. The distributed service managers communicate following a peer-to-peer paradigm, thus eliminating a single point of failure and allowing for the creation of ad-hoc networks.

As an example, consider an application that renders some virtual object in a head-mounted display. To spatially align this object with the real world, it needs its user's position and orientation delivered at high update rates by a tracking device. In DWARF, both the rendering application and the tracker are modeled as services. The tracker has the ability to deliver *PoseData* of the user, and the renderer has the need for exactly the same *PoseData*. Therefore, the service managers can instantiate an event channel and give references to this channel to both the tracker and the renderer. Subsequently, the tracker sends updates of the user's position via this channel.

To specify needs and abilities more precisely, DWARF allows to attribute abilities with name/value pairs. Boolean predicates can be imposed upon needs, thus limiting the choice of matching services. This mechanism can be used to create chains of services in a decentralized fashion [102].

Figure 4.1 shows a UML diagram of a distributed simple augmented reality system built with DWARF.

Reconfiguration at Runtime

In dynamic ubiquitous computing environments, it is crucial to allow parts of the system to adopt to changing environmental conditions. Thus, DWARF allows reconfiguration of its components at run time. By constantly monitoring the availability and descriptions of services, the service managers notify running services of changes in the overall setup. For example, once a user with a mobile setup leaves the tracking range of a given tracker, the tracking data of this device becomes unavailable. However, another device might now deliver matching data.

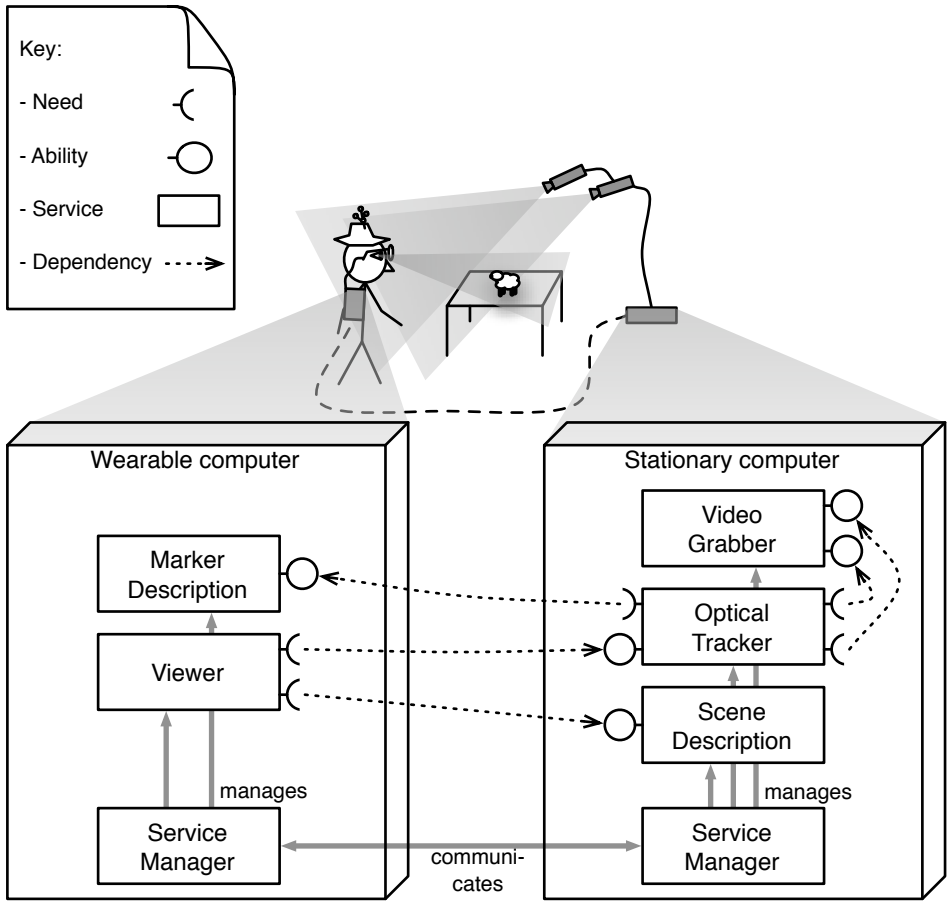


Figure 4.1: Services with needs and abilities, managed by service managers, on distributed hardware of a simple augmented reality system. The user carries an optical marker on his hat, on described in the *Marker Description* service. This is used by the room's optical tracking system (the *Video Grabber* and *Optical Tracker* services) to determine the position of the user's head. The user's *Viewer* service uses this position to display the scene from the *Scene Description* service. The services are connected and coordinated by distributed *service managers*.

Thus, the service managers reconfigure the mobile setup by disconnecting the old tracker and connecting the new one.

In consequence, the loose coupling of DWARF services enables the system to react to dynamic changes within the environment in a highly flexible way. This facilitates using DWARF-based augmented reality systems in dynamic ubiquitous computing environments. It also enables rapid prototyping of real-world augmented reality applications, as parts of the system can be exchanged at run time, thus enabling short modify-recompile-test cycles.

4.1.3 Implementing Spheres of Influence with DWARF

This section concludes our discussion of DWARF by explaining how Spheres of Influence (Section 2.1.1) can be implemented in DWARF. We explain this mapping in three steps. First, we give a real world example that we would like to implement. Second, we explain on a higher level, how the implementation with DWARF would look. Third, we examine at the lowest level of DWARF, how it is implemented through XML service descriptions.

Suppose we would like to model an interaction that requires two input devices and one or two output devices. For example, we would like to do an interaction that requires the user to give speech input and at the same time, the user should use a tracked pointing device to indicate a position in 6DOF. As a result, a new virtual object should be created and displayed on one or two 3D Viewers.

The required components would be: Speech Recognition, Tracker, Viewer and a User Interface Controller. The User Interface Controller will be explained in detail in Section 4.2.2. For now, all we need to know is that a User Interface Controller is a central coordination instance. It consumes tracking and input data and can change views accordingly. Thus, we would like to set up a dataflow network that connects the User Interface Controller to the other involved components. On a higher level, this models a Sphere of Influence required for the interaction.

The XML files for the User Interface Controller component would look like this (unnecessary details are omitted; a complete description of the syntax for XML service descriptions can be found in [101]):

```
<service name="UserInterfaceController">
  <need type="InputDataString" minInstances="1" maxInstances="1">
    <connector protocol="PushConsumer"/>
  </need>
  <need type="PoseData" minInstances="1" maxInstances="1">
    <connector protocol="PushConsumer"/>
  </need>
  <ability type="SceneGraphManipulator" minInstances="1" maxInstances="2">
    <connector protocol="PushSupplier"/>
  </need>
</service>
```

Figure 4.2: XML description for the User Interface Controller.

This User Interface Controller instance will be automatically connected to one input device that delivers Strings. This is achieved by the first need of type `InputDataString`. Similarly, it will be connected to exactly one input device that delivers 6DOF input (need with type `PoseData`). The ability at the end of the service description (type `SceneGraphManipulator` will connect the User Interface Controller to one or two views. All communication will run over via CORBA events. This is achieved by the specification of the `connector`. A `PushSupplier` can send CORBA events, whereas a `PushConsumer` can receive CORBA events.

As a result, we have modeled our Sphere of Influence—the logic inside the User Interface Controller component is ready to run. It only needs to get in reach of the required input and output components. In this example, we have also demonstrated that the cardinality of connections can be specified in ranges. This whole interaction could be run with one or two views.

4.2 The AVANTGUARDE Toolkit

I have created the component-based toolkit AVANTGUARDE [148] that enables the execution of user interfaces in UAR. AVANTGUARDE supports the rapid modifiability of user interface elements. The core components of AVANTGUARDE include a Petri-net-based dialog control management system, a viewer for augmented reality scenes and a set of filters to set up dataflow networks for tangible interactions, context-awareness and advanced visualizations.

Within this section, we first present the architectural principles around which AVANTGUARDE has been built (Section 4.2.1). Then, an overview of its components is given and the core components are described in more detail (Section 4.2.2). The next Section (4.3) presents an example application that illustrates how AVANTGUARDE can be used.

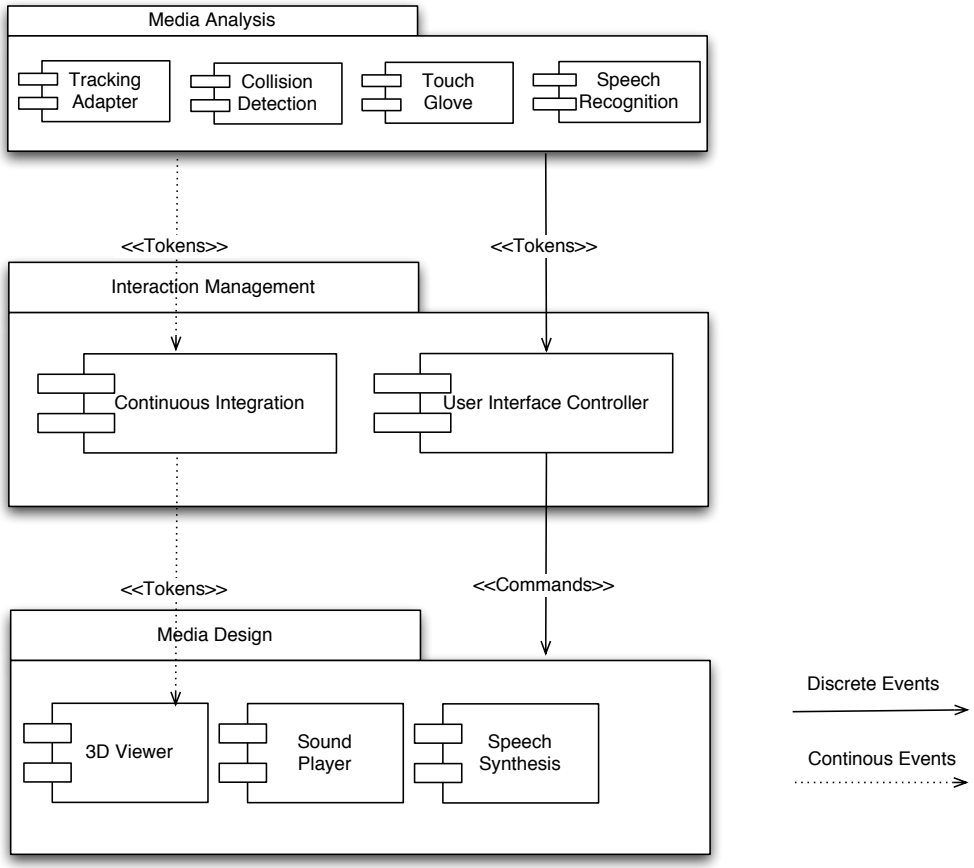


Figure 4.3: Functional decomposition of AVANTGUARDE's components.

4.2.1 Architecture

The architecture of AVANTGUARDE (see Figure 4.3) is centered around six fundamental principles that make it a viable solution for developing user interfaces for UAR. In this section we present these six principles and discuss their implications.

Layering and Device Abstraction

We arrange the UI components in three layers. Data flows from the *Media Analysis* layer, which contains input components, to the *Interaction Management* layer where the interpretation of user input takes place. From there the dataflow continues to the *Media Design* layer where the output components reside.

We have developed a standardized format for tokens that are sent from the input components to the Interaction Management layer. This enables the system to address logical input

devices [49]. Input tokens can be decomposed into four different types: analog values that can be either within a limited range (e.g., orientations) or an unlimited range (e.g., translations) and discrete values that can be either booleans (e.g., pressing a button) or text strings (e.g., the output of a speech recognition process). Due to this standardized format, we can exchange one input device for another – as long as they emit the same type of tokens. For example, a speech recognition component listening for a set of words could be interchanged transparently with tangible buttons with the same set of labels. This input taxonomy is similar in spirit to the one introduced by Mackinlay and colleagues [99].

Similarly, the Interaction Management layer sends commands to the Media Design layer. This setup corresponds to the *command pattern* described by Gamma et al. [52]. The commands consist of actions that have to be executed by the output components (e.g., by presenting the question "yes or no ?" to the user). One can interchange output components in the same way as with input components. Due to this flexible, DWARF-based component model, the exchange of I/O components works even at system runtime.

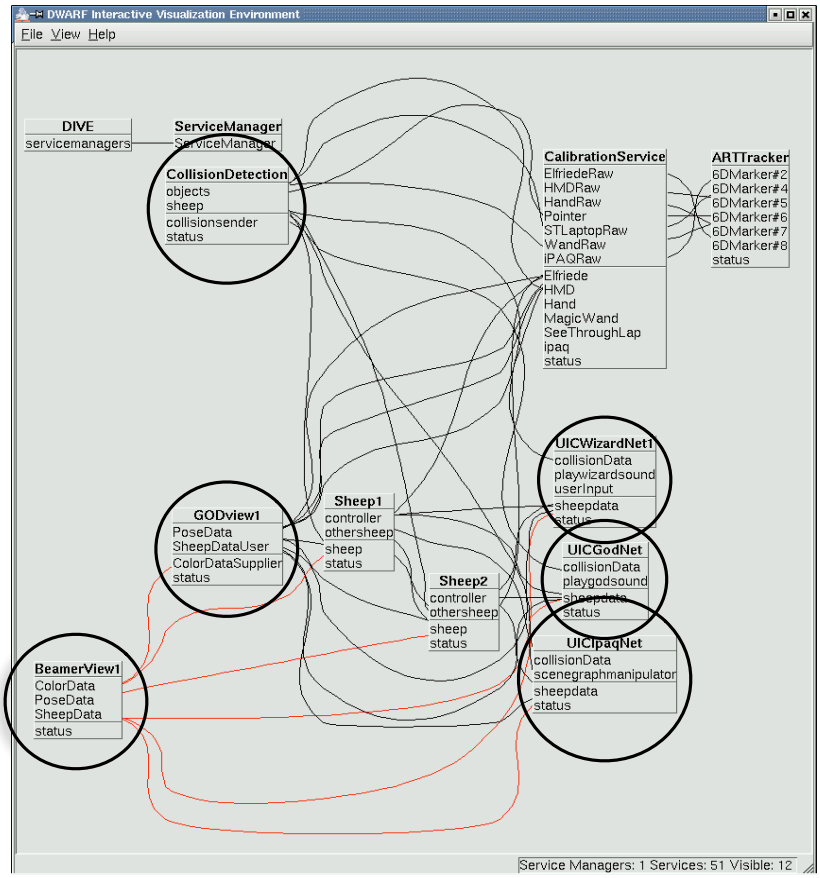


Figure 4.4: Example of a dataflow network between AVANTGUARDE components (highlighted with circles) and other DWARF components displayed during system runtime in the DIVE tool (see Section 5.3.1).

Efficient Coupling of Real and Virtual Worlds

One of the recurring input modalities used by all tangible objects is their location within the scene. For example, mobile displays present scene views depending on their current position. To ensure the most efficient transfer of pose data from trackers to 3D viewers, our system makes it possible to establish direct connections between tracking and viewing components whenever no filter has to be involved. Figure 4.4 shows an example of a dataflow network between AVANTGUARDE and other DWARF components.

For more complex visualizations (e.g., non-linear input processing [129, 161]), a pipe-filter component tree preprocesses the Pose Data emitted by the trackers before feeding it to the

viewer. This approach has already been implemented by several other frameworks. We add to this approach the concept that the filter components can be arbitrarily distributed and interchanged at runtime, once again using the flexible DWARF component model, allowing us to quickly experiment with different arrangements during system runtime. This feature turns out to be very beneficial to user interface development whenever we need to dynamically experiment with different options to interpret the tracking data [103].

In contrast, if the setup is known in advance, quite a few systems exist that are able to process and forward multi-sensory input data to a display system. An example is the OpenTracker system of the Technical University of Vienna [141]. In joint work [15], we have shown that the Open Tracker system can easily be integrated into DWARF by plugging the associated transformation hierarchy of the Open Tracker framework into the DWARF system. Similar approaches can be pursued for other systems.

Central Coordination with User Interface Controllers

Inside the Interaction Management layer, we decided to move all functionality into the DWARF User Interface Controller component (UIC), thereby combining the functionalities of Dialog Control and Discrete Integration. Our implementation of the UIC will be described in more detail in Section 4.2.2.

The UIC combines input tokens sent by the Media Analysis components and then triggers actions that are dispatched to components in the Media Design package. Note that the UIC must have an internal model of the state of the user interface. Otherwise, context-sensitive commands could not be interpreted correctly. Since several input components and several output components can be connected to the UIC, it enables the creation of multimodal and multimedia user interfaces.

Interactions and Controllers: One-to-One Mapping

To conform to our lightweight and distributed approach, we model each interaction in its own UIC. For example, all interactions that occurred in the SHEEP game (see Section 4.3) are each realized with individual UICs. When thinking about tangible interactions, this couples the functionality of each tangible object to a specific instance of a UIC. As a result, the visualization that the UIC provides at runtime tells us the state of the coupled tangible interaction. Additionally this approach fits very well with the tool metaphor for interactions [14].

Lightweight and Stateless I/O Components

To address the flexibility requirement of user interfaces, we chose to keep as much state information as possible in the Interaction Management layer (see Section 4.2.2). As a consequence, the I/O components were designed to keep as little state as possible. This allows us to add and remove I/O components conveniently at system runtime. However, this is not possible for some components. Currently, we are working on a persistence layer to be able to pass states between components.

Equal Treatment for Context and User Input

To make AVANTGUARDE context-aware, we have chosen a very simple approach. The standardized tokens that describe input by the user, can also be used to describe changes in the user's context. From a conceptual point of view, it does not matter whether the user has explicitly triggered an action or whether this action is triggered through a change of context. For example, it does not really matter to AVANTGUARDE whether a map will be displayed, because the user has clicked on a button 'Show map', or whether a context component has determined that the user needs to see a map now.

4.2.2 Components

It is important to notice that reusing these components includes no programming of code, because they are generic and meant to be reused among different applications. To tailor components to a specific application, the components are configured via an XML file.

Before we give details about the two core components of AVANTGUARDE (User Interface Controller and Viewer), we first present an overview of frequently used components. A full list of AVANTGUARDE's components can be found on the thesis' accompanying webpage [146].

Speech Recognition For commands that address the control of the system, usability studies have shown that a command language is preferable to tactile input [120]. Therefore, we provide a Speech Recognition component. Especially in applications that require hands-free working (e.g., maintenance), this type of interaction gains importance. Internally, the Speech Recognition component is a word spotter. It is configured via a context free grammar.

TouchGlove The TouchGlove is a special-purpose input device [24] developed at Columbia University (see Figure 4.5). Input tokens that are emited by this device can be fed into AVANTGUARDE. Interestingly, this device can emit both continuous and discrete input data.

Collision Detection The Pose Data emitted by the Tracking components is used by this component to detect collisions between objects. This includes collisions of real objects with virtual objects, real with real objects and virtual with virtual objects. The first two types of collisions must be considered to capture user input that is provided via movement of tangible objects. A common practice in TUIs is to couple real world objects with virtual objects. This explains the need for the last type of collisions that is virtual with virtual objects. Consider a virtual slider that moves according to the tracked hand of the user. Whenever the slider is at its maximum value, a collision between the slider and the frame, in which it is displayed, is triggered.

PowerMates Linux-based driver for the Griffin Technology PowerMate [56] rotary sensors.

MIDI Input Devices A bridge to the the Unit [118, 116] framework's driver for MIDI devices. Infusion System's iCubeX [71] converts analog sensor data to 7-bit MIDI data, making it easy to support a wide range of MIDI sensors.

Figure 4.5: The TouchGlove input device.

Game Controllers To accomodate game controllers, we connect again to the Unit framework. By using two types of USB adapters that support up to four Sony PlayStation or Sony PlayStation 2 controllers, we exploit the wide range of controllers available for the PlayStation and PlayStation 2 platform, including a generic 1m×1m PlayStation dance pad with 14 buttons and an analog PlayStation controller with buttons and two analog joysticks.

Janus This component is a wrapper for a custom-built eye tracking device [171].

Sound Player The Sound Player component is a simple JAVA application configured via an XML file. It contains the mapping between incoming commands and sound files that are to be played.

The User Interface Controller

The core component of the *Interaction Management* layer is the UIC. It combines the functionalities of *Dialog Control* and *Discrete Integration*. It interprets input tokens sent by the

Media Analysis components and then triggers actions that are dispatched to components in the *Media Design* layer.

This section first explains why we are using Petri nets as underlying data structure for the UIC. Then we give details on how we have implemented the UIC.

Petri Nets for Interaction Management In our approach we use Petri nets to model interactions, as is common practice in the field of workflow systems [2]. In this section, we give a short introduction to Petri nets and how we utilize them in our framework.

A Petri net consists of places, tokens, arcs and transitions. A Petri net is represented by a four tuple $\{P, T, IN, OUT\}$, where $P = \{p1, p2, ..., pn\}$ is the set of all places, $T = \{t1, t2, ..., tn\}$ is the set of all transitions, $P \cup T \neq \emptyset, P \cap T = \emptyset$, $IN \subseteq (P \times T)$ is an input function that defines directed arcs from places to transitions, and $OUT \subseteq (T \times P)$ is an output function that defines directed arcs from transitions to places. Places of Petri nets usually represent states or resources in the system, while transitions model the activities of the system. The arcs connect places and transitions. A transition fires when all places at the end of incoming arcs contain tokens. Transitions execute actions when fired. Optionally, all arcs can have guards on both ends. Guards can define constraints on the type and number of the tokens as well as on the value of the tokens. Transitions only fire when all guards evaluate to true, meaning that all constraints are fulfilled.

In our approach, transitions are used to encapsulate atomic interactions. More complex interactions can be modeled by combining several transitions. The characteristics exhibited by the activities in a multi-modal user interface, such as concurrency, decision making and synchronization, are modeled very effectively with Petri nets. In Figure 4.6, some of these characteristics are represented using a set of simple constructs:

Sequential Execution In Figure 4.6(a), transition $t2$ can fire only after the firing of $t1$. This impose the precedence of constraints $t2$ after $t1$. This construct models the causal relationship among activities.

Concurrency In Figure 4.6(b), the transition $t2$, $t3$, and $t4$ are modeled to be concurrent. A necessary condition for transitions to be concurrent is the existence of a forking transition that deposits a token in two or more output places. Such a construct could be used to manipulate three different output devices at once.

Synchronization In Figure 4.6(c), $t1$ will only be active if all places contain tokens. Synchronized transitions are typically used to combine different modalities to perform a single task, such as speech, gaze or a button.

We utilize Petri nets and the above described constructs among others to model interactions. Therefore, we map input events to tokens that are put into incoming places. Code encapsulated in transitions extracts the content from user input tokens and interprets it. Finally, new tokens containing commands for the *Media Design* components are composed. Tokens generated by

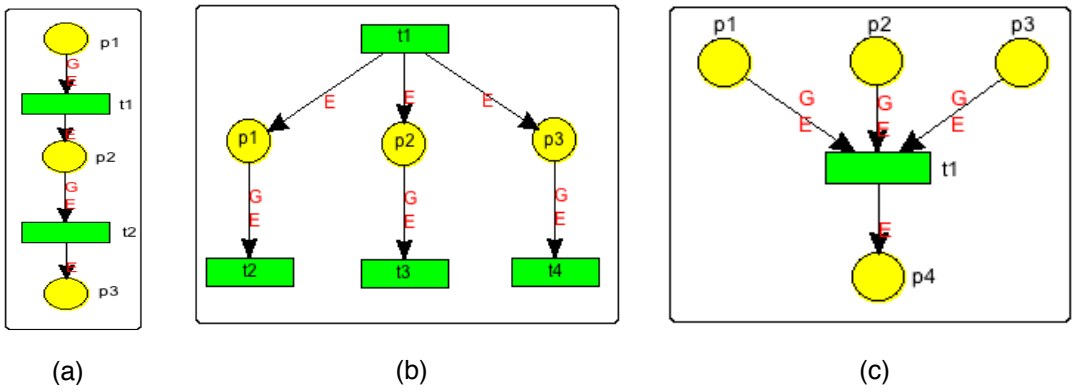


Figure 4.6: From left to right: Petri nets modeling Sequential Execution, Concurrency and Synchronization

transitions are sent to output components. This enables us to model interactions without depending on the input or output devices attached to the control component and possibly even without knowing what devices are or will be used.

Transitions link inputs to a semantic entity. Transitions can be seen as *predicates* over input *attributes*. A transition encapsulates actions that are fired when the predicate evaluates to true, which causes a change in the user interface's state. Examples of these actions could be the addition or removal of an item in a graphical view, or a change of properties of some item in an output component.

A set of places, arcs and transitions, forms an expression or, in terms of user interaction, a declaration of intent. Such constructs (see Figure 4.6) can be used as patterns [52]—reusable entities to model common problems (e.g., insert an element into a view, selection/de-selection).

Implementation The described mechanisms have been implemented on top of the Java-based JFern [80] project. JFern is an object-oriented Petri net simulation framework. JFern's Petri net model is based on the model of hierarchical Petri nets with time [139] with the additional concept of *object-based* tokens—places can contain arbitrary Java objects as tokens. It consists of a lightweight Petri net kernel, providing methods to store and execute Petri nets in realtime, and a simulator including a simple GUI for runtime visualization. JFern also supports XML-based persistent storage of Petri nets and their markings.

We take advantage of two main features, the ability to use arbitrary objects, including DWARF structured events, as tokens and the possibility to describe transitions guards and actions in native Java code. That implies that very little learning is required of programmers familiar with the Java language. The following example code shows a guard on an input arc that checks if there is a single (one and only one) token in the input place, and checks if the token has an appropriate value:

```
public boolean guard() {
  //check the arity of the multiset
  if(getMultiset().size() != 1) return false;
  //check the condition
  Number t = (Number) getMultiset().getAny();
  if(t.intValue() == 10)
    return true;
  else
  return false;
}
```

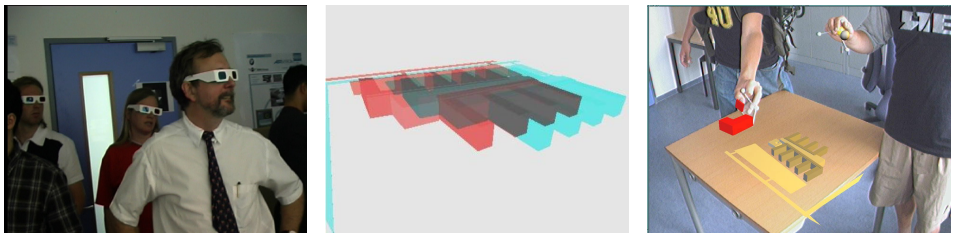
We have built a communication layer around the JFern kernel based on DWARF, connecting input places with components of the *Media Analysis* layer and connecting output places with components of the *Media Design* layer. Furthermore, different UICs, possibly running on different machines, can exchange tokens via DWARF and thus fuse their Petri nets, which follows the toolbox metaphor. That lets users roam freely in a building not only carrying a special input device with them (e.g., a PDA) but also the control component for that device which can connect dynamically to different running applications and thus enable the user to participate and benefit from those emerging applications.

In the first implementation stage, we just combined the abilities of DWARF and JFern (communication and Petri net execution) to model simple interactions and control input and output components. That approach showed that it is powerful enough to control the user interface of mixed reality applications such as SHEEP [103]. It also gives remarkable easy insight at runtime for developers and technical interested users into the processes going on by visualizing the Petri net execution.

The Viewer

The 3D Viewer component displays augmented reality scenes. An important design goal is the ability to update the virtual parts of a 3D scene in real-time. This component turned out to be quite difficult to design and implement. Our current version accepts all important commands that are necessary for the display of dynamic 3D scenes in realtime. Additionally, several viewing modes are supported: videobackground for video see-through displays and visualization of AR scenes, or support for a variety of stereo modes for different stereoscopic displays (see Figure 4.7). Furthermore, the display of content in different reference frames is possible, because the Viewer is based on a scenegraph.

The Viewer's implementation was significantly changed at one point. Our first implementation of the Viewer was based on VRML browsers. Scene descriptions were provided in VRML. A small adapter service written in JAVA communicated via the External Authoring Interface (EAI) with a VRML browser, such as Cortona from Parallelgraphics [123] on Windows platforms, and FreeWRL [51] on palmtop platforms. We encountered various problems in implementing this architecture of the Viewer, such as system incompatibilities and performance



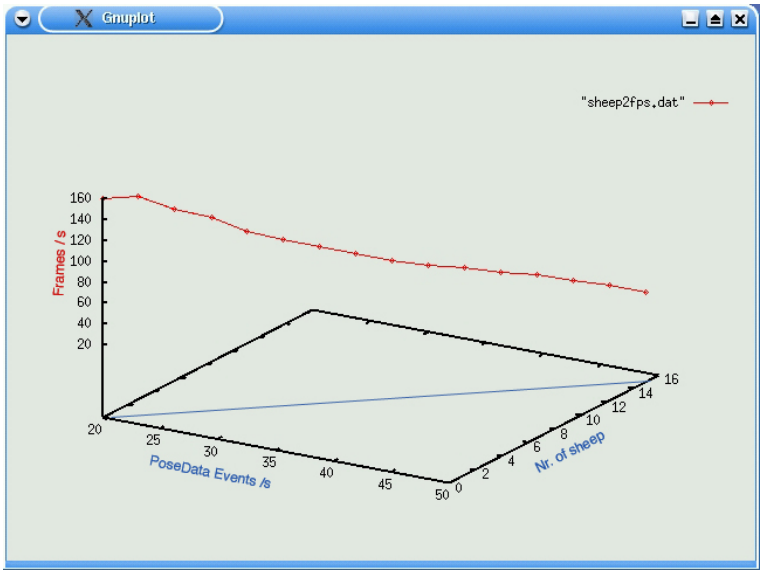
(a) Audience wearing red-cyan glasses (b) View rendered in red-cyan (c) Video overlay (example from the ARCHIE system).

Figure 4.7: Various modes supported by AVANTGUARDE's 3D Viewer.

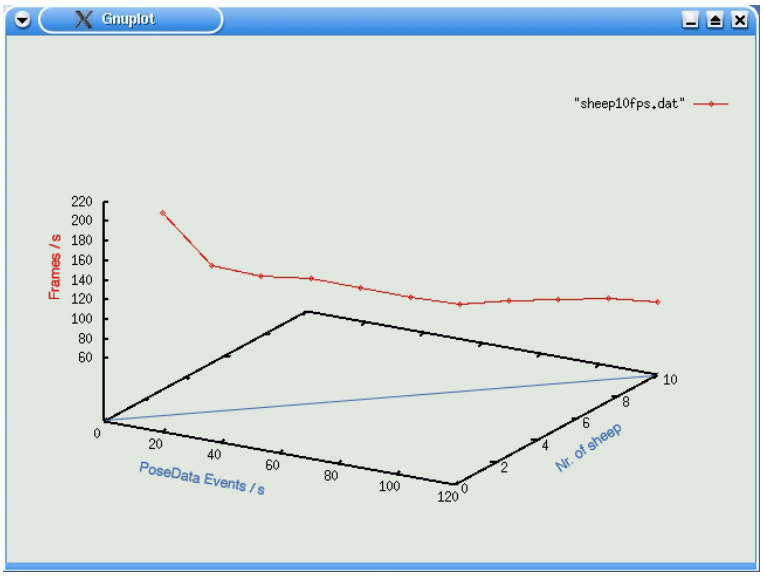
discrepancies across platforms that made us provide several versions of VRML objects for different views. To get acceptable performance on the iPAQ palmtop, we had to compile the EAI libraries of FreeWRL, which are written in Java, into native code with `gcj` [50], yielding a big performance boost from 0.01 to 0.3 frames/sec for a scene containing 162 polygons in a 320 × 200 resolution—which is still too slow, but is mainly due to the lack of hardware graphics acceleration and, even worse, the lack of a floating point unit on the iPAQ. Furthermore, the EAI is not set up to support the required communication traffic (several 100 position updates per second) and the dynamic nature of the scene descriptions when objects might be added and removed at high frequencies. We encountered limitations in the use of the VRML `EXTERNPROTO` mechanism: Because of the EAI requirement that every accessable object has to have a predefined, unique name field, all objects that are potentially created during the course of an interaction have to be planned for in advance when the scene is described in the VRML code. The result is a rigid object structure which not suited for dynamic UAR user interfaces.

Because of these problems, we reimplemented our Viewer based on the open-source Open Inventor [165] implementation Coin3D [168]. This change proved to be a good idea – the flexibility problems were overcome. Also the cross-platform compatibility problems were solved, since the behavior of the Viewers on different platforms proved to be nearly identical. Performance measurements of the Viewer are described in detail in [63]. The results were encouraging. We discuss two sample measurement now.

Both tests (see Figure 4.8) have been run on a Pentium 4 2.5GHz laptop running Linux equipped with 512MB of ram and a ATI Radeon 9000 mobility graphics adapter. They displayed a large, static scene (1500 polygons) and several dynamically moving objects (each 162 polygons). These objects received updates to their position from an external DWARF component. New dynamic objects were constantly added, until the minimum frame rate of 25 fps was reached. During the whole benchmark the viewpoint was set with 20 Hz. Both test were terminated before the minimum frame rate was reached, because the DWARF middleware could not (at the time of the benchmark) handle more connections. Both tests have shown that the Viewer performs well enough for augmented reality systems. This behavior was also verified in several internal and external [147] demonstrations.



(a) Test one used an update rate of 2 Hz for each object. The maximum frame rate was 220 fps without any additional dynamic objects. The minimum frame rate was 36 fps with 15 objects and a total of 50 Hz incoming PoseData events.



(b) Test two used an update rate of 10 Hz for each object. The maximum framerate was again 220 fps with no objects and the minimum was 71 fps with 10 objects and a total of 120 Hz update rate.

Figure 4.8: Benchmark of the Viewer (from [63]): Frames per second, according to the number of objects and their update rates.

4.3 An Example Application: SHEEP

"And what is the sheep seeking?"
"As I said before, I can't express that in words with any precision. What the sheep seeks is the embodiment of sheep thought."
"Is that good"?
"To the sheep's thinking, of course it's good."

HARUKI MURAKAMI, *A Wild Sheep Chase*

To demonstrate the potential of AVANTGUARDE, we have built SHEEP (Shared Environment Entertainment Pasture) [103, 149]. After giving more details about the motivation for SHEEP, we proceed by explaining the interactions that we have realized within this game. Then the implementation of SHEEP is presented. We wrap up by discussing the limitations and contributions of SHEEP.

4.3.1 Motivation

One of the major challenges of current computer systems is to provide users with suitable means to plan, model, and control complex operations which consist of many inter-related processes and dependencies. Multimodal, multiuser interaction schemes are needed to provide adequate control metaphors. It is our hypothesis that tangible user interfaces provide particularly intuitive means for controlling complex systems.

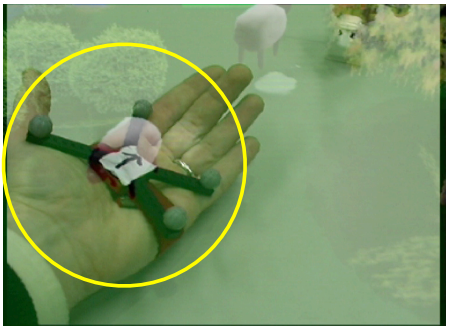
To demonstrate the potential of tangible user interfaces to dynamically visualize, manipulate and control inter-related processes, we have built SHEEP. SHEEP is a multiplayer game centered around a physical table with a pastoral landscape that contains a herd of virtual and physical sheep. The landscape and virtual sheep are projected from a ceiling-mounted projector. Players can assume one of several roles. According to their different roles, players use different input devices and interaction technologies to interact with the game.

Within this system, every sheep is an independent process, communicating with other processes to attract or repell each other. Sheep processes can be created or deleted. They can be visualized and manipulated using various modalities that are packaged. into tangible units. The system was first demonstrated at ISMAR 2002 [149] and formally presented at ISMAR 2003 [103].

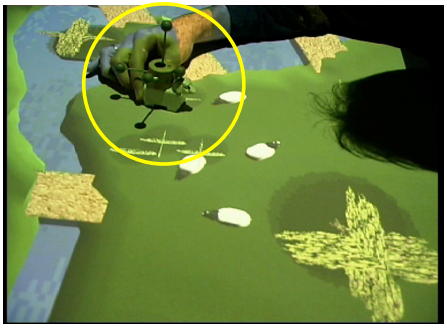
4.3.2 Interactions

The SHEEP game contains numerous interactions which are explained below:

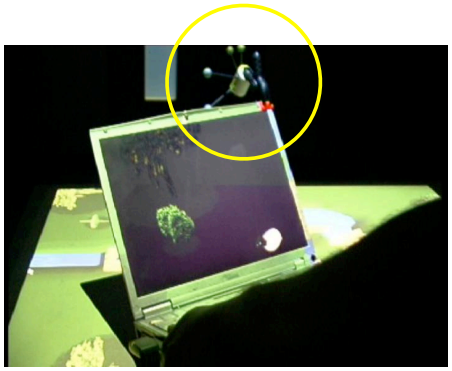
Coloring sheep Figure 4.9a shows the view through an HMD onto the table. A player wearing an HMD can pick up sheep with his tracked hand (note the marker that is attached to



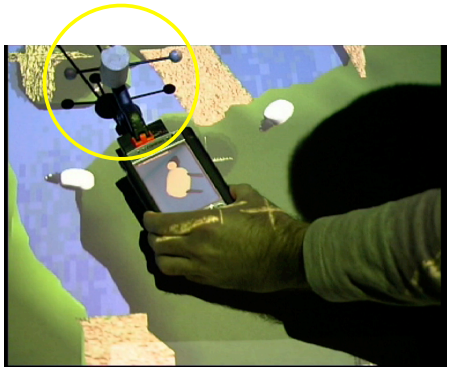
a) View through the HMD while picking up a virtual sheep.



b) Attracting virtual sheep with a tangible sheep.



c) A laptop as windows to the 3D world.



d) Scooping virtual sheep with an iPAQ

e) Demonstration of SHEEP at ISMAR2002

Figure 4.9: Subfigures a) to e) give an overview of the various interactions realized in SHEEP. Additionally infrared-reflective markers (yellow circles) and the camera that is detecting them are highlighted (red circle in figure e).

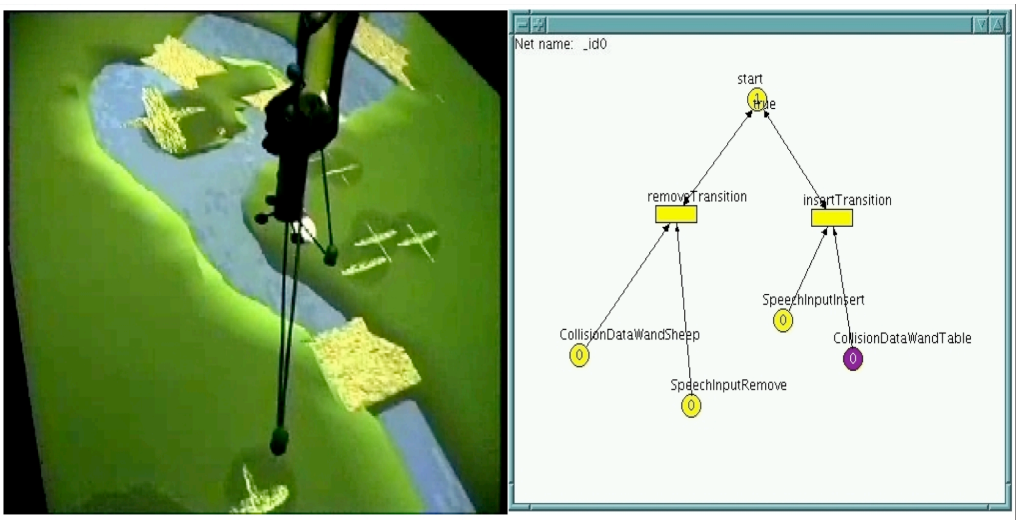


Figure 4.10: Point-and-Speech interaction in SHEEP. (a) The tangible pointing device used to indicate the position of a new sheep, the user then utters "insert" and a new sheep gets created. (b) The corresponding Petri net.

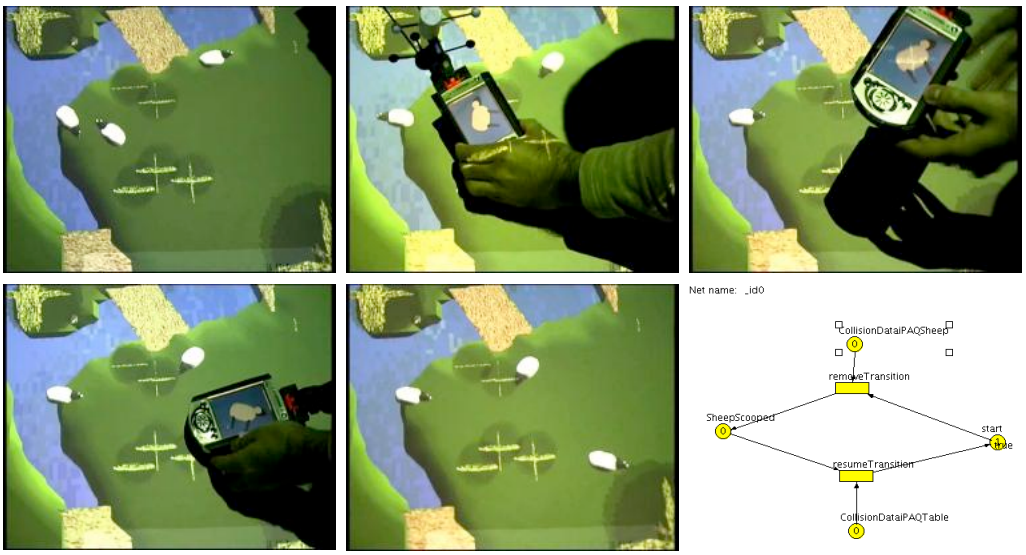


Figure 4.11: Sequence of images for Scoop-and-Drop interaction with a virtual sheep and an iPAQ. In the lower right corner, the corresponding Petri net is shown.

the player's hand) and color them by moving them into color bars shown inside the HMD. After that, he can drop the sheep back onto the table again.

Attracting sheep The scene also allows for physical, tangible sheep. One such object, a small LEGO toy, is shown in Figure 4.9(b). Since all sheep are programmed to stay together, the entire herd can be controlled by moving the tangible sheep—thereby making it the leader of the herd. Moving the sheep around constitutes a tangible interaction that was very popular among users, because of its immediate comprehensability.

Exploring the 3D world A separate laptop can be used to view the scene on the table in three dimensions from arbitrary vantage points (Figure 4.9c). This constitutes a tangible interaction, with the metaphor of moving a window about in the 3D world similar to the active lens of the metaDESK [175].

Creating and removing sheep By putting on a headset with a microphone and grabbing a tracked magic wand, a player can create new sheep and remove sheep from the table. This is done by multimodal *point-and-speak input*. The technical realization and visual sensation for the users is shown in Figure 4.10. This example illustrates how easy it is to model multi-modal interactions in our framework.

Scooping sheep Players equipped with a tracked iPAQ (Figure 4.9d) can use it to scoop sheep up from the table. Scooped sheep can be dropped back somewhere else on the table. During the scooping operation, the scooped sheep is displayed on the palm-sized computer. The entire interaction is illustrated in Figure 4.11. This interaction is similar to Pick-and Drop [142]; however, we use a PDA to pick up virtual objects instead of a stylus,

4.3.3 Implementation

In this section, we present the architecture of the SHEEP system, showing how existing and new DWARF components were used in the areas of tracking, sheep simulation, visualization, interaction and middleware.

The architecture of the SHEEP demonstration system is shown in Figure 4.13.

The basic software components of SHEEP are DWARF services. The services can be divided into the subsystems *tracking*, *sheep simulation*, *visualization* and *interaction*. Many of the DWARF services form adapters to connect to third-party software (shown in gray) for tracking, speech recognition or 3D rendering.

The same service can have one or more instances running in the network. For example, tracking services are running as a single instance, sending positional data to all interested components. Other services, such as user interface controllers or VRML viewers, are provided in many instances. For example, a separate user interface controller is available for each user, and each display has its own VRML viewer. Finally, any number of sheep services can be running on the machines in the network.

The services are distributed on several machines, as shown in Figure 4.12. In DWARF, we follow the tool metaphor [14], bundling software with hardware in units that are easily

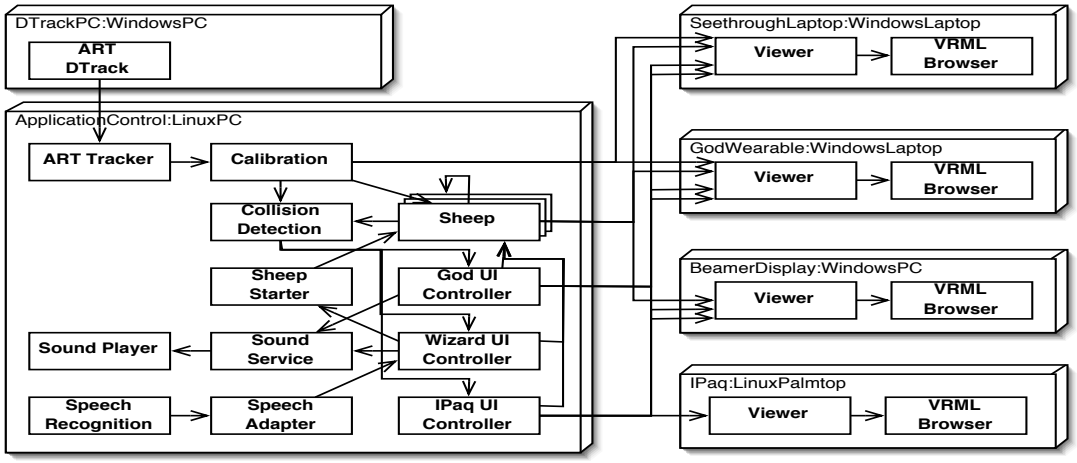


Figure 4.12: System architecture by deployment. The boxes are services and external components; The arrows indicate dataflow. The system consists of three stationary and three mobile computers.

understandable to the user. For example, the palmtop system performs the complete 3D rendering locally (although slowly), rather than retrieving an externally rendered video image from a server. The services run on different machines running Linux, Windows and Mac OS X, and are written in Java and C++. They use CORBA-based middleware to find each other dynamically and communicate via wired and wireless ethernet.

4.3.4 Discussion

SHEEP is a stationary system that makes extensive use of tangible interactions. Synergistic multimodal input was used to improve the usability of the system and coordinated multimedia output was deployed to enhance the immersiveness of the user's experience. Additionally, audio feedback was used whenever possible and a large number of displays was used: tracked laptop, iPAQ, projected landscape on table and a HMD.

This system was the first test of the multichannel abilites of AVANTGUARDE. On a technical level, we were successful in validating our claims about the benefits of AVANTGUARDE. However, no formal usability studies were conducted, so it is not easy to make claims about the usability of SHEEP. From the informal feedback we have gathered, several interesting observations were made.

First, most users of the game preferred interactions that did not require a head-mounted display. Since we were using a state of the art head-mounted display (Sony Glasstron), we can conclude that there is still room for improvement of these devices. There are also other devices

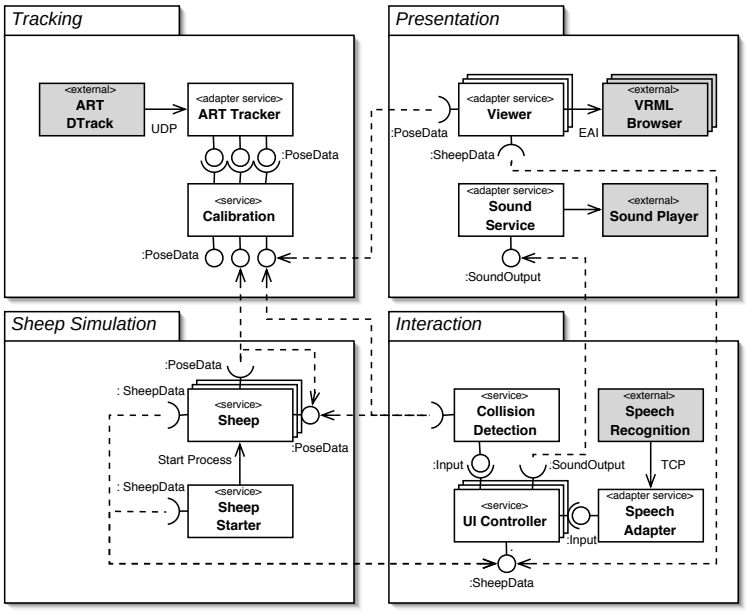


Figure 4.13: System architecture: services arranged by subsystems. Services communicate via their abilities (circles) and needs (semicircles). Third-party components are shown in gray.

that could be used instead, for example retinal displays (e.g.. the Microvision Nomad [109]), that are lighter and have a smaller form factor.

Second, it seems that the ordinary user cares less about performance than do computer scientists. For the implementation of SHEEP, we have used the first version of the Viewer (see Section 4.2.2), which had some performance problems. However almost no ordinary user has complained about this – probably because they were to busy coping with this completely new kind of experience. However most computer scientists we have shown SHEEP to, were fast to remark that there seems to be a lag in the system.

Finally even children could user SHEEP with great ease—some of them even without any explanation. This indicates that our multimodal, tangible user interface in UAR seems easy to use.

4.4 Reflections

The reflections presented in this concluding section are twofold. First the lessons we have learned during the development of DWARF and the planned future improvements are presented. Second, I reflect about the benefits and limitations of AVANTGUARDE.

DWARF

The DWARF framework was developed over several years, involving a large number of developers and end users. We observed anecdotal lessons during development.

Integration of third-party components. The flexibility of DWARF was not only advantageous in requirements elicitation and building prototypes. The loose coupling between components eased the integration of third-party software. Arbitrary pieces of software can easily be integrated in a DWARF system by providing a service wrapping the required functionality.

Developers like design at run time. Most developers readily see the value of design at run time, especially for building prototype systems. It naturally fits the way people work together in building demonstration systems, and encourages integration. However, both development tools and good ground rules for social interaction in design at run time are still issues to be investigated.

Users like design at run time. Many users, once confronted with the new interaction concepts, immediately see potential applications in their area of expertise. When introduced to the SHEEP system, different users have pointed out that the virtual sheep could be replaced with virtual cars, a power plant simulation, a simulation of chemical reactions, software design diagrams, and many other ideas. Users particularly liked the idea of being able to exchange ideas with developers and try out new ideas quickly; however, this has not been tested extensively.

Threshold versus Ceiling. Any software tool can be classified [112] according to two key properties: *Threshold* referring to the difficulty of learning the tool and *Ceiling* denoting how complex are the systems that can be built. The ceiling of DWARF is very high—a wide range of systems has already been built with it. However the threshold of DWARF seems to be very high as well. Although we got very good feedback for some of the development tools we provided, the underlying infrastructure seems extremely complicated. The learning curve for new developers is steep, since they have to deal with the distributed computing paradigm as well as with complex libraries such as scene graphs for augmented reality. This problem has been relieved to a certain degree by providing code examples and templates, but it still remains to be solved.

Performance. While not the fastest augmented reality system available, DWARF's performance has proved adequate to convey the experience of augmented reality to users, at least when using the system for a brief period of time. For deploying production systems, performance optimizations become necessary. In addition, the very nature of a highly distributed system leads to several performance pitfalls that are not as easy for developers to understand as those of a monolithic system.

DWARF has shown its potential for developing highly dynamic applications. We plan on continuing in this direction, applying the experience described in this section to new application domains of augmented reality in real-world settings.

If applications go beyond simple functionality and aim at including business processes, separation of authoring and core development tasks becomes necessary. The DWARF concept of loosely coupled components supports this separation of tasks; however, we have not thoroughly investigated how to apply the DWARF concepts to suitable augmented reality authoring solutions.

DWARF has no explicit data model. Most components such as tracking devices are stateless, enabling an almost exclusively event-based communication. In real-world applications, integration of external data in augmented reality applications becomes essential. We plan on investigating ways to model data sets in a way consistent with the distributed DWARF concepts.

AVANTGUARDE

The abstraction of input devices into logical input devices has a long history. Foley and colleagues have investigated this issue in the 1970s [48] and 1980s [49]. This work has found its way into the CORE graphics standard [162]. However, these works were focussing on WIMP user interfaces, as this was the dominating user interface paradigm of that time. Later work by Mackinlay and colleagues [99] that also addresses virtual environments is more similar to the device abstraction of AVANTGUARDE. Our input taxonomy is much simpler than theirs. We decompose inputs in four different types, whereas they define an input device to be a six tuple, where each element of the tuple can take several different values. However, for the interactions that we have implemented, our taxonomy proved sufficient. Combined with the Spheres of Influence model and its DWARF-based implementation, we are able to implement highly dynamic interactions. A bigger problem in our implementation is the handling of output devices. It would be very convenient to have the same kind of taxonomy for output devices—enabling an even higher degree of abstraction towards the output devices. To find an applicable taxonomy and to integrate it into AVANTGUARDE is future work.

One of the major goals of our research is to provide a rapid prototyping environment within which new user interface ideas can be prototyped and tested easily. To this end, the current setup has already proven to be suitable for joint, online development, testing and enhancement of both individual interaction facilities and multimodal, ubiquitous combinations thereof. The SHEEP game was partly developed in such joint sessions, which we called *Jam sessions* [103].

Furthermore, Kulas has demonstrated that usability evaluations can be seamlessly integrated into the live development and testing process [86]. To this end, user evaluation processes can be created to automatically inspect and evaluate the data streams flowing between the individual interaction devices, tangible objects, users, displays, etc.

As a next step, the user interface architecture (Figure 4.3) can serve as the basis to dynamically integrate further, in-depth enhancements to the analysis and interpretation of user input. By including tools to track a user's gestures or mimics (e.g., by eye-tracking), a cognitive model of the user can be accumulated. Combined with further sources of environmental context data, the User Interface Controller can be extended to react adaptively to changing situations. By extending the information presentation primitives of the 3D Viewer, new presentation metaphors, as well as context-dependent layouts of information within an augmented

environment can be provided. Note that all of these enhancements can be included, tested, modified and/or rejected online and non-exclusively.

The result is a live, dynamically changeable and also dynamically adaptive development environment for UAR user interfaces. The environment can thereby provide us with the opportunity to easily explore, combine and test different concepts that are currently emerging while also providing increasing degrees of automatic adaptation by the tools themselves. We expect this flexible, dynamically changeable setup to be able to provide us with the tools to generate and test new concepts much more rapidly and flexibly. The next chapter describes the steps that we took towards realizing this vision.

It [the computer] is a medium that can dynamically simulate the details of any other medium, including media that cannot exist physically. It is not a tool, although it can act like many tools.

ALAN KAY

Tools for Interaction Development with AVANTGUARDE

This chapter presents several tools built on top of AVANTGUARDE to ease the tasks involved in developing new interactions. Their design space ranges from conventional graphical user interfaces to user interfaces in UAR. These tools can be flexibly combined according to the task to be addressed; thus, providing a toolbox to developers and users.

This chapter presents several prototypical tools for interaction development that are built on top of AVANTGUARDE. First, Section 5.1 highlights the vision, which has driven the development of the tools. We have created tools with different user interface paradigms that can be combined into more powerful tools; thus, creating a toolbox of hybrid, cross-paradigm tools.

Typical tasks that are necessary to develop interactions with user interfaces in UAR are monitoring the user during employment of a user interface (Section 5.2) and adjusting the details of that user interface.

The task of changing the parameters in a running UAR user interface executed in AVANTGUARDE can be divided into three parts: configuring dataflow networks (Section 5.3), specifying dialog control (Section 5.4) and creating context-aware animations (Section 5.5).

The reflections in Section 5.6 compare the benefits and limitations of our toolbox.

5.1 Overview

This introductory section gives an overview of the tools that are presented in this chapter. First, we highlight our overall vision. Second, we give an overview of the relationships between our tools. Finally, we explain the criteria that will be applied in discussing their benefits and limitations throughout this chapter.

The motivation for our tools is a novel idea. We would like them to form a toolbox of lightweight, flexible tools that can be combined by the user interface developer as necessary. They employ a variety of user interface paradigms to support developers in their tasks: tangible user interfaces, augmented reality and conventional WIMP user interfaces. Each of these paradigms has its own benefits and limitations. However, with the appropriate combination of tools, the maximum benefit for the user interface author can be achieved.

For example, tangible user interfaces are considered very easy to use [176], however they are limited to some extent. Considering the tangible answering machine (Figure 1.2): what happens, when there are more messages than available marbles? In general, it can be difficult to create tangible user interfaces for tasks that would be very easy to implement with other paradigms. Almost every WIMP user interface for file system access (e.g., Windows Explorer, Linux Konqueror or the Macintosh Finder) allows users to create a hierarchy of a (practically) infinite number of folders. This would be very hard to implement with a tangible user interface. It might very well be true that an easy to use tool for file system access could be implemented with a combination of a WIMP user interface (e.g., for implementing the huge number of folders) and a tangible user interface (e.g., for sorting files). Similar trade-offs can be found between other user interface paradigms. WIMP user interfaces can provide a very high resolution, due to the advances of display technology. On the other hand, augmented reality visualizations are currently limited to low resolution, since head-mounted display technology is still immature. But thinking about the idea to put information where it is needed, mobile augmented reality user interfaces are much better suited than WIMP user interfaces for tasks that require the user to be mobile.

Another perspective of our approach is the analogy to hybrid user interfaces. These combine visualizations on different displays modalities into a more powerful visualization than each modality would be able to deliver on its own. Similarly, we try to create tools with different user interface paradigms that can be combined into more powerful tools—thus, creating hybrid, cross-paradigm tools.

An overview of our tools is depicted in Figure 5.1. For each tool, the task it addresses and the user interface paradigm it employs are shown. A short summary of these tools is:

T1. A WIMP tool that can collect and evaluate usability data during system runtime [87] (Section 5.2.1).
T2. A tool using an augmented reality visualization [114] (Section 5.2.2) that makes it possible to clearly see the visual attention of a user with a combination of head- and eyetracking.
T3. The graphical editor DIVE (Section 5.3.1) allows the user to adjust dataflow networks.
T4. The immersive visual programming environment [147, 150] (Section 5.3.2) addresses the same task.
T5. The User Interface Controller Editor [64] (Section 5.4) makes it possible to specify dialog control.
T6. A set of tools to experiment with context-aware mobile augmented reality user interfaces (Section 5.5). It is a an example of a combination of tools following different user interface paradigms.

The remainder of this chapter presents these tools. In each section, we present the motivation, technical realization, and an example of use for the corresponding tool. We wrap up the presentation of each tool by discussing its benefits and limitations. Our main criteria for the discussion of each individual tool will be the notion of threshold and ceiling. The threshold refers to the ease of use, whereas the ceiling refers to the limitations of what can be reached.

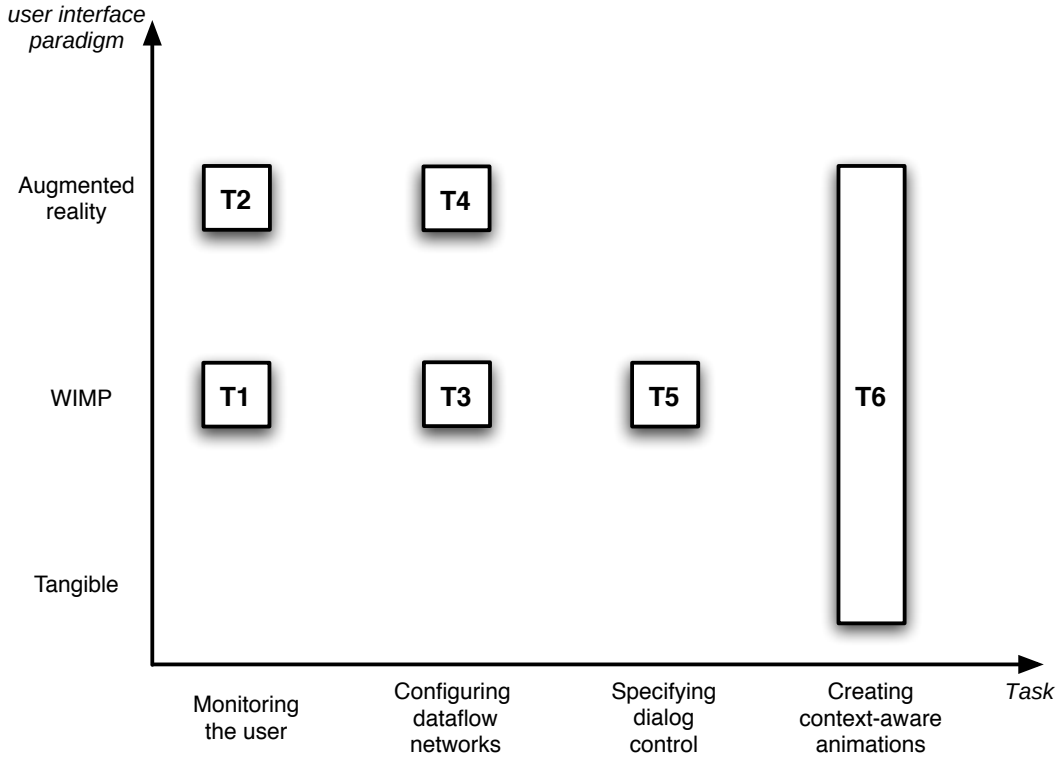


Figure 5.1: Classification of implemented tools. Development tasks are addressed with tools that use different user interface paradigms.

For example, the tangible answering machine is very easy to use, however it is quite limited. Thus, the threshold is low, but the ceiling is low, as well.

All of these tools are following the development at runtime idea. These tools are highly experimental, since they build on an experimental infrastructure and an experimental development process. We have not conducted any usability studies, yet. This makes it impossible to make claims regarding the usability of the tools—however, we present anecdotical evidence about the benefits of using these tools.

We also put each tool in perspective to ideas that have been presented earlier. We discuss how it fits into the overall idea of development at runtime (Section 2.2) and how the features of Avantguarde (Chapter 4) were helpful in developing them.

Finally, in Section 5.6, we discuss to what extent we were successful in realizing our overall vision.

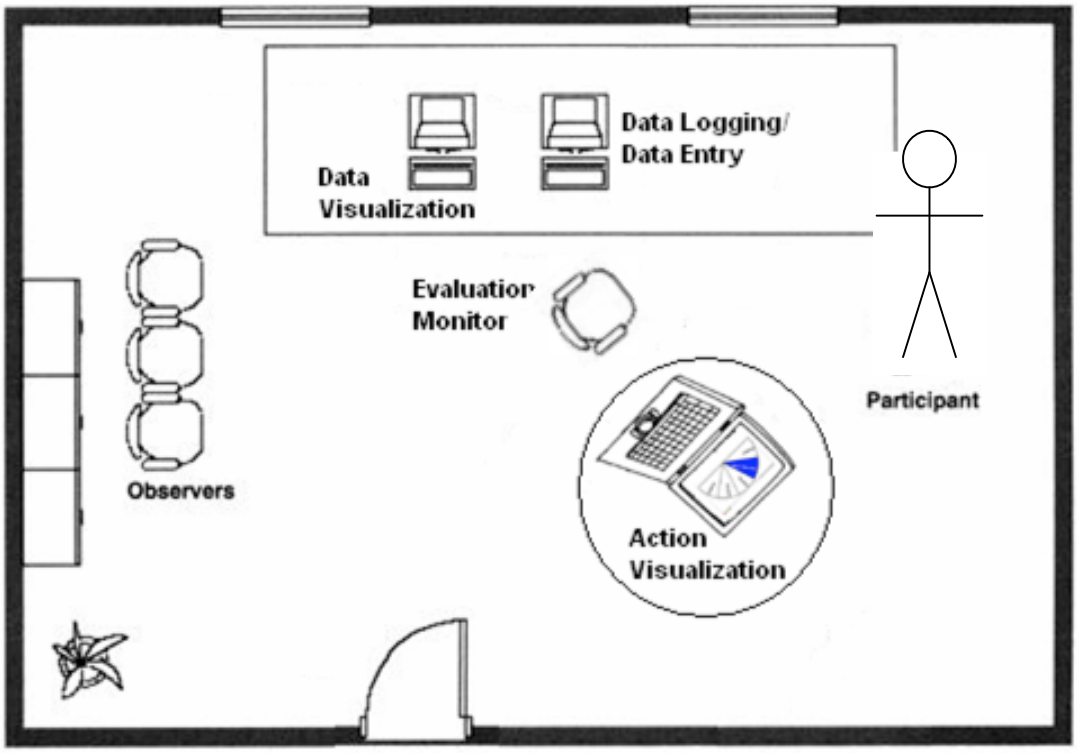


Figure 5.2: Room setup for usability evaluations.

5.2 Monitoring The User

To support development at runtime, monitoring tools that provide immediate feedback about the user's interactions are very helpful. This section presents two such tools: first, a WIMP tool that can collect and evaluate usability data during system runtime [87] (Section 5.2.1). Second, a tool using an augmented reality visualization is presented [114] (Section 5.2.2) that makes it possible to clearly see the visual attention of a user with a combination of head- and eyetracking. The distinction between these two tools is not only the kind of user interface they employ, but also the intended timeframe. While the WIMP tool can be used to explore the usage patterns a user applies during a *longer period*, the augmented reality tool can only be used to collect feedback over a *short period*.

5.2.1 Exploration of Usage Patterns

We have created a WIMP tool to discover usage patterns employed by a user. This section consists of three parts. First, we explain the intended setup for this tool; second, we give

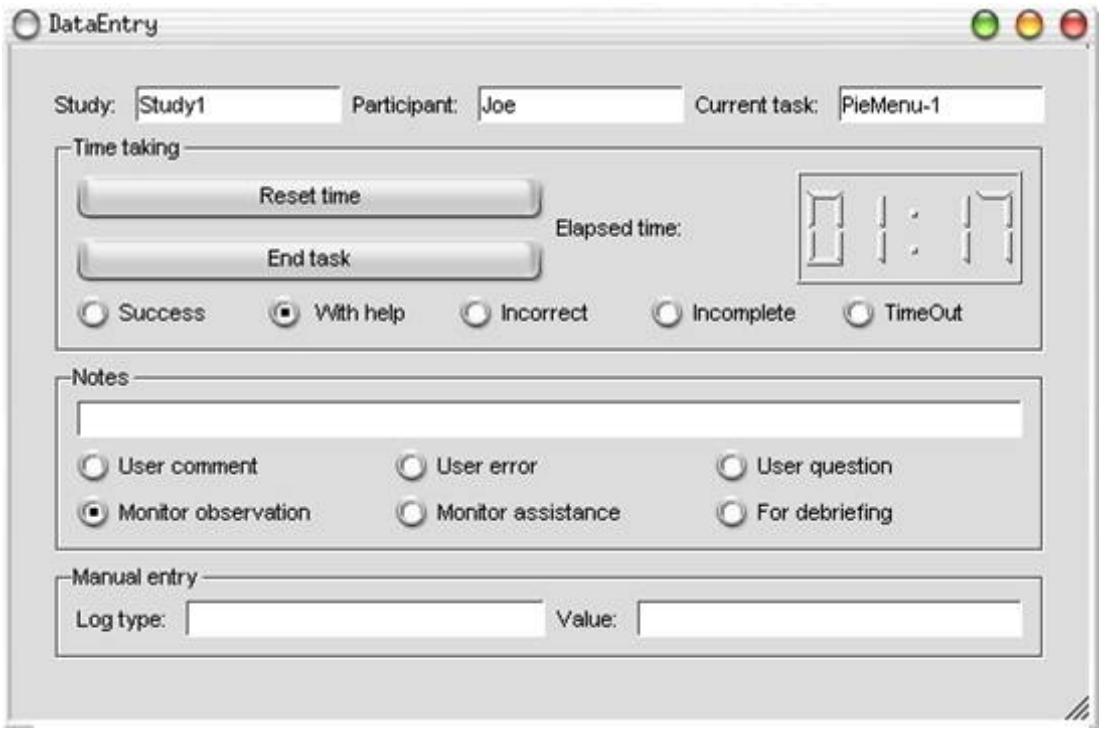


Figure 5.3: Dialog to enter usability data manually.

details about its implementation. Finally, we highlight possible future extensions.

Configuration

We start with a quick overview of the intended setup for conducting usability evaluations *while a group of developers* is working on an UAR user interface. A possible room setup is depicted in Figure 5.2.

The user is placed at a suitable distance from the usability engineer, who is busy entering observations into the usability logging system while also monitoring what the user actually sees on screen and while monitoring real-time visualizations of measured usability data. Multiple developers might at the same time observe internal system behavior and even fine tune the system on the fly while observing usability implications immediately.

This lab-based approach is usually considered as 'Local Evaluation' with both the user and the usability engineer in the same place at the same time. We believe, this is still the best way to capture qualitative usability data on UAR user interfaces. However, when UAR systems are used on a more frequent basis globally, a remote evaluation approach using Remote Usability Evaluation Tools [85] might be more reasonable. Here the system usually presents a wizard-based dialog to the user, asking her details about her opinion on the usability problem after

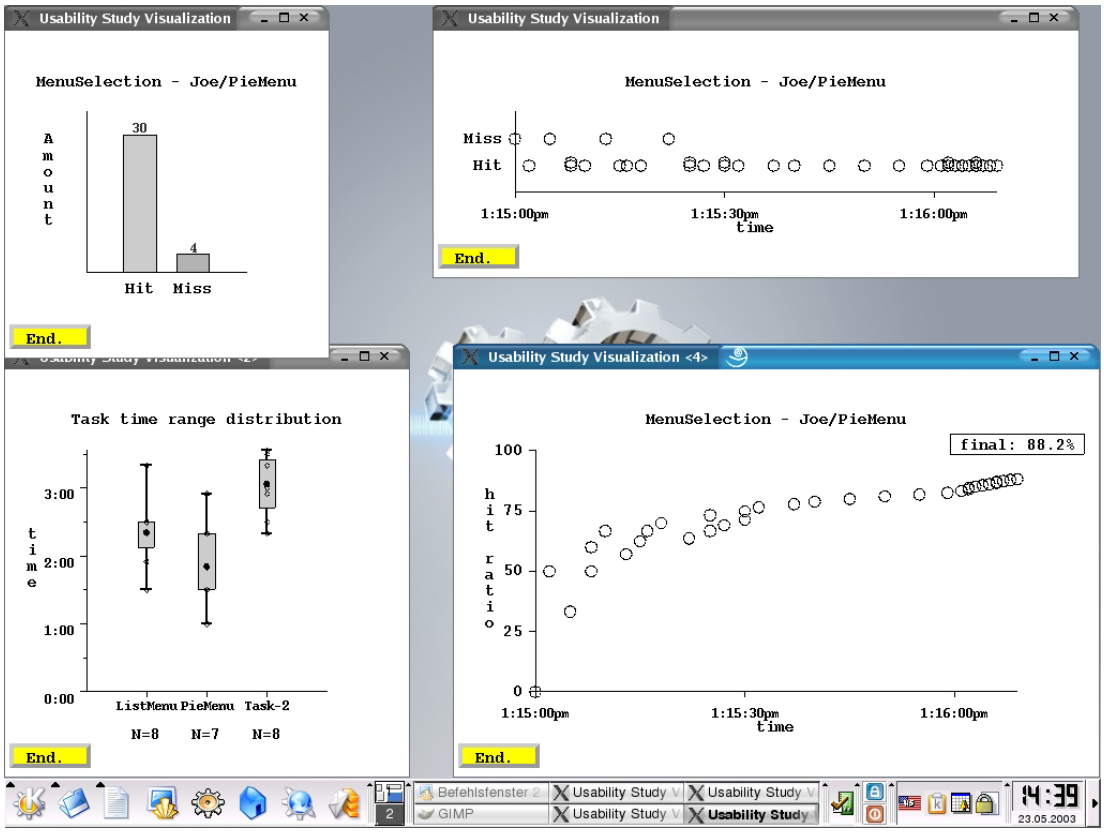


Figure 5.4: Sample realtime visualizations of logged usability data.

recognizing a critical usability incident [59] automatically or after the user triggered the dialog herself. By design, this requires quite a lot of effort on the part of the user herself. Additionally, great care has to be taken regarding issues of user privacy, since passing on collected data without her prior consent is not desirable.

Implementation

With this in mind, the mentioned software components are now covered in more detail. The core component is a fully automated logging tool to capture events from the running UAR system. Additionally, a manual data entry tool (Figure 5.3) can be leveraged to take quick written notes for later review, which are also directly passed on to the data logging component. All performance measurements can finally be visualized in real-time during usage with a number of highly flexible and adaptable scripts. Figure 5.4 shows a number of different sample visualizations. The visualization types from top left to bottom right are:

Absolute Bars Requiring the *study*, *task* and *type* parameters, absolute totals (y-axis) of all different *values* (x-axis) are rendered in horizontal bars. If no *user* is specified, it will output average bars together with a specification of how many users the script averaged over. However, if a *user* name is given, it will output bars using data from this specific user only.

Value Timeline This visualization has the same parameter requirements as the *Relative Error* visualization. Here it is merely shown which event type *value* (y-axis) occurred at what time (x-axis). Figure 5.4 actually shows a slight modification of this basic visualization. An additional line visualizing a study specific additional event *type* and *value* development was added effortlessly to be able to better spot usability flaws of a specific nature [86].

Task Time Range A visualization of all task completion times (y-axis) for all tasks (x-axis) averaged over all participating users. These times are extracted from the log file by filtering for special event *types*, which mark the begin and end of any given *task*.

The actual ranges can be easily visualized in different ways. Shown is the mean and standard deviation. The biggest dot indicates the mean times while the error bars extend to the standard deviation. The smaller light dots show the individual task completion times of all *users*. The stars denote task completion times outside of the standard deviation. Finally below each *task* a number is printed, which depicts the number of averaged tasks, which is equal to the number of *users* who performed this *task*.

We also prepared a median version which renders a big dot at the median time for each task while the box-plot extends to the 25th and 75th percentile. Error-tails extend to the border values while smaller light dots show the individual task completion times for all *users*. All range visualization parameters can be easily adopted on a case-by-case basis.

Relative Error This script is the least flexible, since it requires the *value* fields to be exactly Hit or Miss for the *study*, *user*, *task*, and *type* combination to be analyzed. It visualizes the resulting accumulated hit-ratio (y-axis) over time (x-axis). The final hit-ratio is additionally printed separately in a box.

To acquire the relevant usage data, our UAR systems, which are built modularly leveraging the DWARF framework, communicate internally mostly by means of CORBA-based events running through event channels which can be easily tapped into by any logging tool interested in doing so, such as the newly developed logger.

To visualize the collected usage data, we have decided to base the visualization on the third party tool *ploticus* [128] for multiple reasons. Its scripting language proved to be well suited for rapid prototyping of new visualizations while maintaining a high level of ease of use. Additionally, it already had all the 2D plotting support we required, since it supports out of the box all standard 2D plotting styles including line plots, filled line plots, range sweeps, pie graphs, vertical and horizontal bar graphs, time lines, bar proportions, scatter plots in 1D or 2D, heat-maps, range bars, error bars, and vectors. Numerics, alphanumeric categories, dates

and times (in a variety of notations) can be plotted directly. There are capabilities for curve fitting, computing linear regression, and Pearson correlation coefficient r. There is a built-in facility for computing frequency distributions. Means, medians, quartiles, standard deviations, etc. can also be computed out of the box meeting our needs for default statistical functions.

Discussion

Sample usability study results [86] using the above tools were very promising. It was easy to use, however, to modify the evaluation scripts is still too difficult to be handled by complete novices, because ploticus comes with a custom scripting language. It would be desirable to allow the usability engineer to change these visualizations on the fly (e.g., by clicking on a map representing the system state and exchanging events).

The modular conception of AVANTGUARDE made the implementation of this tool very easy. We have just created one component that had needs for all occurring data streams—thereby collecting all required data to be visualized.

5.2.2 Visualizing the User's Attention

User interfaces for UAR are different from user interfaces for desktop environments because the main amount of visual attention must be spent on the real world. Only bursts of attention can be directed towards the computer interface. Moreover, input devices like conventional mice and keyboards are not available. It is desirable that the system should take the user's focus of attention into account in order to achieve a more intuitive communication. For example, an attentive user interface (AUI) should try to gain the user's attention and wait until she focuses on it (e.g., by turning her head towards an interactive icon like a blinking telephone icon that shows an incoming phone call). Zhai and Jacob have shown that the eye is, in principle, not very suitable to control user interfaces [185, 76]. Instead of controlling, we use the eye direction to notify the system about the user's current focus of attention. By looking at a device, a user conveys her attention for that device and the AUI can adapt its behavior accordingly.

This section consists of three parts. First we explain the intended setup for this tool, second we give details about its implementation and explain an example application of this tool from within the CAR project. Third we highlight possible future extensions.

Configuration

When developing highly interactive user interfaces, knowledge from many different research areas such as human factors, human-machine communication and computer science is required. Based on their experience, the development team members need to discuss, simulate and evaluate different strategies for attracting the user's attention in the least disruptive and most consistent way. To this end, a rapid prototyping environment is needed which allows domain experts to easily configure and modify different attentive user interaction schemes without requiring major changes to the underlying computer system.

Figure 5.5: Virtual cones show the user's attention. This application is used to monitor the deployment of visual attention.

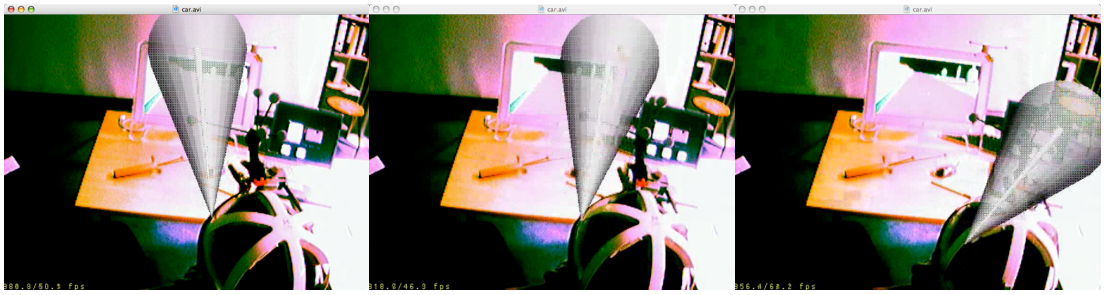


Figure 5.6: Visualization of the user's attention with virtual cones. First, the user's attention is set on the driving task. Eventually, a flashing icon in the AUI causes the user to look at it and provides her with additional information.

The desired setup should allow teams of domain experts to jointly develop and exchange interaction strategies at runtime, thereby saving a lot of time between design iterations. Using a visualization that augments a user's current view with a viewing cone (see Figure 5.5), the design team can evaluate the efficiency and effectiveness of various attentive strategies and design elements (e.g., level of details, forms, colors, brightness, size, etc).

Systems that use attention as an input have been presented already [17], but with our tool one could visualize why certain behavior of the system occurs. The main contribution to current eye tracking methods [76] is that our visualization method with superimposed cones shows the eye direction within the scope of the application instead of snippets of the environment viewed by the eye camera. Moreover, it can be used for monitoring multi-user applications by augmenting all tracked users. This project focusses on exploring interruptive qualities (not disruptive vs. effective attention requests) and maximizing usability while a user's attention is divided between several tasks.

Implementation

Our tool for attentive user interfaces receives input from a commercial tracking system [1] and a custom built eye tracking device [171] to obtain the user's head position and eye direction. The eye direction is visualized as a virtual cone and superimposed on the video of the user (Figure 5.6). The virtual cones represent foveal and parafoveal areas, and the outer boundaries of these areas are generally set at $\pm 1^\circ$ and $\pm 6^\circ$ degrees of visual angle, respectively.

Our tool was used within the CAR project (see Section 5.5) to experiment with attentive user interfaces for car drivers. A typical scenario that we address is:

> After receiving an event (e.g., phone call, event reminder, or information) the system presents initial information on a peripheral display, trying to attract the user's attention. The application remains idle, until the user looks at the peripheral display. To determine whether the user focuses on the attentive icon, the position of the display is also registered in the laboratory setup. When the user moves her head, the intersection of her viewing cone with the plane representing the display is computed. As soon as the cone intersects with the display area showing the attentive icon, an event is generated upon its behalf. As a reaction to the eye contact, further action to respond to the inital event (e.g., a phone call) is taken.

The visualization for this scenario is depicted in Figure 5.6.

Discussion

In this section, we have presented our AR workbench. It has been used in our laboratory to build, evaluate and improve an AUI for cars. Our visualization methods facilitated the comprehension of the AUI concept and thus motivated communication among team members. This approach can be applied in all physical environments with 3D tracking, and can therefore be used for evaluation of all kinds of user interfaces and devices by observing user's eye direction

in the given context. In future work, we want to implement further strategies to attract the user's attention and conduct user studies.

Several components of AVANTGUARDE were useful when developing this tool. The Viewer component was used for visualizing the augmented reality overlays. The discussion of the example from the CAR project showed, that this tool works together nicely with the User Interface Controller component.

5.3 Configuring Dataflow Networks

A common task in changing the parameters of a running UAR user interface executed in AVANTGUARDE, is to configure the dataflow networks that occur. For this purpose, the graphical editor DIVE (Section 5.3.1) will be briefly presented. We have also created an immersive visual programming environment [147, 150] (Section 5.3.2) that addresses the same task.

5.3.1 DWARF's Interactive Visualization Environment

DWARF's Interactive Visualization Environment (DIVE) was initially developed by Daniel Pustka [133] (advised by Asa MacWilliams) and later refined by Markus Geipel [53] (advised by Asa MacWilliams and co-advised by me). The two main goals for DIVE were:

1. Enable developers to *monitor* the dataflow network of components.
2. Enable developers to *adjust* that dataflow network.

In this section, we first explain the facilities offered by DIVE to monitor a dataflow network; then, we present the facilities for adjusting dataflow networks.

Monitoring a Dataflow Network with DIVE

Figure 5.7 shows the visualization of a set of connected DWARF components. Additionally the needs, abilities and attributes of components are displayed.

Services can also be grouped according to several criteria (e.g., subsystem or the machine they are running on). Figure 5.8(a) shows a set of services grouped by hosts. To get an overview of all services connected to a DWARF network, a list view has been implemented showing all those services (Figure 5.8(b)).

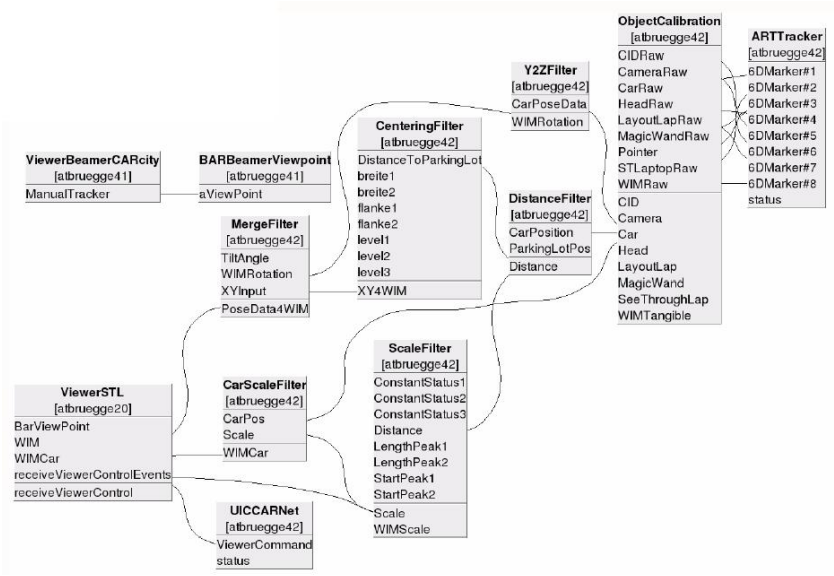
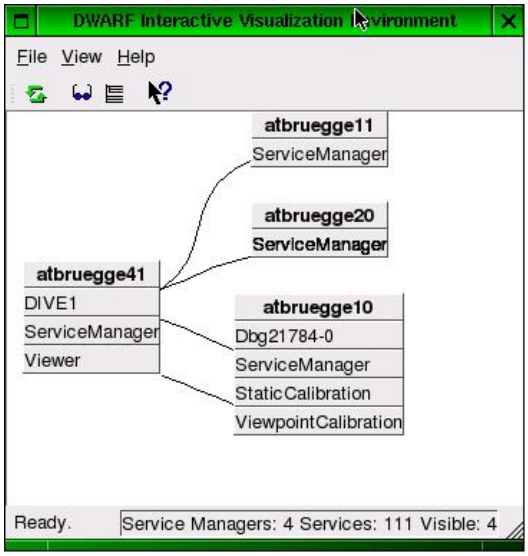
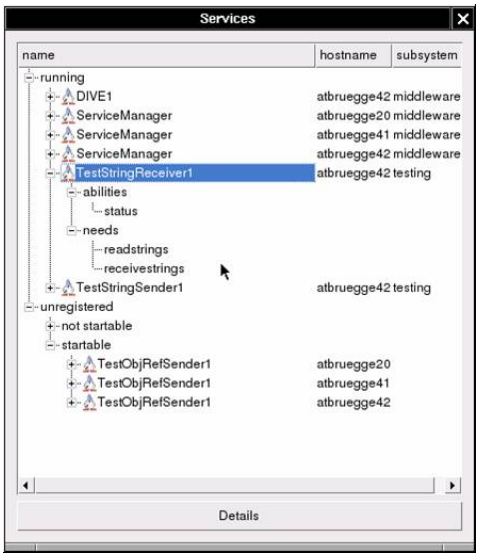


Figure 5.7: A network of cooperating services established in the DWARF system and visualized via DIVE (from [53]).

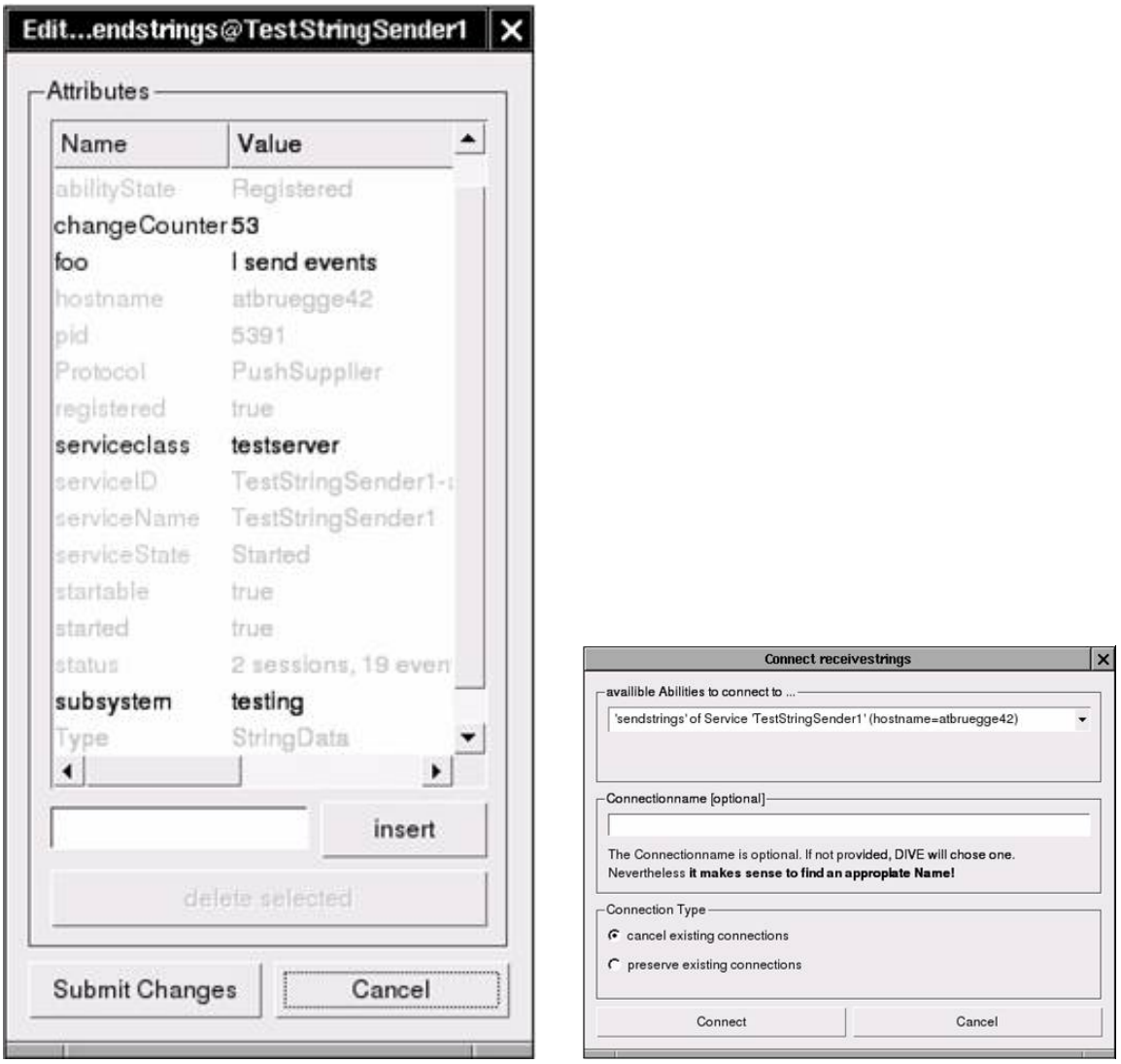


(a) A set of components grouped by the host they run on.



(b) A list of all services connected to a DWARF network.

Figure 5.8: Advanced monitoring features of DIVE (from [53]).



(a) Changing the attributes of a service.

(b) Manually connecting components.

Figure 5.9: Adjusting dataflow networks with DIVE (from [53]).

Adjusting a Dataflow Network with DIVE

DIVE also offers facilities to adjust the dataflow network between DWARF components. To change the needs, abilities or attributes of a component, the developer just has to click on a component and he will be presented a GUI for changing these values (see Figure 5.9(a). These changes can affect the connectivity structure of components. A more direct approach to change the connectivity structure is depicted in Figure 5.9(b). Via a menu entry, a developer can bring up a GUI for establishing new connections for a component.

Discussion

DIVE makes it possible to adjust dataflow networks. To use DIVE, the developer has to have a deep understanding of DWARF and distributed systems—this makes it rather complicated to use. However, in the context of developing applications with DWARF and AVANTGUARDE, DIVE proved to be the favorite tool among developers. All developers of DWARF or AVANTGUARDE systems were constantly using DIVE for monitoring the current state of the distributed system.

For our development at runtime process, DIVE also proved to be very important. Especially when exchanging components during system runtime, connecting the new components to the current dataflow network was typically done using DIVE.

In my opinion, the biggest usability problem of DIVE is the visualization of huge dataflow networks. Whenever the networks do not fit on the screen anymore, usability greatly decreases. To address this problem, I suggest using a zoomable user interface [124] for DIVE. This way, it could be guaranteed that the dataflow networks almost always fit on the screen, thereby increasing usability a lot.

5.3.2 Immersive Configuration

This section presents the first steps toward building a mixed-reality system that allows users to configure the dataflow networks of user interfaces in UAR. Thereby creating a tool with very similar functionality to DIVE, presented in Section 5.3.1; however, we tried to create a tool that is much easier to use.

The work in this section has been done collaboratively with Alex Olwal, Blaine Bell, Nick Temiyabutr and Steven Feiner and has already been published [147, 150]. My part of this work was the overall conception of this tool. Also, I have implemented most parts of the system (see the white components in Figure 5.12). The content in this section is very similar to the primary publication about it [150]. However, a short discussion of my contributions to this work has been added.

The remainder of this section is structured as follows: we start by explaining the motivation for our prototype in more detail, then we present its interaction design. We proceed by explaining some implementation details. Then, my contributions to this prototype are discussed. Finally, we discuss the benefits and limitations of this tool.

Motivation

A key idea underlying our work is to immerse the user within the authoring environment. Immersive authoring has been explored by Lee and colleagues [91] (see also Section 3.2.3), in a system that has a wider range of possible parameters than we currently support. While their system is restricted to a single view and interaction with real objects is limited to ARToolkit [67] markers, our system supports multiple coordinated views with different visualizations and interaction with a variety of physical controllers.

As a first step towards allowing end users to configure the dataflow network of user interfaces in UAR, we have realized a specific scenario, which we support with an augmented reality overlay, presented on a head-tracked, see-through, head-mounted display. In our scenario, a user interacts with physical input devices and 3D objects drawn on several desktop displays. The input devices can be configured to perform simple 3D transformations (currently scale, translation, and rotation) on the objects. The user's see-through head-mounted display overlays lines that visualize dataflows in the system, connecting the input devices and objects, and annotates each line with the iconic representation of its associated transformation. The user wields a tracked wand with which she can reconfigure these relationships, picking in arbitrary order the three elements that comprise each relationship: an input device, a 3D object, and an operation chosen from a desktop menu.

Interaction Design

We address two general issues in designing a reconfigurable user interface: presenting appropriate feedback and supporting interactive reconfiguration.

Visual Feedback We developed a simple graphical language to visualize the relationships between objects, the input devices that control them, and the associated operation. This provides the user with an intuitive overview of the active mappings in the environment.

A real object (or its virtual representation) is visually connected with a tracked controlling input device through a line that can be seen in the head-mounted display. An iconic representation of the currently assigned operation is attached to the line. Figure 5.10(a) shows a view of the interaction and its overlay as seen from the vantage point of another user. To avoid clutter, we show a relationship's visualization only while the user manipulates its associated input device or reconfigures its mapping, as described below.

Some of our input devices are not tracked, and, therefore, their locations are unknown. We represent an untracked input device by an image that is screen-stabilized (fixed to the coordinate space of the head-mounted display), and draw a line from the appropriate part of that device to the virtual object that it controls, as shown in Figure 5.10(b).

Interactive Reconfiguration The user can modify existing relationships or create new ones interactively with a tracked wand, as shown in Figure 5.10(c). To establish a relationship between a physical device and a virtual object, three attributes need to be chosen (in any order): an

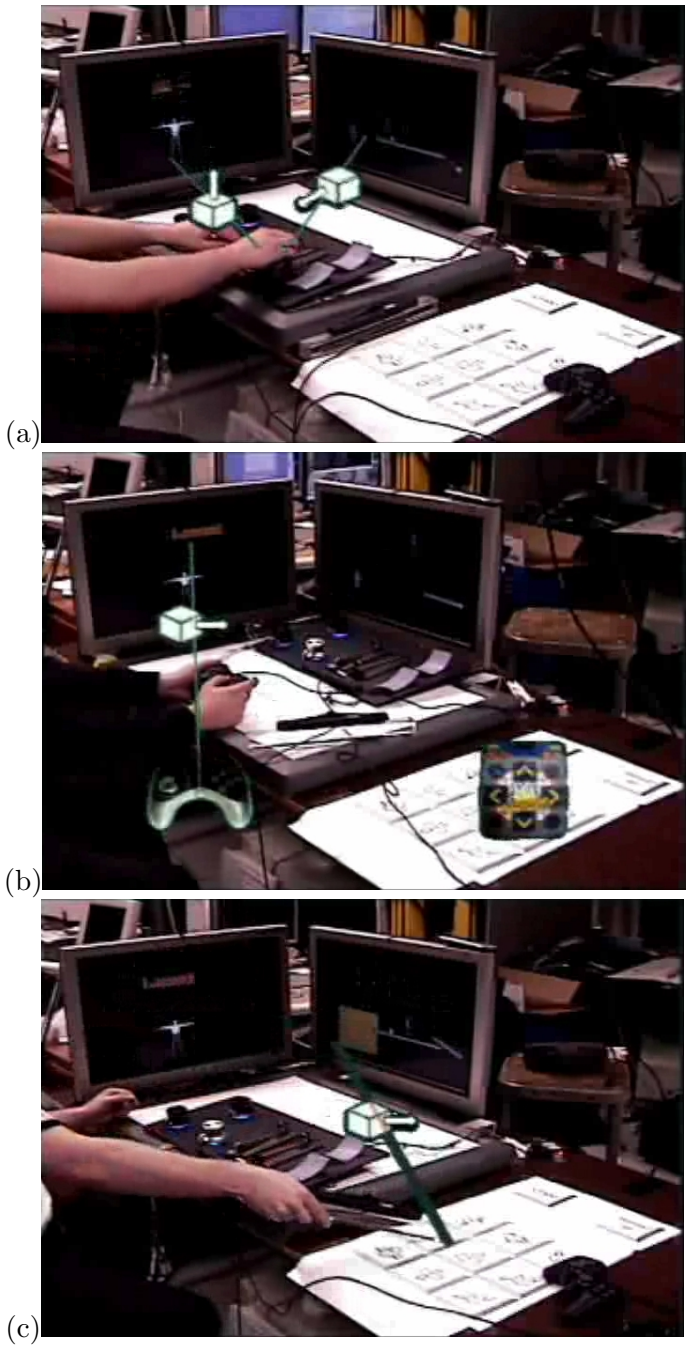


Figure 5.10: Videomixed view through another user's tracked, see-through, head-mounted display. (a) Lines show the dataflow between tracked input devices and virtual objects. (b) Untracked input devices are shown as screen-stabilized models. (c) A tracked wand is used to reconfigure the dataflow network.

Input			Line Output		
Target	Device	Function	Begin	End	Icon
		✓	bwand	twand	✓
	✓		twand	device	
	✓	✓	twand	device	✓
✓			twand	target	
✓		✓	twand	target	✓
✓	✓		device	target	
✓	✓	✓	device	target	✓

Legend	
bwand	beginning of wand
twand	the wand's tip
device	physical device
target	Virtual object

Figure 5.11: Table specifying visual feedback provided during reconfiguration of tracked devices.

operation, a physical device, and a target virtual object. The physical device and target virtual object are selected by moving the wand within the proximity of a physical object or a virtual object's projection on a physical display, triggering highlighting and voice feedback. The operation is selected by moving the wand's tip to one of the operations displayed on a printed 3×3 grid at a known location on the desk. Our menu, inspired by the printed wall-mounted menu of [178], allows the specification of translation, rotation, and scale along the x, y, or z axis.

Figure 5.11 represents all possible states that can occur when a relationship with a tracked device is being configured: the input (target object, input device, and operation) is mapped to how the connection is displayed (the beginning and end points of the line, and whether the icon is shown). For example, Figure 5.10(c) shows the state of our system when both the target object (highlighted on the screen) and the operation (displayed as an icon) are selected, the user still needs to select the input device, and a line is drawn from the wand tip to the target object, as specified in the highlighted line of Figure 5.11. For untracked controllers, touching a 'Learn' button causes the next device manipulated by the user to be selected for assignment. (In contrast to our fixed printed menu, we are experimenting with projecting the Learn button on the desk such that it automatically avoids being occluded from the user's viewpoint by the tracked board [18].)

Implementation

Our prototype is built using a set of existing frameworks. The overall architecture and most of the components are taken from AVANTGUARDE [148], the input device handling is inspired by Unit [118], and the material presented on the head-mounted display uses the DP (data programming) framework [18], as shown in Figure 5.12. We chose this mapping based on the strengths of each framework. The head-mounted display view relies on the ability to easily specify rules in DP, while the many input devices supported by Unit made it a natural match

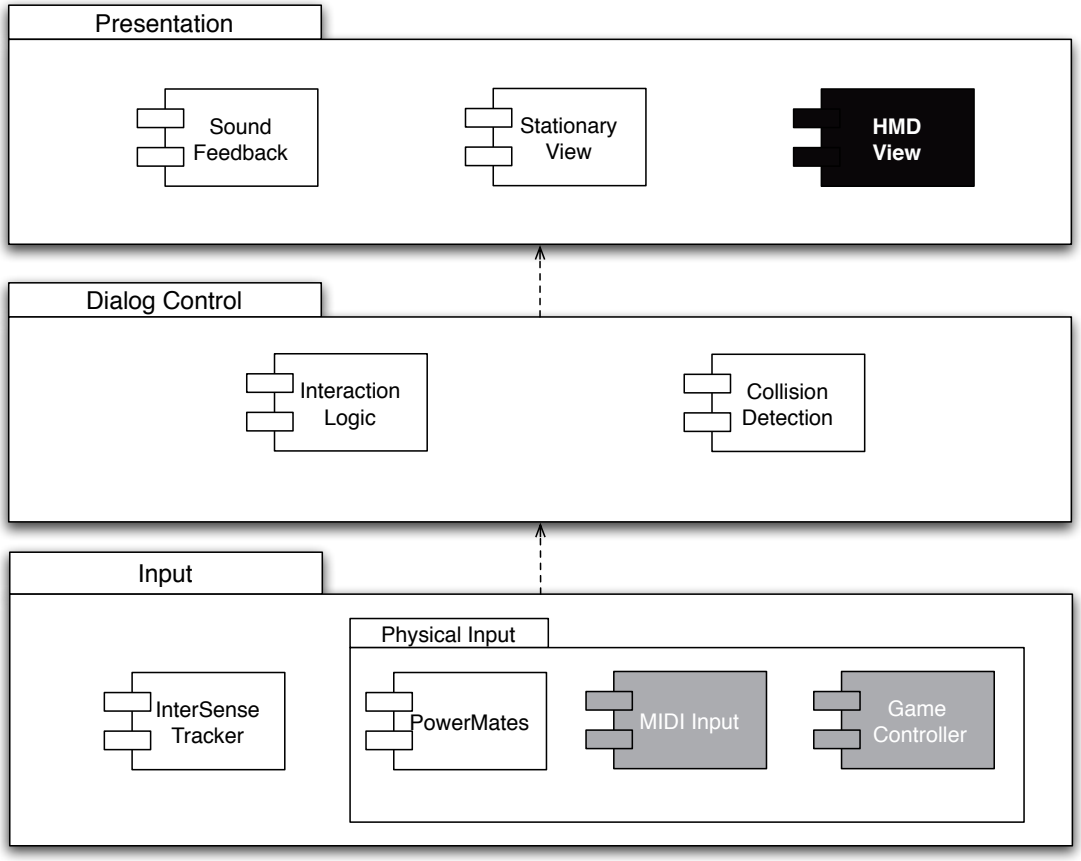


Figure 5.12: Mapping of components to subsystems. White components are implemented in AVANTGUARDE, light grey components in Unit, and dark grey components in DP.

for the input device drivers. The remaining components are based on existing AVANTGUARDE components (with only the PowerMates component implemented from scratch).

The dataflow between the components relies on two different mechanisms. Within each framework, we use the framework's native communication protocol: CORBA (Common Object Request Broker Architecture) events for AVANTGUARDE, and UDP (User Datagram Protocol) for Unit and DP. Across framework boundaries, we also use UDP, which has proven to be a simple, yet viable solution. We run Unit and DP frameworks on Windows XP Professional, and AVANTGUARDE on SuSE Linux Professional 9.1. Our components include:

Stationary Views Our two stationary views reuse AVANTGUARDE's Open Inventor-based Viewer component (Section 4.2.2). We extended this component to send 2D screen-space bounding

Figure 5.13: Input devices used in our prototype: (a) Dance mat. (b) Game controller. (c) Wand with attached InterSense 6DOF tracker. (d) Tracked board with PowerMate sensors (left) and MIDI sensors (right): sliders and bend sensors attached to playing cards.

box information for objects to the collision-detection component, to allow collisions between the wand and objects on the screens to be detected. This information is also sent to the head-mounted display view, enabling it to visualize relationships (through overlaid lines) between objects on the screens and input devices.

Head-mounted Display View Each head-mounted display view uses the DP framework, which can efficiently implement tabular mappings, such as those of Figure 5.11.

Interaction Logic The interaction logic is modeled within AVANTGUARDE's User Interface Controller component (Section 4.2.2).

Collision Detection This AVANTGUARDE component is based on the Euclidean distance of the center points of tracked physical objects and the centers of the 2D screen-space bounding boxes of virtual 3D objects.

InterSense Tracker 3D tracking is implemented using InterSense IS-900 and IS-600 trackers. This component is a straightforward wrapper for the InterSense native library.

Figure 5.14: Virtual mirror-images of tracked objects (wand, board) provide additional feedback.

PowerMates Our tracked board includes several Griffin PowerMate [56] rotary sensors.

MIDI Input Devices The availability of many different MIDI (Musical Instrument Digital Interface) input devices motivated us to support them in our system. We use the Unit framework to encapsulate MIDI functionality into a separate Java-based library. An A/D converter [71] converts analog sensor data to 7-bit MIDI data, making it easy to support bending sensors and sliders.

Game Controllers To accomodate game controllers, we use a platform-specific library for low-level interfaces to peripheral devices through the RAWINPUT functionality in Windows XP. We support any number of Windows-compatible game pads or joysticks, including the Microsoft Sidewinder FreeStyle Pro gamepad, which has two accelerometers for sensing pitch and roll. By using two types of USB adapters that support up to four PlayStation or PlayStation 2 controllers, we exploit the wide range of controllers available for the PlayStation and PlayStation 2 platform, including a generic 1m×1m PlayStation dance pad with 14 buttons and an analog PlayStation controller with buttons and two analog joysticks.

My Contributions

In this joint project, I was responsible for several implementation decisions and interaction designs. This section presents a short summary and discussion of these parts.

Feedback on stationary views In addition to the visual feedback presented to the user in the HMD, visual feedback was also provided on the stationary views. A virtual representation of

(a) Text-labels. (b) Icons.

Figure 5.15: Two ways of visually representing geometric transformations.

tracked input devices was displayed on the stationary views (see Figure 5.14). The virtual representation was updated in real time, to behave like a mirror-image of the real object. When a user selects an input device on the tracked board, a virtual yellow box around the mirror-image confirms the selection (see Figure 5.10(c)). This mirror metaphor [27] proved to be easy to understand for users. Another benefit is that users not wearing a HMD could see which input device was selected.

Interaction design: naming geometric transformations. Three elements in the interaction design a closely related: audio feedback, visual feedback in the HMD and the printed menu, since they all refer to the geometric transformations that a user can perform. In our first prototype, the printed menu contained nine entries: translateX, translateY, translateZ, rotateX, rotateY, rotateZ, scaleX, scaleY and scaleZ. The same labels were displayed in the HMD view (see Figure 5.15(a)). The audio feedback consisted of a voice that read this label for the user.

This representation of the geometric transformations proved to be difficult to understand for users, in line with the discoveries of the Alice project [34]. In our second iteration, I replaced the labels with icons that show a cube with arrows that illustrate the geometric transformation (see Figure 5.15(b)). This solved the visualization problem for the printed menu and the HMD. However, it did not solve our problem for audio feedback, since it is not trivial what should be the aural representation of an icon. I decided to use a generic voice feedback ("operation"), when a geometric transformation was selected. To come up with a better solution seems a very interesting aspect for future work.

Interaction logic When reconfiguring the dataflow network, users must specify a triple: (target object, input device, and operation). There seem to be two fundamental approaches to the selection of each of the triple elements. For both approaches, the system performs the reconfiguration whenever the triple is fully specified. Approach 1 is to let users select an element repeatedly—the last selected element is valid. Approach 2 is to let users select an element just

once—selections are final. In this approach (which I decided to use), an undo functionality for selections is imperative. The main reason, why I think that the second approach is better: when selecting input devices on the tracked board, these objects are very close to each other. The typical gesture performed by most users to pick input devices with the virtual wand was this: they picked precisely the input devices they wanted, however, when they pulled the wand back, they were very close to the other input devices. With approach 1, this would have led to many faulty selections.

Discussion

Our UAR user interface allows the user to reconfigure dataflow networks We support interactive end-user reconfiguration of the mapping between devices, objects, and operations. Using a head-tracked, see-through display, we provide overlaid visual documentation of the system's current configuration and overlaid visual and auditory feedback as the system is reconfigured.

The threshold of this tool is very low, since it is very easy to use. However, the ceiling of the tool is also very low, since our system supports only a fixed, and very limited number of operations. We are exploring how we can extend it to allow users to specify new operations at runtime, such as model deformation. While we anticipate using programming-by-demonstration [92] to address a carefully planned universe of possibilities, supporting arbitrary operations through demonstration and generalization is an open problem. A more pragmatic approach to increase coverage would be to use the Python services in AVANTGUARDE [148], since Python is well suited for rapid development by end-users who can program. Additional enhancements that we plan include support for grouping and selecting multiple devices, operations, and objects, along with the ability to load and save configurations. For example, we would like to make it easy for a user to select a single device and specify that it controls multiple operations (e.g., scaling in x, y, and z) on a group of objects.

As we extend our user interface, we will be designing a formal user study to validate our approach, benefiting from the informal user feedback that we gathered when demonstrating earlier versions of the system. After receiving a brief explanation of how to use the system, these early users found it easy to reconfigure the system themselves, and told us that they appreciated the overlaid visual and audio feedback provided during reconfiguration. Based on user feedback, we replaced textual annotations on the lines (visible in the first segment of the accompanying video) with the iconic representations of operations that we currently use. However, an overview visualization that showed all of the relationships simultaneously, proved too confusing to be useful because of the visual clutter caused by overlapping lines and icons. Therefore, we removed it from the current version of the system.

5.4 Specifying Dialog Control—The User Interface Controller Editor

In this section, we present an approach for interactive prototyping of interaction management for UAR user interfaces. Our approach is based on AVANTGUARDE's User Interface Controller (UIC) component (see Section 4.2.2), which embodies an infrastructural element to specify and

execute interaction management logic. We have developed a GUI frontend (the UIC Editor) that user interface designers can use to quickly and interactively reconfigure the UIC, to try out various user interface concepts without having to go through long (re)programming periods.

The remainder of this section is organized as follows: We start by highlighting the motivation for the UIC Editor in Section 5.4.1. Section 5.4.2 proceeds by explaining how we enabled developers to change the Petri nets during system runtime more easily. Section 5.4.3 illustrates this by an example, and finally Section 5.4.4 concludes by discussing the benefits and limitations of our approach.

5.4.1 Motivation

The first UIC implementation comprised some drawbacks. The standard mechanism to describe Petri nets in JFern is a combination of XML (for the net structure) and Java code (for the guards and transition declarations). That proved to be feasible for the simple Petri nets used in SHEEP, but as soon as the nets became more complex, it became very difficult to maintain the overview of the complete source code. Furthermore, the rigid nature of Petri nets forced us to shut down the UIC every time we wanted to change the structure of the net or even a small detail in the actions or guards. That insight led to the idea of a completely graphical approach, an interactive development environment for Petri nets and thus a visual programming environment for UAR user interfaces. Similar approaches have been done for general programming and for WIMP user interfaces [75, 31].

5.4.2 Implementation

We have implemented a visual programming environment that allows us to cover three main tasks: building and modifying the Petri net structure and hence controlling the dataflow and behavior of the complete user interface. Figure 5.16 shows the tool in net modification mode. The next task is to modify the transition's *actions* and the *guards* on arcs. The last main task is to control and modify the DWARF *needs* and *abilities* dynamically, which allows us at runtime to connect and disconnect devices from the *Media Analysis* and *Media Design* layers. That allows us further to constrain those connections by defining attributes and predicates on single connections. The remainder of this section presents details for how the UIC Editor supports these three tasks.

Net Structure Modification

Modifying the net structure means to define the dataflow through the user interface. On one hand, that means to define how many inputs and what sort of inputs are needed to execute one task (e.g., a gesture and a speech command), and on the other hand, what sort of output is generated. With the net structure, we also define how different tasks are related to each other.

To keep the complexity of the nets as small as possible, we use the concept of *sub nets* [78] which are small interaction entities that model one isolated interaction (e.g., insertion of an

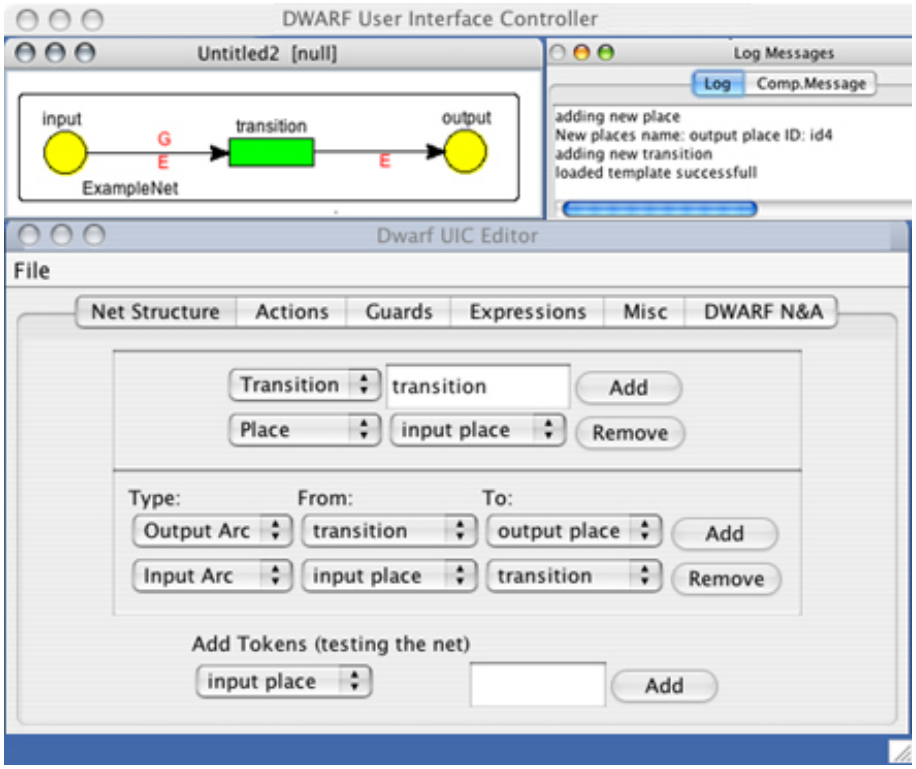


Figure 5.16: The DWARF UIC showing a very simple Petri net and the net structure modification tab.

object in a 3D scene). Those sub nets can be inserted into the overall nets without showing all included places, transitions and arcs (see Figure 5.17). Each net (including sub nets) can contain an unlimited number of sub nets. Our implementation allows us to add, remove and edit all net atoms (places, transitions, arcs, sub nets) at runtime.

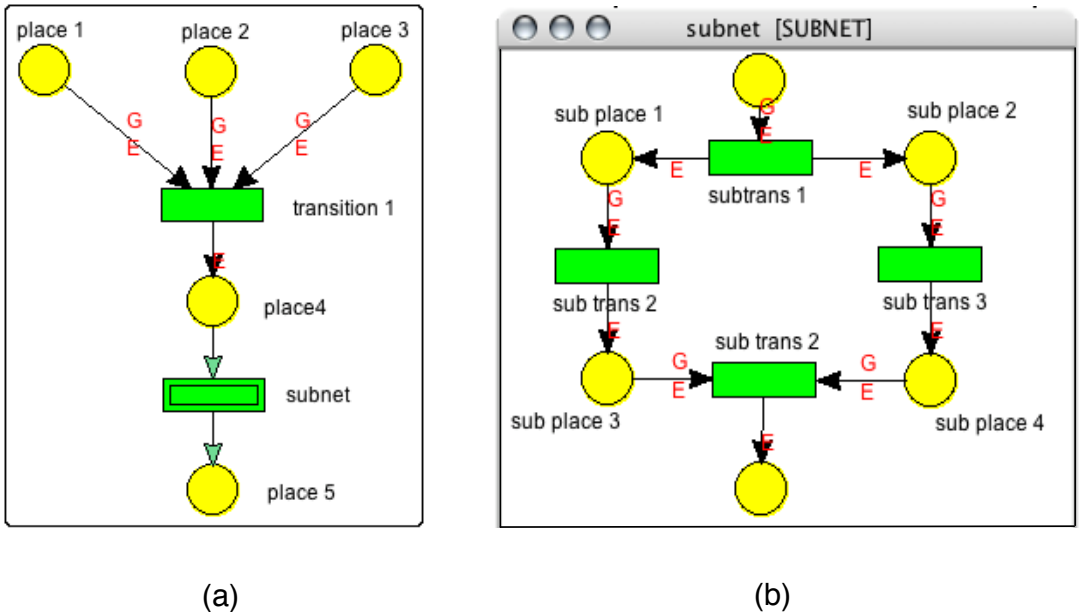


Figure 5.17: (a) Petri net containing a sub net. (b) The sub net in a separate pop up window executing two transitions separately.

Dynamic Code Modification

In the previous section, we stated that we can change how the data flows through the user interface control structure. To really change the behavior of the system, we need to modify the data manipulation that is done within the control structure. In our case, this means to exchange the code of the *guards* and *actions*. The *guards* check whether special constraints on the input are fulfilled (e.g., are there three tokens of type 'speech command'). The *actions* fire when all *guards* on incoming arcs of the encapsulating transition evaluate to true. An action executes arbitrary Java code and has access to the tokens that have been put into input places. The code executed actually changes the state of attached components from the *Media Design* layer. In most cases, the code contained in actions boils down to a few lines. Essentially actions extract data from input tokens, which are DWARF structured events, and compose new DWARF structured events (commands), which are sent to the *Media Design* components, and, finally, the same structured event is placed into the connected output places.

The standard API does not support dynamic code modification, which is essential for our approach. We used a third party extension library for dynamic compilation [83]. The Graham-Kirby compiler provides an interface to access the Java compiler dynamically from a running program. This library enables us to exchange the code of *guards* and *actions* at runtime and in consequence enables us to modify the user interface behavior dynamically.

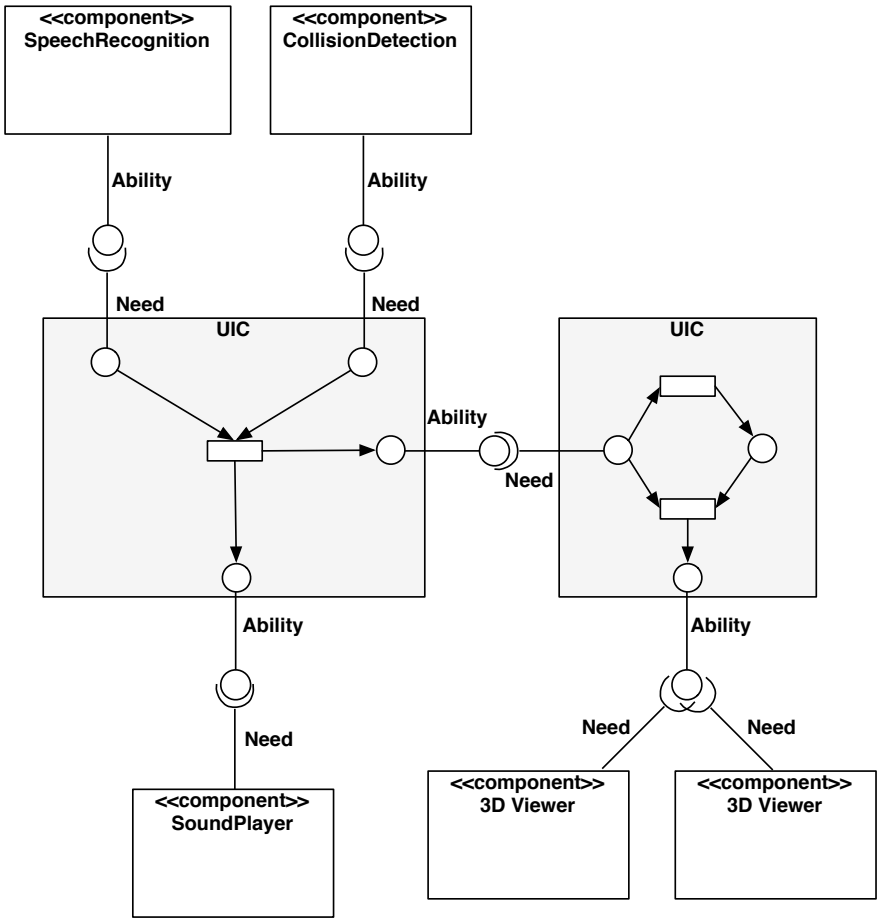


Figure 5.18: Schematic DWARF user interface incorporating different connection possibilities.

Connectivity Management

So far, we have described how we utilize Petri nets to model multi-modal interactions. Now we show how we control and model the communication with attached components of the *Media Analysis* and *Media Design* layers. Connections between all components are based on DWARF *needs* and *abilities* and communication channels are set up at runtime. Developers can define attributes on *abilities* and predicates on *needs* to specialize the connection criteria [102].

Within our tool, the developer can add new *needs* to attach input components to input places of the Petri net. Furthermore, the developer can define predicates on that *need* to select from different components that have *abilities* of the same type. The connections will be set up whenever a matching pair of *needs* and *abilities* is present in the network environment. Whenever attributes or predicates change, the regarding connections are disconnected and new

communication partners are connected if available.

The DWARF approach allows developers (and even users) to detach, attach or exchange input components at runtime and thus experiment with different modalities. One could also think of replacing components that are currently not available by others that simulate the behavior of the original component. This method showed to simplify the testing process in systems that incorporate a variety of rather experimental devices.

Abilities and output places can be modified accordingly. This allows us to flexibly adapt different output components and control which components receive which commands. So we can use different modalities to present content to the user or show different content on devices belonging to different users or user groups (e.g., private informations vs. publicly available informations). Alternatively, one can setup one-to-many connections so that many output components are connected with one output place (e.g., controlling several views simultaneously). Figure 5.18 shows a sample connectivity structure illustrating different possibilities to connect user interface components.

Another aspect of our architecture allows developers to keep full control over the granularity of their Petri nets. Since any arc in a Petri net can be replaced by a DWARF connection, a developer can model everything in one self-contained component or on the other end have several interwoven components each modeling just one single interaction. Such atomic Petri nets can then be reused in different applications.

This is also interesting in aspects of ubiquitous computing where several more or less independent UICs can connect to each other at runtime and thus form a richer, more powerful control structure enabling user interface aspects not available to the single sub applications.

5.4.3 Example

To explain the benefits of the UIC Editor, we discuss an example from the CAR system (to be explained in Section 5.5) that shows the benefits gained from using the UIC Editor. The example we present in this section also shows the actual combination of tools. The UIC Editor was combined with the tool for visualizing the user's eye gaze from Section 5.2.2, since the decisions for adjusting the interactions with the UIC Editor are heavily based on the user's eye gaze.

CAR incorporates concepts of mixed and augmented reality, adaptive user interfaces, multimodal user interfaces and attentive user interfaces (AUI). AUIs monitor the user's visual attention and coordinate their behavior accordingly. For user interfaces in automotive environments, it is very important to consume as little of the user's attention as possible since the user has to concentrate on the driving task. In the CAR project, we used several sensor technologies to track the user's head orientation and the gaze direction to measure when the user looked at the car's central information display (CID). We used then several techniques to attract the user's attention (visual, spatial audio and combinations of those) and adapted the information displayed whenever the user looked at the currently active part of the user interface. We will use that setup to conduct user studies in the future to increase usability and security for automotive user interfaces.

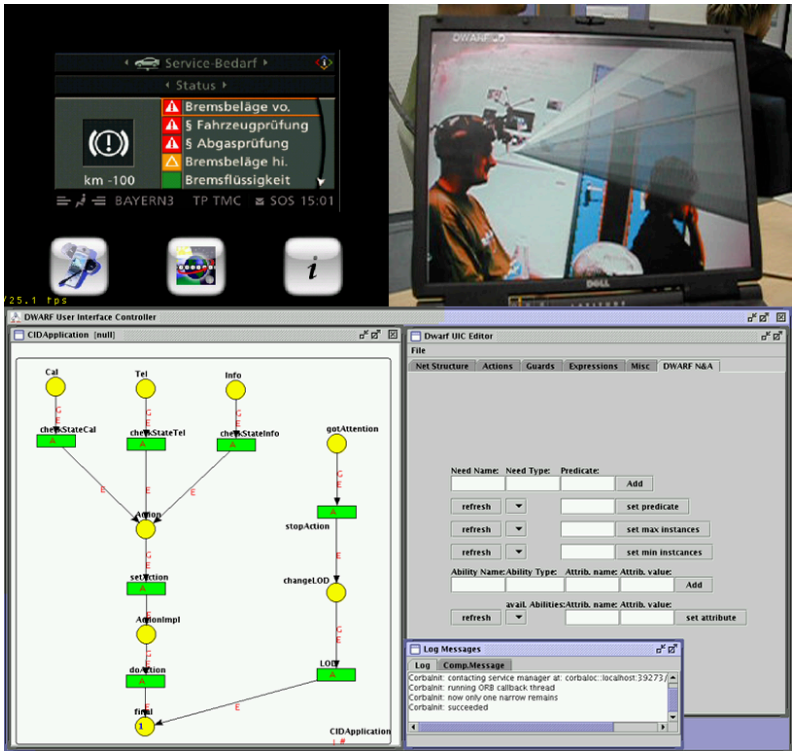


Figure 5.19: (a) The CID. (b) The AR visualization of a user's gaze. (c,d) The UIC we used to modify the AUI at runtime.

We used the UIC to control the described AUI, specifically to adapt the various sensors and filters and connect them to the graphical representation. We also utilized it to change the behavior of the user interface at runtime (e.g., duration and kind of attempts to get the user's attention) and the content shown when the attention has been attracted. Another interesting aspect of CAR is the large number of participating displays; each of them showing different dynamic content (e.g., driver's CID, head up displays, wall-sized displays for observers and 3D debugging views for developers). All of them are controlled by four Petri nets with small to average complexity (less than ten places).

5.4.4 Discussion

Our interactive runtime development environment has reached a level where programmers that have a significant level of experience in the usage of computers (e.g., masters students in computer science) can use it to quickly assemble and tune UAR user interfaces after about one

week. It needs more work to make it useful for less experienced programmers or even the end user in the long term.

The current 2D implementation needs some improvement to increase usability (e.g., convenience methods and help system). We are working on a set of reusable interaction components that can be dropped into Petri nets from a repository to reduce redundant work. Currently there is still some programming needed to define the actions and guards of the Petri nets. In the future this could be reduced or, ideally, completely replaced by a programming-by-example approach [92]. Another option would be to replace the underlying Petri net framework with a more sophisticated one (e.g., [79]).

5.5 Creating Context-Aware Visualizations—The CAR Environment

With the advent of mobile and even wearable computers, the demand for novel human-computer interaction concepts that are not based on the desktop metaphor is becoming ever more urgent. Mobile computers are used not only while users are sitting at their desk; but also while they are roaming a large indoor or outdoor environment, such as an industrial plant or a city. Tasks typically involve overall navigation within the real world, as well as specific, work-related tasks, such as the assembly, inspection, maintainance, or repair of a machine or system.

Within these scenarios, computers and interaction devices may be worn directly by the user or they may be ubiquitously embedded within a mobile environment that is travelling with the user, such as a car, a crane or a push-cart. In this section, we discuss how to provide a development environment to support the design of multi-media and augmented reality-based user interfaces for car drivers.

This section is organized as follows: first (Section 5.5.1), we start by clarifying the problem that we are attacking with this tool. Then (Section 5.5.2), we give an overview of our approach. Next, the generic components that make up our approach are presented (Section 5.5.3). Details about the implementation are explained in Section 5.5.4. In Section 5.5.5, we conclude this chapter by pointing out the benefits and limitations of our approach.

5.5.1 Motivation

The problem statement consists of two parts. First, we present the issues in the target domain of novel human-computer interaction concepts for car drivers. Second, we discuss the requirements for a development environment addressing theses issues.

Issues in Designing Human-Computer Interaction Concepts for Car Drivers

To support mobile users—and, in particular, car drivers—while they are moving in the real world, computers are expected to supply users with information pertinent to the current task: navigational advice (local and global directions), information on the current driving conditions

(speed, distance to other cars, traffic signs, conditions of the road and the weather) and convenience information such as access to the radio, CD player and the mobile phone. But where and in which way should such a wealth of information be presented to car drivers? In the following, we will present a list of issues that a user interface design team has to resolve when designing interfaces for cars.

Information presentation across several displays. Driving a car requires users to keep their eyes as much as possible on real objects rather than on monitors in their vicinity. To this end, there are efforts to move two-dimensional information such as warning icons or the speedometer from consoles and other displays into the drivers' real-world view (e.g., through a heads-up display (HUD)).

This is much in line with reseach on augmented reality technology—which suggests showing virtual information fully embedded within the real world by inserting it into a person's field of view. To augment HUDs with navigational information, the system needs to register both the car with respect to the surrounding environment (e.g., via GPS) and the driver's head relative to the car and in particular relative to the windshield.

Yet, this approach is suitable for only a small subset of the information, and only under certain circumstances. There is the danger of overwhelming users with virtual information to the extent that they become unable to still perceive the real world—let alone focus on it. Good user interfaces require guidelines for when to present information in the HUD and when to migrate it to a secondary display and vice versa.

Information presentation with respect to different reference frames. It is unclear where information should be presented in a HUD and which field of view of a driver should be covered. Due to current technical and financial limitations, information is currently shown in a fixed, small location on the windshield. But there is the danger that virtual information can occlude the driver's view of critical real objects. Furthermore, the information cannot be seen by the driver during wide head motions (e.g., while backing up and while parking a car). All technical constraints barred, drivers should be surrounded by a transparent hull of virtual information being presented in the appropriate way wherever they turn their head, while also acknowledging the special physical properties of mirrors and other viewing aids. To what extent should this vision be realized? What kinds of cognitive constraints limit drivers in understanding and using an informational hemisphere?

Several reference frames have to be taken into account when presenting information. Information pertaining to real 3D objects is to be shown with respect to the global, outdoor world coordinate system (world-based reference frame). But what about 2D objects, such as the speedometer, that do not have a unique relationship to a particular real object? Current technology places such information in a fixed position on the windshield (screen-based reference frame). As discussed above, this approach does not support drivers while they are looking to the side or to the back. Another approach presents 2D information stabilized with respect to the user's head. The information then follows head motions such that it always appears in a

specific place in the user's field of view (head-fixed reference frame). Yet, this also means that drivers can never actively turn towards an information shown in their peripheral view to focus on it.

Spatially and temporally consistent non-overlapping information placement. Important real objects should not be covered by virtual information and icons or informative texts should not hide each other. Thus, all potentially presentable items of information should negotiate space constraints with one another as well as with an agent representing the viewing cones towards critical real objects. Yet, self-adjusting information placement cannot operate without a certain amount of temporal consistency. Drivers need to be able to quickly glance at a piece of information at an expected position without having to search their entire field of view. To this end, temporal consistency constraints have to be considered in addition to spatial constraints [19].

These issues need to be weighed and traded off by a user interface design team, identifying different classes of information that should be treated according to different strategies and depending on the nature and size of the information, as well as on the current real-world situation and on the recent interaction history.

Selection of suitable input modalities for user control. In cars, mice and keyboards are not available. Speech recognition, as well as the analysis of head and hand gestures is currently explored with respect to well defined sets of gestures. User head and eye motion is also used in virtual and augmented reality systems to adapt the presentation to the current user position or pose. Head and eye motion can be further used to monitor a driver's current focus of attention. In cars and other mobile environments travelling with the user, further sensors such as the current speed, friction of the wheels, acceleration and use of the gas pedal, orientation of the steering wheel, and distance measurements with respect to other objects can be taken into account as well.

The optimal fusion of this wealth of input parameters into a suitable, situation-adaptive concept for providing currently relevant information at a suitable place is still a subject of research.

Issues in Providing a Development Environment for a User Interface Design Team

The development of suitable human-computer interaction concepts for mobile users is, in its very nature, a multi-disciplinary task involving experts from fields such as computer science, human factors and psychology. To ensure optimal working conditions for such a collaborative team, a number of new tools and facilities have to be developed [86, 103].

How to provide mockups for technical equipment that is still under development? Due to the immersive and adaptive nature of location-dependent mobile human-computer interaction, it is important for designers, developers and evaluators to quickly obtain a personal, live impression of the concepts they are proposing. Yet, in many cases, the input and output devices that are

under consideration haven't even been built yet. Rather, only prototypical, technically limited solutions are available.

It is one of the expected results of a user interface design team to provide guidelines specifying which kinds of devices are likely to be successful in supporting mobile users, and which concepts are likely to fail and thus should not be built. To this end, the design team needs a virtual prototyping environment within which many of the envisioned devices are provided by digital or simple physical mockups. This is the classical task of virtual or mixed reality-based simulators.

How to provide flexible system configuration facilities for non-programming experts? Current simulators are typically large, complex systems that are tuned for high performance rather than flexibility and easy reconfigurability. The inclusion of new input or output devices, as well as fused, interdependent use of several devices generally can only be achieved by system experts. This is counterproductive during a brainstorming process, when multidisciplinary experts propose and describe novel interaction concepts to one another.

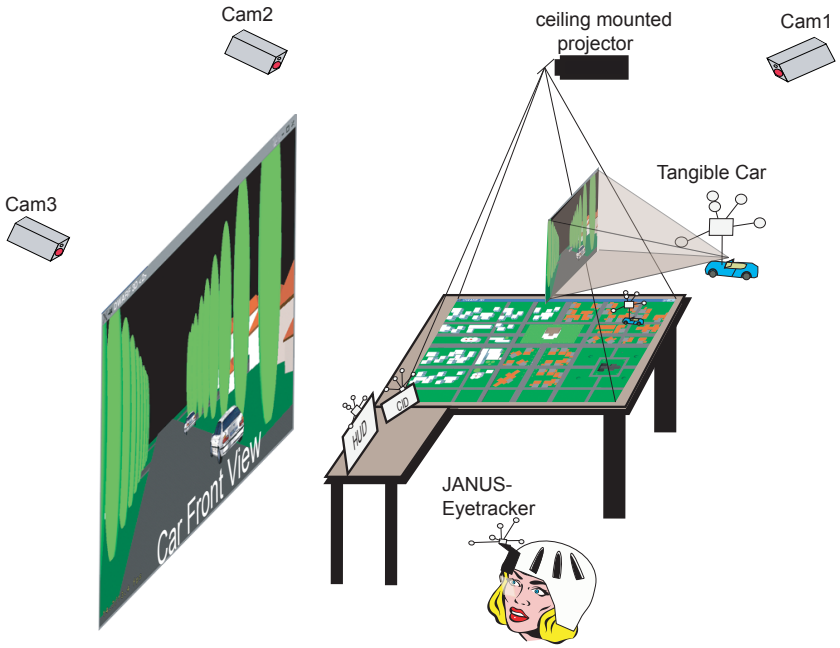
A number of projects exist that aim at providing user interfaces tools that are easy to use by non-expert users. However, these tools have a relatively specialized focus; for example, 3D animations with Alice [34] or Augmented Reality on a single display with DART [96].

In analogy to the "sketch-on-a-napkin" metaphor, design teams are in need of a rapid prototyping environment (RPE) that allows them to quickly visualize and act out user interaction scenarios involving new concepts of interpreting and fusing sensor data and generating interrelated system responses across a number of information presentation devices. Early instantiation and evaluation of such sketches allows design teams to maintain a creative brainstorming atmosphere while exploring their ideas in a mixed reality environment.

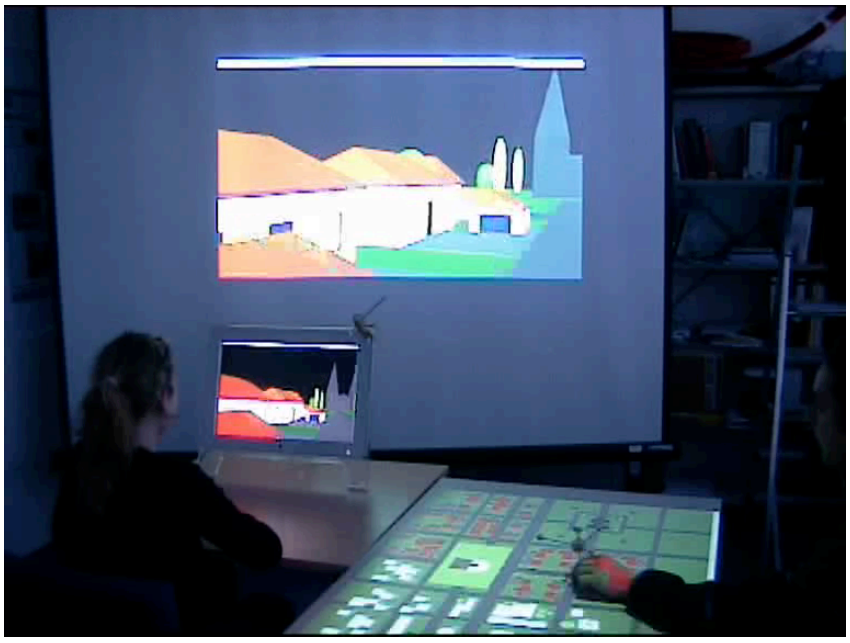
5.5.2 Approach

In this section, we present an approach towards building a flexible RPE that supports user interface design teams in developing, discussing and evaluating novel, spontaneously evolving concepts of human-computer interaction in cars. To date, we have built a first prototypical RPE within which we have been able to show a number of user interface ideas and trade-offs to a design team of human factors researchers, car designers, linguists and engineers in a joint industry-sponsored project. Subsequent sections report on the user interface design and system engineering concepts underlying the development of the RPE.

In order to provide the required flexibility in bringing in new devices and in associating arbitrary functional dependencies between sensor data and reactive information displays, the structural dependencies between such components cannot be kept internal to the system itself. Otherwise, major programming effort is required to reorganize such dependencies. Modifying the interactive behavior of the system via menu-based access to configuration files is not that suitable either, since non-experts quickly lose the ability to understand all inter-related customizations when the customization files become large and numerous. Rather, we propose to provide mixed reality and tangible interaction metaphors [74] to let the design team act



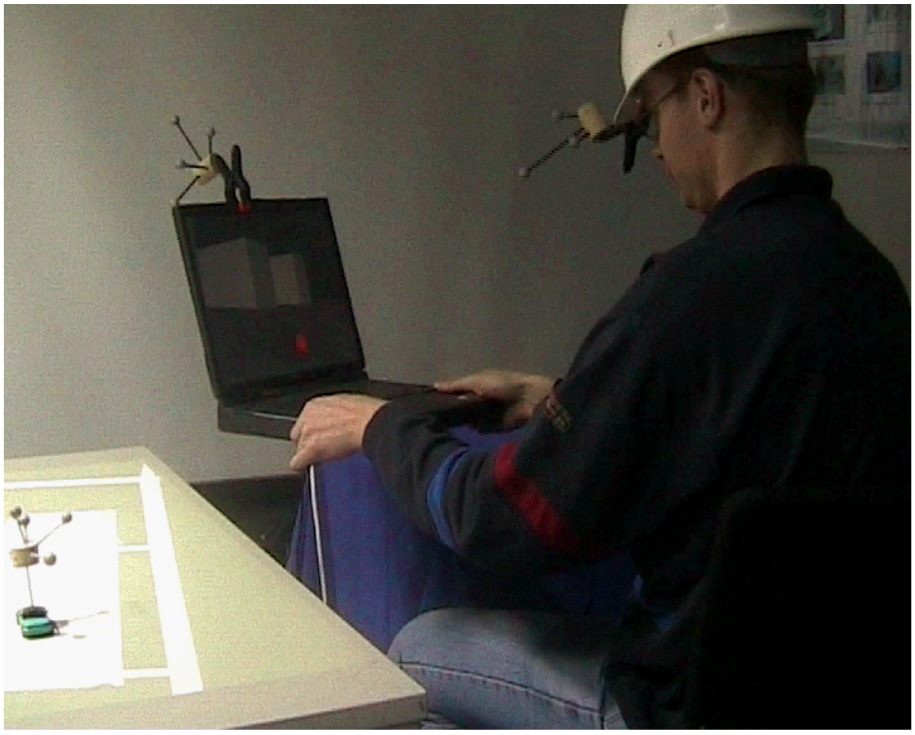
(a) Conceptual drawing.



(b) Photo of the actual setup (rotated 90 degrees relative to the drawing).

Figure 5.20: Physical setup for the RPE.

(a) Inclusion of a second display.



(b) View-dependent information presentation.

Figure 5.21: Further details for the RPE setup.

out desired sensor-display combinations in visionary situations, much like children act out a scenario in a play room.

Physical Setup

We have set up a tangible simulator for studying car navigation metaphors in traffic scenes. The basic setup is shown in Figure 5.20. It consists of two separate areas: a simulation control area (big table with a tracked toy car) and a simulation experience area (person sitting at the small table with a movable computer monitor in the front and a stationary large projection screen in the back). The room is equipped with an optical tracking system [1] consisting of three cameras in the corners of the lab. The cameras are able to track several mobile objects, such as a toy car, the computer monitor, and the user (impersonating a car driver). Each tracked object is equipped with a marker consisting of a rigid, three-dimensional arrangment of reflective spheres. Information is presented on several devices and surfaces in the room: A projector mounted on the ceiling projects a bird's eye view of a city down onto the large, stationary table on the right. Another projector presents the current, egocentric view of a virtual car driver sitting in the toy car on the large screen at the front wall. A third, location-dependent visualization of the driving scenario is shown on the mobile computer monitor. The spatial relationships between all tracked and stationary objects are maintained in order to show a set of consistent views for people in different roles. A similar setup has already been used in the Sheep system [103].

The two conceptually separate interaction areas cover different functionalities of the system:

Simulation control area In the simulation control area on the right (see Figure 5.20b), one (or potentially more) toy cars, equipped with trackable targets, are placed on the table on top of the city map. The members of the design team can simulate traffic situations by moving cars on the table, thereby being able to control the simulator via a tangible object.

Simulation experience area The simulation experience area to the left represents the cockpit of a car and the driver. The picture projected on the large screen in the front displays the view a driver would have when sitting in the toy car. The movable monitor in front of the driver shows partial views of a conceptualized information hemisphere of information surrounding the driver. When the driver rotates the monitor to the right or left, she is able to see views through the side and rear windows of the car (Figure 5.21b). This is our approach towards conceptually providing a mockup of a completely surrounding HUD. The user interface design team can thus explore the potential use of adding HUD functionality to other windows (e.g., for a parking scenario, without having to build it). Further movable monitors can be added at run-time to the setup, if more than one view is needed—e.g., in order to represent a central information display (Figure 5.21a) in the cockpit or a rear view mirror. Movable monitors can be augmented with additional driving information according to the evisioned information presentation concepts of the user interface design team. Depending on the reference frame, the augmented information can move with the monitor, the user's head or the real world.

Designers can jointly and interactively discuss the optimal position of augmented information in a driver's environment by moving the monitor in any direction they want. They can exchange monitors in order to test a different augmentation concept, since such concepts can be associated with different display devices. Thus, designers can use and exchange tangible objects in order to simulate different information presentation options that they may have in mind. This allows them to explore and compare different concepts.

By tracking the driver's head in addition to tangible objects, users can control the system via head motion (Figure 5.21b), as well as via other input modalities, such as speech.

5.5.3 Generic Components of CAR

This section presents several components of our RPE that can be used to provide early immersive impressions of novel user interface concepts while their trade-offs are discussed by the user interface design team.

Basic Input and Display Functionality

Where should information be displayed? How large does the HUD have to be? Will we need a full information hemisphere?

To answer such questions, the basic display functionality of the RPE offers several views of a virtual city environment, as presented in the previous section. The system provides both exocentric and egocentric views from a driver's and a bird's eye perspective in the two areas of the simulator. Whereas the principal viewing pose of the driver is determined by the toy car in the control environment, the driver can adapt the viewing parameters further via the movable screen.

HUD Augmentation with a World in Miniature Model

Current car navigation systems typically present a map of the environment in a CID. Depending on the complexity of the local area, the map is automatically zoomed or scaled back to provide either important local details or a wide overview. Drivers can also interactively control the zooming behavior.

One of the issues to discuss in the design team is the question, of whether and how to present a navigation map on a HUD. Where should it be placed? How large should it be? What level of detail should it provide? Should it be a two-dimensional map or a tilted view onto a three-dimensional environmental model, called a World in Miniature model (WIM)? A WIM was first used for mobile augmented reality in [70]. However, if so, which viewing angle should be selected? Will the angle, as well as the position of the WIM and the size and zoom factor adapt to some kind of sensor parameter, such as the current position of the car while approaching a critical traffic area in a town?

The RPE provides a component for augmenting a scene with the view of a WIM, parameterized to account for different positions, sizes and 3D viewing angles onto the world model.

Flexible Three-Dimensional Placement of Labels and other Information

When icons and texts are added to the HUD, their positioning can depend on a number of design decisions. Each can be placed in a fixed location or they can adaptively negotiate for space according to a number of rules that the user interface design team needs to specify: the choice of a reference coordinate system, and rules what to do when labels overlap critical real objects or when they overlap one another, as well as rules specifying temporal consistency.

Further design issues can arise when labels or icons are not positioned flat on the screen but rather oriented such that they align with surfaces of real objects. This issue can be relevant to the positioning of traffic signs or sign posts providing navigational hints.

The RPE provides a component with an adapted version of Bell's label placement algorithm [19]. Depending on the provided information, it presents street labels, or other information that is attached to real objects. It provides a starting point towards providing flexible arrangements of icons with differing semantics.

Adjusting the Context-Aware Animations of the WIM

We adapted the basic idea of context handling from [20]. Figure 5.22 shows how the WIM behaves dynamically according to the distance to the parking lot. There are four parameters of the WIM that are adjusted dynamically. The centering and zoom of the WIM behave similarly: when the driver comes near the parking spot (3 meters), the WIM is zoomed and centered. The scaling is a more complex functions that was chosen arbitrarily to illustrate our idea. Note that the zoom operation is like scaling, but the screen space consumed when zooming remains constant, since clipping is applied. Finally the WIM is constantly tilted along the horizontal axis by 30 degrees. How is that behavior implemented in our collaboration platform? The necessary steps are:

1. Non-programmers can change initial values in filters using tangible interactions. E.g. the tilt of the WIM which is constant can be adjusted by tilting a tracked plate (Figure 5.23b). At the same time, the initial value for zoom can be set by moving the tracked plate closer or further away (Figure 5.23c).
2. Non-programmers can change the dynamic behaviors using a sketching tool. See Figure 5.24 noting that the x-axis is mirrored compared to figure 5.22.

5.5.4 Implementation

The RPE is a flexible software system building upon the ad-hoc connectivity and the peer-to-peer-based framework described in Chapter 4. In addition to the components described in the above source, we have created some additional software components:

Janus eyetracker A driver feeding the eyetracking data into our framework has been developed. As hardware device we have used the Janus Eyetracker, which is a head-mounted eyetracker

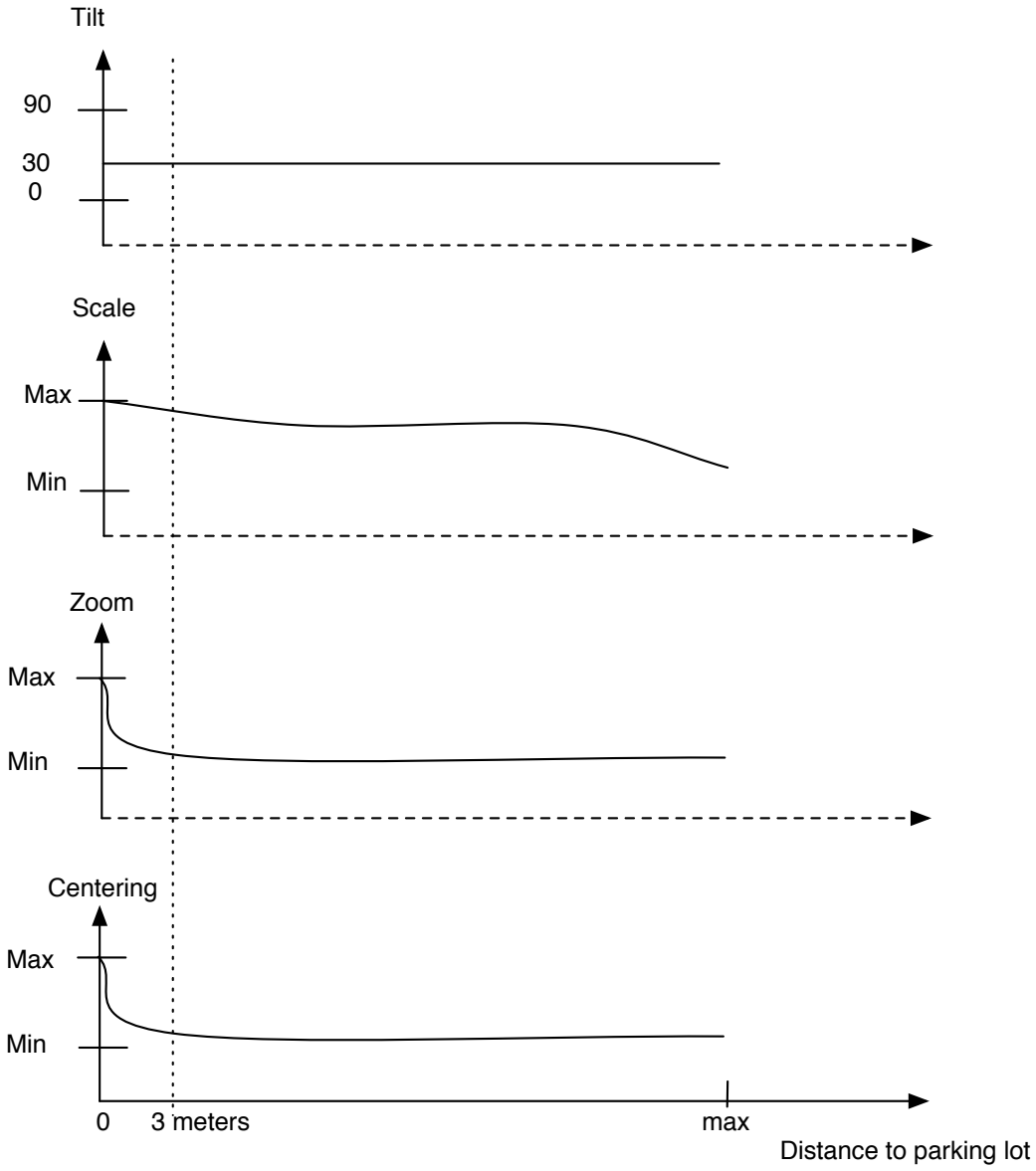
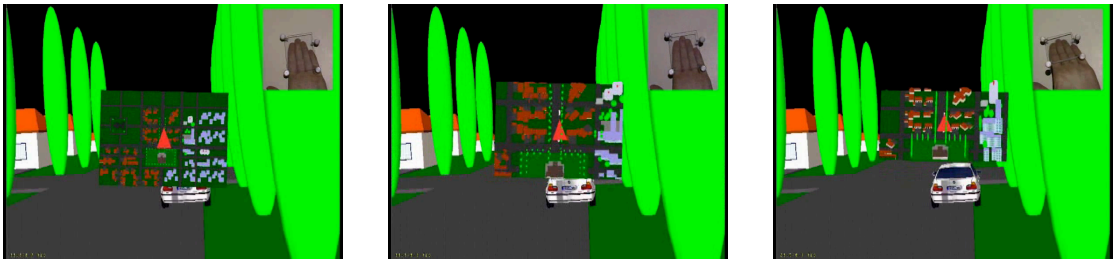
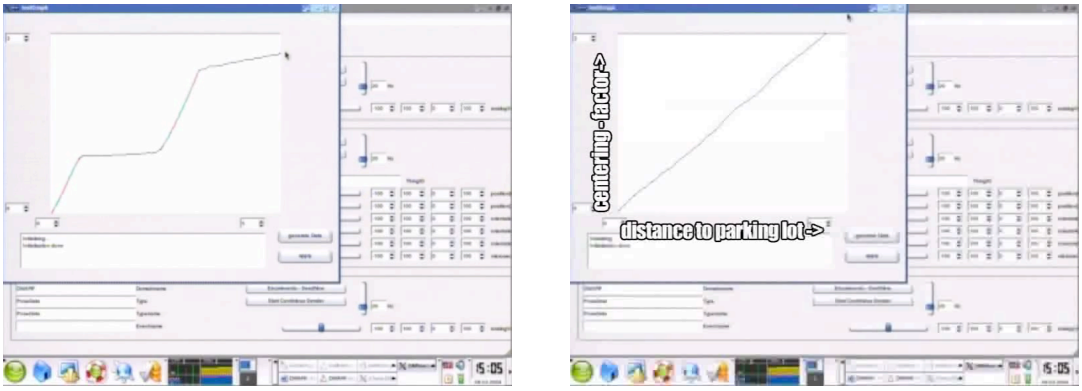


Figure 5.22: Parking scenario: dynamic behavior of the WIM.



(a) Initial WIM. (b) Zoomed WIM. (c) Tilted WIM.

Figure 5.23: Tangible Interaction for adjustment of the WIM.



(a) Stair-case function. (b) Linear function.

Figure 5.24: Sketching the context-visualization function.

that was developed by one of our project partners and is very similar to commercially available eyetracking systems (e.g., the *ASL* 501).

Automatic layout The Automatic Layout is a wrapped version of the space management library [19] from Blaine Bell at Columbia University. It can be downloaded from his webpage [33]. It enables the developer to impose geometric constraints to the virtual objects in the 3D Viewer. As a result, it is possible to display dynamic 3D scenes with no overlaps of virtual objects in realtime.

5.5.5 Discussion

The RPE was developed in the context of an industry-sponsored multi-disciplinary project to investigate issues pertaining to the design of user interfaces for cars. We have used the RPE on several occasions with our project partners. Pictures from one of these live demonstrations are given in Figures 5.25 and 5.26. The results of these joint explorative session are very encouraging. The tangible nature of control objects made it easy for non-programming experts to enter discussions focussing on the design issues.

The principles embodied by the RPE can not only be used for user interfaces in cars. Simulation of spatial context by tangible interactions and modification of context-aware animations by sketching are techniques applicable to any kind of mobile user interface. Additionally, realtime visualization of the attention of users with augmented cones is a novel visualization technique that can be used for any kind of user interface. However, we think that it shows its strength especially in multi-display user interfaces and in multi-user environments.

Limitations

The RPE was intentionally set up to fill the gap between sketching ideas on a napkin and generating complex virtual simulations in a fully functional driving simulator, or generating even more realistic new simulations in a real test car. We made intentional compromises with respect to the achievable realism. In many design decisions, we sacrificed greater realism for faster feedback and rapid modificability.

A conceptual limitation of the setup is reached when interactive devices such as a real gas pedal are included into the design considerations: Since the RPE provides a strict separation between the control area (represented by the tangible car) and the experience area, the setup faces inconsistencies if the driver aims at changing the car position via input devices other than the tangible car: the virtual car position in the traffic simulator can be changed by the gas pedal. Yet, this change cannot be transformed automatically into an appropriate motion of the toy car on the table. This chasm between real and virtual worlds cannot be resolved without involving automatic tele-manipulators.

Another practical limitation of the RPE is that the animation parameters that can be controlled via tangible interactions are limited. If a designer comes up with a new parameter to change during a collaborative exploration session, the coupling of input devices to parameters

Figure 5.25: Interdisciplinary brainstorming with our RPE.

has to be programmed first. Although this process does not take long (see the rapid prototyping facilities for programmers described in Chapter 4), it would be more convenient to be able to achieve this with no programming effort.

Future Work

To ease the configuration tasks of coupling animation parameters to input devices, a simple prototype has been developed [147, 150]. However this system changes only static properties of 3D objects with tangible interactions. Future work will be to integrate these techniques into our RPE.

Formal user studies have not yet been conducted. However the informal feedback we have obtained from our project partners was encouraging and makes us believe that our novel authoring concepts are going to be well perceived in formal user studies.

Another area for future work is that we are currently exploring options to integrating the RPE deeper with a fully functional driving simulator. Currently, we are doing this integration,

Figure 5.26: Discussion of the WIM and attentive icon concepts.

gaining additional realism by incorporating a real car interior and a software simulation for the physics of driving.

Another interesting area for future work is to transfer the prototypical user interfaces designed in the RPE into a real car. The industry-sponsored project within which the RPE was developed aims to achieve that by mid 2006, thus accomplishing our final goal of taking Augmented Reality to the real world.

5.6 Reflections

Within this chapter, we have presented a variety of tools. In this concluding section, we first give a summary of the limitations and benefits of each tool. Then, we present insights that we have gained regarding our overall vision—the combination of tools that employ different user interface paradigms. Finally, we point out future work for realizing our overall vision.

The threshold and ceiling for each individual tool are summarized in Figure 5.27. An explanation for the entries in the tables is provided below:

T1. This WIMP tool can collect and evaluate usability data during system runtime [87]. It is easy to use, since prefabricated scripts can be controlled by a WIMP user interface to evaluate usability data. However, to modify the scripts is too difficult to be handled by complete novices, since this is done in a custom scripting language. However, for computer scientists, this language is easy to understand.
Since the scripting language for this tool is Turing complete, every possible way of evaluating collected data can be implemented.

T2. A tool using an augmented reality visualization [114] that makes it possible to clearly see the visual attention of a user, with a combination of head- and eyetracking, has been presented. Its threshold is extremely low, since all that has to be done to use it, is to start

Tool	Section	Task	User interface paradigm	Threshold	Ceiling
T1	5.2.1	Monitoring the user	WIMP	medium	high
T2	5.2.2	Monitoring the user	AR	low	high
T3	5.3.1	Adjusting dataflow networks	WIMP	high	high
T4	5.3.2	Adjusting dataflow networks	AR	low	low
T5	5.4	Specifying dialog control	WIMP	high	high
T6	5.5	Creating context-aware animations	WIMP, TUI, AR	low	medium

Figure 5.27: Summary of tools presented in this chapter (High threshold means: difficult to use; High ceiling means: complex results can be achieved).

our visualization. The ceiling is high, since we can easily visualize the visual attention of users.

T3. The graphical editor DIVE makes it possible to adjust dataflow networks. To use DIVE, the developer has to have a deep understanding of DWARF and distributed systems—that is why we classify this tool as having a high threshold. On the other hand, since DIVE allows to change *any* dataflows within a DWARF system, its ceiling is high.

T4. Our immersive visual programming environment [147, 150] addresses the same task as DIVE. Since it does not require any knowledge of computer science, its threshold is low. However, compared to DIVE, the number of things that can be changed with it is rather limited. This explains our classification of it as having a low ceiling.

T5. The User Interface Controller Editor [64] (Section 5.4) makes it possible to specify dialog control. Similar to DIVE, it requires sufficient knowledge of computer science (e.g., distributed systems, DWARF Petri nets and Java programming) to be useful—so the threshold is high. The expressiveness of the created Petri nets is Turing complete, since the critical portions of the Petri net are programmed in Java, which is a Turing complete language.

T6. A set of tools to experiment with context-aware mobile augmented reality user interfaces has been presented. It is a an example of a combination of tools following different user interface paradigms. The threshold for these tools is low, since very little knowledge is required to use them. The ceiling is medium, because there are certain kinds of context-aware animations (e.g., when they are dependent on several context parameters) that are not possible to implement with it.

Regarding the combinability of these tools, we have gained some insights:

T1 and T2. These two tools complement each other nicely. While T1 is geared towards exploring longer periods of usage, T2 is meant to be used for short periods,

T2 and T5. The original publication of T2 [114] describes the combination of T2 and T5. The rationale for this combination is that when experimenting with attentive user interfaces, the main area of concern is to try out different ways of attracting the user's attention. Since which user interface elements will be displayed is part of the dialog control, T5 comes in handy.

T3 and T4. T3 and T4 complement each other very well. T3 can be used to perform much more complex operations, on the other hand the immersive tool provides more intuitive interactions. The tools also differ in the level of abstraction. While T3 addresses the low-level aspects of a system by manipulating entities that are only on a component level, the immersive tool interacts with real-world objects and only a few abstract operations. This shields users from complexity. An interesting point is that when we were developing T4, one of the most useful tools for debugging it was T3, since it could provide a ground truth for the results of the interactions that T3 should achieve.

T1, T2, T3, T4, T6. When we were developing and using T6, we have actually combined it with T1, T2, T3 and T4. This approach is probably the most complex development environment for mobile augmented reality user interfaces to date. The results were encouraging. In the context of our development at runtime, huge groups of developers with very heterogeneous backgrounds could work together nicely. Each developer could use tools that were fitting his knowledge and background.

Since our tools are early prototypes, there remain several points for future work. Probably the most important thing will be to conduct usability studies for all tools and improve them accordingly. Similarly, the combinability of tools should be investigated in more detail, supported by usability evaluations.

When looking at Figure 5.1, another area of future work becomes obvious. Since we haven't covered the whole range of task/paradigm combinations, yet, the implementation of tangible tools for usability evaluation, specification of dialog control and configuration of dataflow networks comes next.

Finally, we present a possible combination of our WIMP tools (T1, T3 and T4) that could help a usability engineer. In Figure 5.28, the UIC, DIVE and the user performance real-time visualizations are combined on one screen. The monitoring tool (bottom right) shows raw unfiltered event communication between service components, while at the same time showing all running services with full details on their states. The current version of our monitoring tool is really only useful for the programmer, but extensions are imaginable which also make this worthwhile for the usability engineer. In Figure 5.28, the UIC (bottom left) might help the usability engineer to understand where the user is currently within the interaction graph.

The real-time user performance measurement windows (top row) should enable the evaluation of the actual usability at the same time. While the first two tools could reveal that a certain action was successfully triggered by the user, it does not become apparent how many tries there were, at which time frame, or how many errors there have been until this final

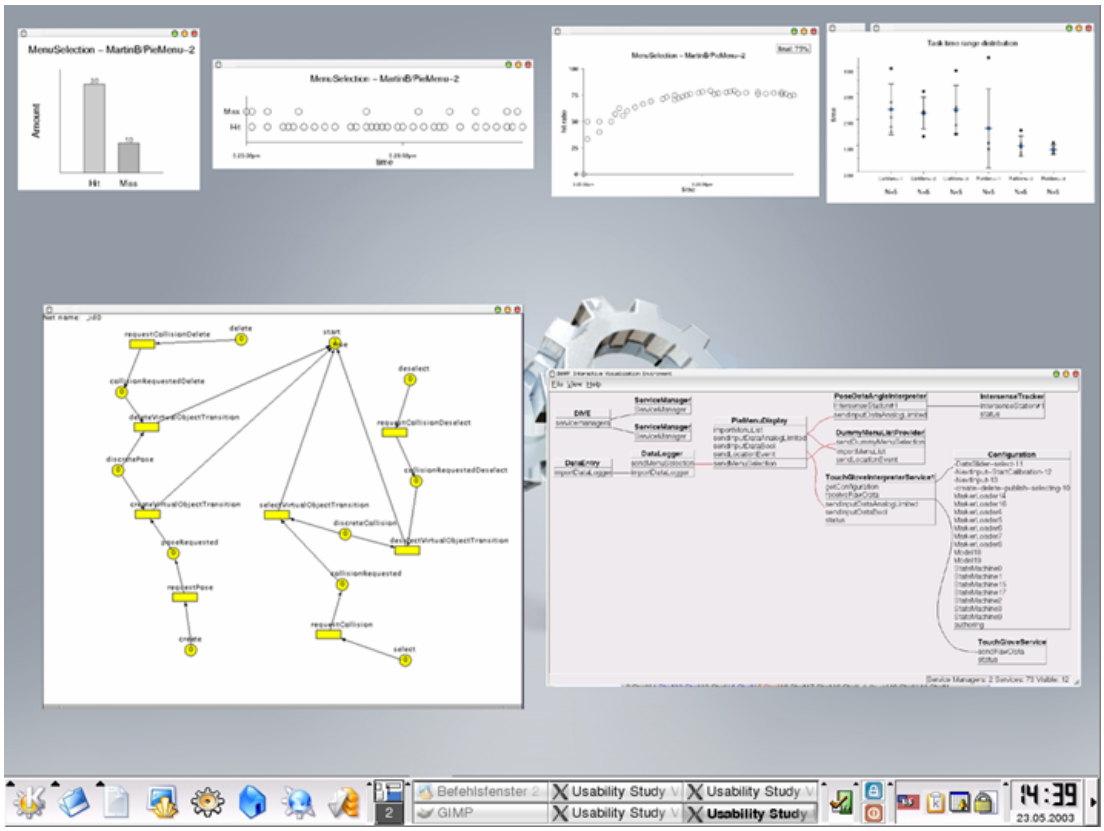


Figure 5.28: Usability engineer tool setup.

tool is taken into consideration. Observations in the top windows will likely usually lead to implementation fine tuning (e.g., to trigger actions differently) or they might reveal the need for a whole new service (e.g., to install a data filter for better usability), thereby in effect overhauling the design.

To be suspicious is not a fault. To be suspicious all the time without coming to a conclusion is the defect.

LU XUN

Conclusions

The AVANTGUARDE toolkit proves to be a suitable solution in many systems. The results achieved with the tools built on top of AVANTGUARDE are encouraging. However, several important challenges remain to be solved.

This concluding chapter first reviews the research contributions of my approach, embodied by AVANTGUARDE and its authoring tools. Then, I discuss how well the requirements presented in chapter 2 have been addressed with my approach by reflecting upon its benefits and limitations. Finally, possible directions for future research are pointed out.

6.1 Contributions

The core contribution of this thesis is threefold.

First, user interfaces for UAR have been defined and explored. During this discourse, two novel models have been proposed: the Spheres of Influence Model and the Model of Superimposed Lenses. Both models extend formerly known models with significant new ideas. The conclusions from these theoretical explorations have been applied throughout this thesis. Other researchers, trying to build similar systems, can benefit by applying these models and conclusions.

Within this thesis, a toolkit, AVANTGUARDE was developed. Its expressiveness is suitable for building UAR systems. The comparison to related work (Section 3.3) shows that this has not been achieved yet by other software infrastructures. AVANTGUARDE has been used in a large number of systems, with and without my participation. The accompanying webpage of this thesis [146] contains a full list of these systems. More detailed descriptions can be found in Sections 4.3 (SHEEP), 5.3.2 (Immersive configuration) and 5.5 (CAR). Furthermore, all tools described in Chapter 5 have been implemented with AVANTGUARDE. This proves the applicability of AVANTGUARDE to user interfaces in UAR. The design decisions for the architecture and components of AVANTGUARDE (Chapter 4) could be applied by other researchers independently from my implementation.

The prototypical authoring tools presented in Chapter 5 have been used and evaluated informally. They embody an exploration of the design space for tools to create UAR user interfaces. The novel idea of combining several lightweight tools, each using different user interface paradigms, into more powerful, cross-paradigm tools has been explored. The insights gained by these prototypes can be used by other researchers as guidelines for creating more authoring tools.

6.2 Discussion

This section discusses the benefits and limitations of the decisions made while developing AVANTGUARDE and its authoring tools to address the core challenges that were presented in Chapter 2.

6.2.1 The AVANTGUARDE Toolkit

AVANTGUARDE has been developed to address the specific requirements for user interfaces in UAR. A high-level description of AVANTGUARDE is that it is a hybrid of the relatively old idea of an User Interface Management System (UIMS) [115] and a component-based toolkit for dataflow networks. It is a UIMS approach, because of the application of a formal model (Petri nets in the User Interface Controller component), clear layering and well defined tokens. It also has a toolkit character regarding the flexibly connectable filters that form dataflow networks. Some UIMS also provide the capability to create dataflow networks (e.g., [77]), however they do it in a monolithic fashion, as opposed to our flexible and distributed toolkit AVANTGUARDE.

Graphical widgets that are encapsulated in Open Inventor nodes in the Viewer component can easily be reused and thereby are another feature of a lightweight toolkit. The combination of these two aspects fosters the advantages of both approaches. UIMS have nice properties regarding reusability and rapid prototyping. However they did not catch on [112], because of their limits regarding execution speed, difficulties to extend them for new interactions and their tendency to force programmers to use a certain specification model. The speed of interpretation of a formal model (in our case Petri nets) is hardly an issue anymore these days. We hope to overcome the last two concerns by using a lightweight, component-based approach.

By its continuous usage over three years in our research group, AVANTGUARDE has proven to fulfill the non-functional requirements of modifiability, reusability and maintainability. How well were the functional requirements of user interfaces in UAR addressed?

Multiple displays, input devices and users The flexible and loose coupling of components connected by the DWARF middleware proved to be a sound connectivity infrastructure for distributed user interfaces. The idea to control the interactions with a central component worked out well. Its implementation, the DWARF User Interface Controller, was powerful enough to handle even complex interactions with multiple displays, input devices and users.

Although multi-user setups were implemented with our framework (e.g., SHEEP), we did not address the social implications in depth. Humans apply certain usage patterns when collaborating (e.g., turn-taking). These patterns can be implemented with the User Interface Controller, but it does not provide dedicated facilities to address them.

Mixed reality displays AVANTGUARDE's Viewer component fully supports the requirements for mixed reality displays. It can display virtual objects in different reference frames by connecting objects in the Viewer's scenegraph with tracking data emitted by tracking components. The virtual objects can be superimposed on the real world by using optical- or video see-through

displays in realtime. By controlling several Viewer components with a User Interface Controller, visualizations consisting of several, possibly superimposed displays, can be implemented.

A problem for the Viewer components is the missing distributed data management. Since the state of a Viewer is determined only by the set of commands that it has received from a connected User Interface Controller, lost commands or restarts of a Viewer can lead to inconsistencies with other Viewers displaying the same scene.

Tangible interactions Several tangible interactions were implemented with AVANTGUARDE. It turned out that implementing them by specifying their logic in a User Interface Controller and possibly a dataflow network of Filter components is sufficient for most tangible interactions.

We have realized, however, that the crucial design decisions in tangible interactions are not so much on the software side, but on the design of the physical objects that act as tangible control for digital information. Industrial designers and psychologists are needed to decide on the best physical object to use. On the software side, once the functionality of the interaction has been specified, due to the standardized tokens for input devices, developers can experiment with different physical objects to find the best setup.

Context-awareness The flexible dataflow networks that can be built with AVANTGUARDE are applicable to model context-aware user interfaces. The core problem for context-aware systems is, however, to determine the current context. To do this, machine-learning approaches are promising, but have not yet been integrated into AVANTGUARDE. We focussed on the simulation of the results of context determination instead.

6.2.2 Authoring Tools

One of the major goals of our research is to provide a rapid prototyping environment within which new user interface ideas can be prototyped and tested easily. To this end, our tools have already proven to be suitable for joint, online development, testing and enhancement of interaction facilities, both individually and in multimodal, ubiquitous combinations.

Our idea to combine tools that employ different user interface paradigms is novel. We employ a combination of these user interface paradigms: tangible user interfaces, augmented reality and conventional WIMP user interfaces. By a combination of tools, which use different paradigms, the deficiencies of each paradigm can be compensated. This combination of tools with different user interfaces has led to some interesting insights. For example, there seems to be a trade-off between ease of use for a tool and the complexity of results that can be accomplished with it. WIMP tools can be used to model more complex interactions, whereas ease of use is greater with tools that have a tangible user interface or an augmented reality user interface. Specifically, I have made these observations about my tools:

WIMP. For tasks that require a huge amount of screen space or text input, a desktop based tool seems the best solution, since these actions are problems in other paradigms.

Tangible user interfaces. On the other hand, for tasks that are to be solved collaboratively and that mainly employ a direct manipulation approach, tangible authoring seems more applicable than desktop tools. A perfect example is moving the toy car in the CAR system (see Section 5.5).

Augmented reality user interfaces. Finally, for tasks that require spatial input (e.g., positioning a sound source in 3D [108]), authoring with an augmented reality user interface seems the best way.

We now discuss how well the functional requirements of authoring UAR user interfaces were addressed.

Monitoring the user We have demonstrated that usability evaluations can be seamlessly integrated into the live development and testing process [86]. To this end, user evaluation processes can be created to automatically inspect and evaluate the data streams flowing between the individual interaction devices, tangible objects, users, displays, etc.

The augmented reality visualization of the user's eye gaze has been very useful for developing attentive user interfaces, since the user's eye gaze is the core context, determining how to adapt the user interface.

Specifying dialog control The core facility for addressing these is the User Interface Controller Editor. It turned out to be difficult to use, since the Petri nets can become quite complicated for complex interactions. To address this problem, we support the development of a user interface entirely without using Petri nets and replacing them with simple Python scripting components. This is made possible by the DWARF component model. As long as the Python component has the same interface as the User Interface Controller, they can be transparently exchanged—even during system runtime. After fine-tuning the parameters in the Python component, the logic can easily be ported back into the Petri net model.

Configuring dataflow networks The combination of tools with a traditional graphical user interface (DIVE, Section 5.3.1) and tools that follow the UAR paradigm to address this task (Immersive configuration, Section 5.3.2) has led to some interesting insights. There seems to be a trade-off between ease of use for a tool and the complexity of results that can be reached with it. Conventional tools can be used to model more complex interactions, whereas the ease of use is greater with tools that have a UAR user interface.

Creating context-aware visualizations The principles embodied in the CAR environment can not only be used for user interfaces in cars. Simulation of spatial context by tangible interactions and modification of context-aware animations by sketching are techniques applicable to any kind of mobile user interface.

6.3 Future Work

During the development of the work presented in this thesis, several interesting research topics were identified, but not investigated in depth. This section concludes this thesis by showing how these issues could be used as directions for further research.

Persistence The problem of a missing persistency layer for AVANTGUARDE has been described by the example of the Viewer component in Section 6.2.1. However the same rationale is also applicable to other components. For robust systems, a persistency layer is mandatory. When thinking about the user interface experiments to which my approach is geared, it would be very useful to save and load user interface configurations, as well.

Specifying dialog control As pointed out already in Section 6.2.2 in the discussion of the limitations of the User Interface Controller, Python seems to be a faster way to specify interaction logic than our User Interface Controller. To further speed up this process, a Python package containing objects for modeling dialog control would be desirable.

Context determination Current research suggests that the determination of the user's current context is best done by machine-learning methods. The usage of these techniques introduces new problems. For example, how can the user correct false conclusions of a machine-learning algorithm. And, how can this false classification be detected? Since we were using a wizard of Oz type context determination, these problems never occurred in our prototypes.

Fully integrated immersive authoring environment The immersive authoring capabilities presented in Sections 5.3.2 and 5.5 can be seen as the first steps towards a bigger vision. The idea of collapsing authoring and runtime environment into one can finally result in a live, dynamically changeable and also dynamically adaptive development environment for UAR user interfaces. The environment can thereby provide us with the opportunity to easily explore, combine and test different currently emerging concepts, while also providing increasing degrees of automatic adaptation by the tools themselves. Another related idea is to bootstrap tools. Starting from a set of simple, generic tools, the developers can build new tools that can be used to build new tools, and so on.

The biggest concern of critics for an immersive authoring approach is that it requires sophisticated infrastructure such as trackers, head-mounted displays, and projectors. However, when developing UAR user interfaces, this concern is not justified. Since user interfaces in UAR *require* a sophisticated infrastructure to run, why not use this infrastructure for authoring purposes, as well?

Usability studies To quantify the usability of our tools, it will be necessary to conduct usability studies comparing multiple tools addressing the same task (e.g., one using a UAR user interface,

and another one using a traditional graphical user interface). For example, dataflow networks can be configured both with DIVE and with our immersive configuration environment.

Mobile setups So far, we have built only stationary or semi-mobile prototypes. Although we could prove with theoretical considerations that our approach is feasible for highly dynamical, heavily distributed systems, building prototypes that are fully mobile would be better suited to prove our claims. However, since a technical infrastructure for wide area tracking is still very expensive, this remains future work.

Development of new interaction techniques The feasibility of our tools has been shown in several projects. We have started to use our tools with experts from other research areas, such as human factors, psychologists, linguists and engineers. Our tools are now at a level where collaborative research can lead to the discovery of new interaction and visualization metaphors. A recent example that was realized with AVANTGUARDE is [173].

Device ontologies We have been able to achieve full flexibility for exchanging input devices at run-time. For output devices, however, we have only been partially successful. The underlying problem has been rather complex. It turned out that it is very difficult to define the semantic expressiveness of an output component (e.g., which auditory interfaces can be mapped to GUIs and which cannot)? For input components the definition of expressivenes was relatively simple, because the receiver of emited tokens is a *computer*. For output components, the receiver of content is a *human*. Perception and cognitive processing of information within the human mind are not yet understood well enough to come up with an ontology of output devices. Promising approaches to this complex problem can be found in [12, 28, 144, 186].

To refine our input and output ontologies, the work from Mackinlay and colleagues seems like a good starting point. They have already proposed a general input device ontology [99]. Furthermore, Mackinlay has proposed an ontology for output components; however, he only addressed graphical represenations of relational data [97, 98]. Applying these approaches to the domain of UAR seems very promising to me.

Bibliography

The page numbers, provided at the end of each entry, indicate on what pages this work was cited. All web pages referenced in this bibliography have been online in June 2005.

[1] A. R. T. GmbH. A. R. T. Homepage. `http://www.ar-tracking.de`. 96, 121

[2] Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. 72

[3] Advanced Visual Systems. Homepage of AVS. `http://www.avs.com`. 39

[4] Alias Systems. Maya Homepage. `http://www.alias.com`. 43

[5] David Anderson, James L. Frankel, Joe Marks, Aseem Agarwala, Paul Beardsley, Jessica Hodgins, Darren Leigh, Kathy Ryall, Eddie Sullivan, and Jonathan S. Yedidia. Tangible Interaction and Graphical Interpretation: A new Approach to 3D Modeling. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 393–402, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 47

[6] Apple. iPod Homepage. `http://www.apple.com/ipod/`. 29

[7] Apple. Shake Homepage. `http://www.apple.com/shake/`. 39

[8] ARTLab. OpenGL Homepage. `http://www.opengl.org`. 42

[9] Avid Technologies. Softimage Homepage. `http://www.softimage.com`. 43

[10] Ron T. Azuma. A Survey of Augmented Reality. *Presence, Special Issue on Augmented Reality*, 6(4):355–385, August 1997. 16, 29

[11] Ronald M. Baecker. *Interactive Computer-mediated Animation*. PhD thesis, Dep. of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA, March 1969. 43, 57

[12] Aleksandar M. Bakic, Matt W. Mutka, and Diane T. Rover. An On-line Performance Visualization Technology. *Softw. Pract. Exper.*, 33(15):1447–1469, 2003. 138

[13] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Stefan Riss, Christian Sandor, and Martin Wagner. Design of a Component-Based Augmented Reality Framework. In *Proceedings of the International Symposium on Augmented Reality (ISAR)*, pages 45–54, October 2001. 59

[14] Martin Bauer, Bernd Bruegge, Gudrun Klinker, Asa MacWilliams, Thomas Reicher, Christian Sandor, and Martin Wagner. An Architecture Concept for Ubiquitous Computing Aware Wearable Computers. In *Proceedings of the 2nd International Workshop on Smart Appliances and Wearable Computing (IWSAWC 2002)*, pages 785–790, Vienna, Austria, July 2002. 59, 69, 80

[15] Martin Bauer, Otmar Hilliges, Asa MacWilliams, Christian Sandor, Martin Wagner, Joe Newman, Gerhard Reitmayr, Tamer Fahmy, Gudrun Klinker, Thomas Pintaric, and Dieter Schmalstieg. Integrating Studierstube and DWARF. In *International Workshop on Software Technology for Augmented Reality Systems (STARS)*, October 2003. 69

[16] Kent Beck. *eXtreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA, USA, 1999. 33

[17] Ashweeni Kumar Beeharee, Roger Hubbold, and Adrian J West. Visual Attention Based Information Culling for Distributed Virtual Environments. In *VRST '03: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 213–222, Osaka, Japan, October 2003. 96

[18] Blaine Bell. *View Management for Distributed User Interfaces*. PhD thesis, Department of Computer Science, Columbia University, New York, NY, 2005. 18, 103

[19] Blaine Bell, Steven Feiner, and Tobias Höllerer. View Management for Virtual and Augmented Reality. In *UIST '01: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, pages 101–110, Orlando, Florida, 2001. 117, 123, 126

[20] Blaine Bell, Tobias Höllerer, and Steven Feiner. An Annotated Situation-awareness Aid for Augmented Reality. In *UIST '02: Proceedings of the 15th Annual ACM Symposium on User interface Software and Technology*, pages 213–216, Paris, France, 2002. ACM Press. 123

[21] Steve Benford and Lennart E. Fahlen. A Spatial Model of Interaction in Large Virtual Environments. In *ECSCW '93: Proceedings of the Third European Conference on Computer Supported Cooperative Work*, pages 109–124, 1993. 9, 14

[22] Eric Bergman. *Information Appliances and Beyond*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2000. 3

[23] Allen Bierbaum. VR Juggler: A Virtual Platform for Virtual Reality Application Development. Master's thesis, Iowa State University, 2000. 44, 50

[24] Gabor Blaskó and Steven Feiner. A Menu Interface for Wearable Computing. *6th International Symposium on Wearable Computers (ISWC 2002)*, pages 164–165, October 2002. 70

[25] Christian Boitet and Mark Seligman. The "Whiteboard" Architecture: A Way to Integrate Heterogeneous Components of NLP Systems. In *Proceedings of the 15th Conference on Computational Linguistics*, pages 426–430, Morristown, NJ, USA, 1994. Association for Computational Linguistics. 38

[26] Jeff Butterworth, Andrew Davidson, Stephen Hench, and Marc Olano. 3DM: A Three Dimensional Modeler Using a Head-Mounted Display. In *Proc. 1992 Symposium on Interactive 3D Graphics*, pages 135–138, Cambridge, Massachusetts, 1992. 49

[27] Andreas Butz, Clifford Beshers, and Steven Feiner. Of Vampire mirrors and privacy lamps: privacy management in multi-user augmented environments. In *UIST '98: Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, pages 171–172, New York, NY, USA, 1998. ACM Press. 107

[28] Stephen M. Casner. Task-analytic Approach to the Automated Design of Graphic Presentations. *ACM Trans. Graph.*, 10(2):111–151, 1991. 138

[29] Renato Cerqueira, Carlos Cassino, and Roberto Ierusalimschy. Dynamic Component Gluing Across Different Componentware Systems. In *DOA '99: Proceedings of the International Symposium on Distributed Objects and Applications*, page 362, Washington, DC, USA, 1999. IEEE Computer Society. 42

[30] Renato Cerqueira, Christopher K. Hess, Manuel Roman, and Roy H. Campbell. Gaia: A Development Infrastructure for Active Spaces. In *Workshop on Application Models and Programming Tools for Ubiquitous Computing*, pages 74–83, Atlanta, USA, 2001. 41, 42

[31] Wayne Citrin, Michael Doherty, and Benjamin Zorn. Design of a Completely Visual Object-Oriented Programming Language . In M. Burnett, A. Goldberg, and T. Lewis, editors, *Visual Object-Oriented Programming*, pages 19–35. Prentice-Hall, New York, 1995. 109

[32] Coast Team. Homepage of the Coast Project. `http://www.opencoast.org/`. 38

[33] Columbia University. Downloadpage for Spacemanager. `http://www1.cs.columbia.edu/~blaine/SpaceManager/license.htm`. 126

[34] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, and Kevin Christiansen. Alice: Lessons Learned from Building a 3D System for Novices. In *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 486–493, The Hague, The Netherlands, 2000. 44, 107, 118

[35] Cycling '74. Max/MSP Homepage. `http://www.cycling74.com/products/maxmsp.html`. 44, 45

[36] Jack B. Dennis. First Version of a Data Flow Procedure Language. In *Programming Symposium, Proceedings Colloque sur la Programmation*, pages 362–376, London, UK, 1974. Springer-Verlag. 39

[37] Richard W. DeVaul and Alex Pentland. Toward the Zero Attention Interface: Wearable Subliminal Cuing for Short Term Memory Support Memory. In *ISWC '02: Proceedings of the 6th IEEE International Symposium on Wearable Computers*, pages 141–142, 2002. 54

[38] Richard W. DeVaul, Michael Sung, Jonathan Gips, and Alex Pentland. MIThril 2003: Applications and Architecture. In *ISWC*, pages 4–11, 2003. 15, 38, 54

[39] Anind K. Dey. *Providing Support for Building Context-Aware Applications.* PhD thesis, Georgia Institute of Technology, 2000. 24

[40] Discreet. 3DS Max Homepage. `http://www.discreet.com/`. 43

[41] Ralf Dörner, Christian Geiger, Michael Haller, and Volker Paelke. Authoring Mixed Reality—a Component and Framework-based Approach. In *IWEC '02: Proceedings of First International Workshop on Entertainment Computing*, pages 405–413, Makuhari, Chiba, Japan, 2002. 41, 50

[42] Christoph Endres. Towards a Software Architecture for Device Management in Instrumented Environments. In *Doctoral Colloquium at Ubicomp*, pages 245–246, Seattle, USA, October 2003. IEEE Computer Society. 38, 41

[43] Greg Ewing. Pyrex Homepage. `http://www.cosc.canterbury.ac.nz/~greg/python/Pyrex/`. 42

[44] Lennart E. Fahlen and C. G. Brown. The Use of a 3D Aura Metaphor for Computer Based Conferencing and Teleworking. In *4th Multi-G Workshop*, pages 69–74, 1992. 9, 14

[45] Steven Feiner. Environment Management for Hybrid User Interfaces. *IEEE Personal Communications*, 7(5):50–53, October 2000. 4, 18

[46] Steven Feiner, Blair MacIntyre, Marcus Haupt, and Eliot Solomon. Windows on the World: 2D Windows for 3D Augmented Reality. In *UIST '93: Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, pages 145–155, 1993. 18, 29

[47] Steven Feiner and Ari Shamash. Hybrid User Interfaces: Breeding Virtually Bigger Interfaces for Physically Smaller Computers. In *UIST '91: Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology*, pages 9–17, Hilton Head, SC, November 11–13, 1991. 4

[48] James D. Foley and Victor L. Wallace. The Art of Natural Graphic Man-machine Conversation. *SIGGRAPH Comput. Graph.*, 8(3):87–87, 1974. 84

[49] James D. Foley, Victor L. Wallace, and Peggy Chan. The Human Factors of Computer Graphics Interaction Techniques. *IEEE Comput. Graph. Appl.*, 4(11):13–48, 1984. 67, 84

[50] Free Software Foundation. GCJ Homepage. `http://gcc.gnu.org/java/`. 75

[51] FreeWRL. FreeWRL Homepage. `http://freewrl.sourceforge.net`. 74

[52] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995. 22, 67, 73

[53] Markus Geipel. Run-time Development and Configuration of Dynamic Service Networks. `http://ar.in.tum.de/twiki/pub/DWARF/SepGeipel/SEP.pdf`, March 2005. 97, 98, 99

[54] Asa Granlund, Daniel Lafrentiere, and David A. Carr. A Pattern-supported Approach to the user interface design process. In *Workshop Report, UPA '99: Usability Professionals Association Conference*, Scottsdale, AZ, June 29–July 2 1999. 33

[55] Saul Greenberg and Chester Fitchett. Phidgets: Easy development of physical interfaces through physical widgets. In *UIST '01: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, pages 209–218, 2001. 46

[56] Griffin Technology. PowerMate Homepage. `http://www.griffintechnology.com/products/powermate/`. 70, 106

[57] Sinem Güven and Steven Feiner. A Hypermedia Authoring Tool for Augmented and Virtual Reality. *New Review of Hypermedia*, 9(1):89–116, 2003. 44, 46

[58] Matthias Haringer and Holger Regenbrecht. A Pragmatic Approach to Augmented Reality Authoring. In *ISMAR '02: Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 237–245, Darmstadt, Germany, 2002. 44

[59] Hans R. Harston and Jose C. Castillo. Critical Incident Data and Their Importance in Remote Usability Evaluation. In *Human Factors and Ergonomics Society 44th Annual Meeting*, pages 590–593, 2000. 92

[60] Havok. Homepage of the Havok Physics Engine. `http://www.havok.com/products/physics.php`. 46

[61] Pilar Herrero and Angelica de Antonio. A Formal Awareness Model for 3D Web-based Collaborative Environments. *SIGGROUP Bull.*, 21(3):49–53, 2000. 9, 14

[62] Ralph D. Hill. Supporting Concurrency, Communication and Synchronization in Human-Computer Interaction—The Sassafras UIMS. *ACM Trans. Graph.*, 5(3):179–210, 1986. 41

[63] Otmar Hilliges. Development of a 3D-Viewer Component for DWARF. `http://www1.in.tum.de/pub/DWARF/SepHilliges/main.pdf`, 2003. 75, 76

[64] Otmar Hilliges, Christian Sandor, and Gudrun Klinker. A Lightweight Approach for Experimenting with Tangible Interaction Metaphors. In *Proc. of the International Workshop on Multi-user and Ubiquitous User Interfaces (MU3I)*, 2004. 11, 57, 88, 129

[65] Tobias Höllerer, Steven Feiner, and John Pavlik. Situated Documentaries: Embedding Multimedia Presentations in the Real World. In *ISWC '99: Proceedings of the 3rd IEEE International Symposium on Wearable Computers*, pages 79–86, Washington, DC, USA, 1999. IEEE Computer Society. 46

[66] Roland Holm, Erwin Stauder, Roland Wagner, Markus Priglinger, and Jens Volkert. A Combined Immersive and Desktop Authoring Tool for Virtual Environments. In *VR '02: Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 93–100, 2002. 10, 50, 52, 56

[67] Human Interface Technology Laboratory Washington. ARToolkit Homepage. `http://www.hitl.washington.edu/artoolkit`. 42, 101

[68] Human Interface Technology Laboratory Washington. Example Image of Augmented Reality. `http://www.hitl.washington.edu/`. 3, 18

[69] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 409–416. ACM Press/Addison-Wesley Publishing Co., 1999. 43

[70] Thomas A. Furness III. The Application of Head-Mounted Displays to Airborne Reconnaissance and Weapon Delivery. Technical Report TR-69-241, U.S. Air Force Avionics Laboratory, Wright-Patterson AFB, OH, April 1969. 122

[71] Infusionsystems. i-CubeX Homepage. `http://infusionsystems.com/catalog/index.php`. 46, 70, 106

[72] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the Future: The Story of Squeak, A Practical Smalltalk Written in Itself. In *OOPSLA '97: Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 318–326, Atlanta, Georgia, United States, November 1997. 42, 44

[73] Intel. OpenCV Homepage. `http://www.intel.com/research/mrl/research/opencv/`. 42

[74] H. Ishii and B. Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *CHI '97: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Atlanta, USA, March 1997. ACM. 118

[75] Robert J. K. Jacob. A State Transition Diagram Language for Visual Programming. In Ephraim P. Glinert, editor, *Visual Programming Environments: Paradigms and Systems*, pages 51–59. IEEE Computer Society Press, 1990. 109

[76] Robert J. K. Jacob. Eye tracking in advanced interface design. In W. Barfield and T. Furness, editors, *Virtual environments and advanced interface design*, pages 258–288, New York, NY, USA, 1995. Oxford University Press, Inc. 94, 96

[77] Robert J. K. Jacob, Leonidas Deligiannidis, and Stephen Morrision. A Software Model and Specification Language for Non-WIMP User Interfaces. *ACM Transactions on Computer-Human Interaction*, 6(1):1–46, 1999. 41, 44, 45, 55, 57, 134

[78] Ramaswamy Jagannathan and Earl A. Ashcroft. Eazyflow: A Hybrid Model for Parallel Processing. In *Proceedings of the International Conference on Parallel Processes*, pages 514–523. IEEE, August 1984. 109

[79] Paul Janecek, Anne V. Ratzer, and Wendy E. Mackay. Redesigning Design/CPN: Integrating Interaction and Petri Nets In Use. In *Proceedings of Second Workshop on Practical Use of Coloured Petri Nets and Design/CPN*, pages 119–133, Aarhus, Denmark, October 1999. 115

[80] Jfern Team. Jfern Homepage. `http://sourceforge.net/projects/jfern`. 73

[81] Brad Johanson and Armando Fox. Extending tuplespaces for coordination in interactive workspaces. *J. Syst. Softw.*, 69(3):243–266, 2004. 54

[82] Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *IEEE Pervasive Computing*, 1(2):67–74, 2002. 38, 54

[83] Graham Kirby. Dynamic Compilation in Java. `http://www-ppg.dcs.st-and.ac.uk/Java/DynamicCompilation/`. 111

[84] Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. Papier-Mâché: Toolkit Support for Tangible Input. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Vienna, Austria, 2004. 46, 47

[85] Narayanan Kodiyalam. Remote Usability Evaluation Tool. Master's thesis, Virginia Polytechnic Institute and State University, 2003. 91

[86] Christian Kulas. Usability Engineering for Ubiquitous Computing. Master's thesis, Technische Universität München, München, Deutschland, 2003. 84, 93, 94, 117, 136

[87] Christian Kulas, Christian Sandor, and Gudrun Klinker. Towards a Development Methodology for Augmented Reality User Interfaces. In *MIXER '04: Proc. of the International Workshop Exploring the Design and Engineering of Mixed Reality Systems*, Funchal, Madeira, 2004. 10, 88, 90, 128

[88] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Interactive Control For Physically-Based Animation. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 201–208, New Orleans, LA, 2002. 43

[89] Florian Ledermann. An Authoring Framework for Augmented Reality Presentations. Master's thesis, Technische Universität Wien, 2002. 44

[90] Gun A. Lee, Gerard Jounghyun Kim, and Chan-Mo Park. Modeling Virtual Object Behavior within Virtual Environment. In *VRST '02: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 41–48, New York, NY, USA, 2002. ACM Press. 49

[91] Gun A. Lee, Claudia Nelles, Mark Billinghurst, and Gerard Jounghyun Kim. Immersive Authoring of Tangible Augmented Reality Applications. In *ISMAR '04: Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 172–181, Arlington, VA, November 2–5, 2004. IEEE Computer Society. 10, 11, 49, 50, 56, 101

[92] Henry Lieberman, editor. *Your Wish Is My Command*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2001. 108, 115

[93] Kent Lyons and Thad Starner. Mobile Capture for Wearable Computer Usability Testing. In *Proceedings of IEEE International Symposium on Wearable Computing (ISWC), October 08-09 2001, Zurich, Switzerland*, pages 69–76, 2001. 3

[94] Blair MacIntyre and Steven Feiner. Language-Level Support for Exploratory Programming of Distributed Virtual Environments. In *UIST '96: Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, pages 83–94, Seattle, WA, November 6–8 1996. 42, 55

[95] Blair MacIntyre and Steven Feiner. A Distributed 3D Graphics Library. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 361–370, Orlando, FL, July 19–24 1998. 38, 41, 42, 55

[96] Blair MacIntyre, Maribeth Gandy, Steven Dow, and Jay D. Bolter. DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences. In *UIST '04:*

Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology, pages 197–206, 2004. 10, 44, 46, 118

[97] Jock D. Mackinlay. Automatic Design of Graphical Presentations, 1986. Unpublished Doctoral Dissertation, Stanford University, Computer Science Department, Stanford, CA. Also Tech. Rep. Stan-CS-86-1038. 138

[98] Jock D. Mackinlay. Automating the Design of Graphical Presentations of Relational Data. *ACM Transactions on Graphics*, 5(2):110–141, 1986. 138

[99] Jock D. Mackinlay, Stuart K. Card, and George G. Robertson. A Semantic Analysis of the Design Space of Input Devices. *Human-Computer Interaction*, 5(2):145–190, 1990. 67, 84, 138

[100] Asa MacWilliams. Software Development Challenges for Ubiquitous Augmented Reality. In *GI Augmented Reality and Virtual Reality Workshop*, September 2004. 6

[101] Asa MacWilliams. *A Decentralized Adaptive Architecture for Ubiquitous Augmented Reality Systems*. PhD thesis, Technische Universität München, München, Germany, 2005. 2, 7, 8, 25, 59, 60, 64

[102] Asa MacWilliams, Thomas Reicher, and Bernd Brügge. Decentralized Coordination of Distributed Interdependent Services. In *IEEE Distributed Systems Online – Middleware '03 Work in Progress Papers*, Rio de Janeiro, Brazil, June 2003. 62, 112

[103] Asa MacWilliams, Christian Sandor, Martin Wagner, Martin Bauer, Gudrun Klinker, and Bernd Brügge. Herding Sheep: Live System Development for Distributed Augmented Reality. In *ISMAR '03: Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 123–132, Tokyo, Japan, 2003. 10, 27, 59, 69, 74, 77, 84, 117, 121

[104] Mark Maybury and Wolfgang Wahlster. *Readings in Intelligent User Interfaces*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1998. 36

[105] Deborah J. Mayhew. *The Usability Engineering Lifecycle*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1991. 26

[106] Mark P. McCahill and Julian Lombardi. Design for an Extensible Croquet-Based Framework to Deliver a Persistent, Unified, Massively Multi-User, and Self Organizing Virtual Environment.
`http://www.opencroquet.org/About_Croquet/whitepapers.html`. 42, 44

[107] M. Douglas McIlroy. Mass Produced Software Components. In P. Naur and B. Randell, editors, *Proceedings, NATO Conference on Software Engineering*, pages 88–89, Garmisch, Germany, October 1968. 41, 61

[108] Frank Melchior, Tobias Lauterbach, and Diemer de Vries. Authoring and User Interaction for the Production of Wave Field Synthesis Content in an Augmented Reality System. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 48–51, Vienna, Austria, October 2005. 136

[109] Microvision. Nomad Homepage. `http://www.microvision.com/nomadexpert/`. 81

[110] Paul Milgram and Fumio Kishino. A Taxonomy of Mixed Reality Visual Displays. *IEICE Transactions on Information Systems*, E77-D(12):1321–1329, December 1994. 9, 16, 17

[111] Mark R. Mine, Frederick P. Brooks, Jr., and Carlo H. Sequin. Moving Objects in Space: Exploiting Proprioception In Virtual-Environment Interaction. *Computer Graphics*, 31(Annual Conference Series):19–26, 1997. 18

[112] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, Present, and Future of User Interface Software Tools. In John M. Carroll, editor, *Human-Computer Interaction in the New Millenium*, pages 213–234. Addison-Wesley Publishing, Reading, MA, 2002. 25, 39, 41, 83, 134

[113] Joseph Newman, David Ingram, and Andy Hopper. Augmented Reality in a Wide Area Sentient Environment. In *ISAR '01: Proceedings of the IEEE and ACM International Symposium on Augmented Reality*, pages 77–86, Washington, DC, USA, 2001. IEEE Computer Society. 38, 54

[114] Vinko Novak, Christian Sandor, and Gudrun Klinker. An AR Workbench for Experimenting with Attentive User Interfaces. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 284–285, Arlington, VA, USA, November 2004. 10, 56, 88, 90, 128, 130

[115] Dan Olsen. *User Interface Management Systems: Models and Algorithms*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1992. 11, 37, 39, 134

[116] Alex Olwal. Unit—A Modular Framework for Interaction Technique Design, Development and Implementation. Master's thesis, Department of Numerical Analysis and Computer Science at the Royal Institute of Technology (KTH), Stockholm, Sweden, 2002. 39, 51, 55, 56, 70

[117] Alex Olwal, Hrvoje Benko, and Steven Feiner. SenseShapes: Using Statistical Geometry for Object Selection in a Multimodal Augmented Reality System. In *ISMAR '03: Proceedings of the The 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 300–301, Washington, DC, USA, 2003. IEEE Computer Society. 56

[118] Alex Olwal and Steven Feiner. Unit: Modular Development of Distributed Interaction Techniques for Highly Interactive User Interfaces. In *GRAPHITE '04: International Conference on Computer Graphics and Interactive Techniques*, pages 131–138, Singapore, June 15–18, 2004. ACM Press. 40, 51, 70, 103

[119] Sharon L. Oviatt. Mutual Disambiguation of Recognition Errors in a Multimodel Architecture. In *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 576–583, 1999. 36

[120] Sharon L. Oviatt. Ten Myths of Multimodal Interaction. *Communications of the ACM*, 42(11):74–81, 1999. 70

[121] Charles Owen, Arthur Tang, and Fan Xiao. ImageTclAR: A Blended Script and Compiled Code Development System for Augmented Reality. In *International Workshop on Software Technology for Augmented Reality Systems (STARS)*, pages 23–28, October 2003. 42

[122] Laila Paganelli and Fabio Paternò. Intelligent Analysis of User Interactions with Web Applications. In *ACM Symposium on Intelligent User Interfaces, San Francisco, CA, 2002*, pages 111–118, 2002. 33

[123] Parallelgraphics. Parallelgraphics Homepage. `http://www.parallelgraphics.com`. 74

[124] Ken Perlin and David Fox. Pad: An Alternative Approach to the Computer Interface. *Computer Graphics*, 27(Annual Conference Series):57–72, 1993. 100

[125] Phidgets Inc. Phidgets Homepage. `http://www.phidgets.com`. 42, 46

[126] Wayne Piekarski. *Interactive 3D Modelling in Outdoor Augmented Reality Worlds*. PhD thesis, University of South Australia, 2002. 38, 50, 54

[127] Ben Piper, Carlo Ratti, and Hiroshi Ishii. Illuminating Clay: A 3-D Tangible Interface for Landscape Analysis. In *CHI '02: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 355–362, Minneapolis, Minnesota, US, 2002. ACM Press. 22, 23

[128] Ploticus Team. Ploticus Homepage. `http://ploticus.sourceforge.net`. 93

[129] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR. In *UIST '96: ACM Symposium on User Interface Software and Technology*, pages 79–80, 1996. 37, 68

[130] Ivan Poupyrev, Desney S Tan, Mark Billinghurst, Hirokazu Kato, Holger Regenbrecht, and Nobuji Tetsutani. Tiles: A Mixed Reality Authoring Interface. In *INTERACT '01: 7th Conference on Human-Computer Interaction*, pages 334–341, Tokyo, Japan, 2001. 49

[131] Ivan Poupyrev, Suzanne Weghorst, and Sidney Fels. Non-isomorphic 3D Rotational Techniques. In *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 540–547, New York, NY, USA, 2000. ACM Press. 29

[132] Ivan Poupyrev, Suzanne Weghorst, Takahiro Otsuka, and Tadao Ichikawa. Amplifying Spatial Rotations in 3D Interfaces. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 256–257, New York, NY, USA, 1999. ACM Press. 29

[133] Daniel Pustka. Visualizing Distributed Systems of Dynamically Cooperating Services. `http://ar.in.tum.de/Students/SepPustka`, March 2003. 97

[134] Python Software Foundation. Python Homepage. `http://www.python.org`. 42

[135] Eric S. Raymond. *The Art of UNIX Programming*. Addison-Wesley, Reading, MA, USA, 2004. 15

[136] Holger Regenbrecht and Martin Wagner. Interaction in a Collaborative Augmented Reality Environment. In *CHI '02: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 504–505, Minneapolis, USA, 2002. 39

[137] Thomas Reicher. *A Framework for Dynamically Adaptable Augmented Reality Systems*. PhD thesis, Technische Universität München, München, Germany, 2004. 7, 59

[138] Thomas Reicher, Asa MacWilliams, Bernd Bruegge, and Gudrun Klinker. Results of a Study on Software Architectures for Augmented Reality Systems. In *ISMAR '03: Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 274–275, Tokyo, Japan, 2003. 36, 61

[139] Wolfgang Reisig. Petri Nets, An Introduction. *EATCS Monographs on theoretical Computer science*, 4, 1985. 73

[140] Gerhard Reitmayr and Dieter Schmalstieg. An Open Software Architecture for Virtual Reality Interaction. In *VRST '01: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 47–54, Banff, Alberta, Canada, 2001. 44

[141] Gerhard Reitmayr and Dieter Schmalstieg. OpenTracker—An Open Software Architecture for Reconfigurable Tracking Based on XML. In *VR '01: Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 285–286, 2001. 39, 55, 69

[142] Jun Rekimoto. Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments. In *UIST '97: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pages 31–39, 1997. 80

[143] Jun Rekimoto, Brygg Ullmer, and Haruo Oba. DataTiles: A Modular Platform for Mixed Physical and Graphical Interactions. In *CHI '01: Proceedings of the SIGCHI*

Conference on Human Factors in Computing Systems, pages 269–276, New York, NY, USA, 2001. ACM Press. 48

[144] Steven F. Roth and Joe Mattis. Data Characterization for Intelligent Graphics Presentation. In *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 193–200, New York, NY, USA, 1990. ACM Press. 138

[145] II Russell M. Taylor, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. VRPN: A Device-Independent, Network-Transparent VR Peripheral System. In *VRST '01: Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 55–61, New York, NY, USA, 2001. ACM Press. 39, 46

[146] Christian Sandor. Accompanying Webpage to this Thesis. `http://www.ubiquitousaugmentedreality.com`. 8, 11, 70, 133

[147] Christian Sandor, Blaine Bell, Alex Olwal, Nick Temiyabutr, and Steven Feiner. Visual End User Configuration of Hybrid User Interfaces. In *ACM SIGMM 2004 Workshop on Effective Telepresence, New York, NY, USA, Oct. 15, 2004.*, pages 67–68, November 2004. 11, 56, 75, 88, 97, 100, 127, 129

[148] Christian Sandor and Gudrun Klinker. A Rapid Prototyping Software Infrastructure for User Interfaces in Ubiquitous Augmented Reality. *Personal Ubiquitous Comput.*, 9(3):169–185, 2005. 9, 59, 65, 103, 108

[149] Christian Sandor, Asa MacWilliams, Martin Wagner, Martin Bauer, and Gudrun Klinker. SHEEP: The Shared Environment Entertainment Pasture. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, October 2002. 59, 77

[150] Christian Sandor, Alex Olwal, Blaine Bell, and Steven Feiner. Immersive Mixed-Reality Configuration of Hybrid User Interfaces. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 110–113, Vienna, Austria, October 2005. 11, 56, 88, 97, 100, 127, 129

[151] Bill Schilit, Norman Adams, and Roy Want. Context-Aware Computing Applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994. 7, 24

[152] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavari, L. Miguel Encarnaco, Michael Gervautz, and Werner Purgathofer. The Studierstube Augmented Reality Project. *Presence*, 11(1):33–54, 2002. 38, 41, 42, 54

[153] Albrecht Schmidt. Implicit Human Computer Interaction Through Context. *Personal Technologies*, 4(2+3):191–199, June 2000. 24

[154] R. Schoenfelder, B. Bruederlin, G. Wolf, M. Reessing, and R. Krueger. A Pragmatic Approach to a VR/AR Component Integration Framework for Rapid System Setup. In *1. Paderborner Workshop: Augmented Reality/Virtual Reality in der Produktentstehung*, Paderborn, Germany, 2002. 44, 50

[155] Ben Shneiderman. *Designing the User Interface*. Addison-Wesley Publishing, Reading, MA, 1997. 26

[156] Shalin Shodhan. The Panda3D Engine—Python Scripting for Game and Simulation Development. `http://www.python.org/pycon/dc2004/papers/29/`. 42, 44

[157] Smalltalk Team. Smalltalk Homepage. `http://www.smalltalk.org`. 42

[158] David A. Smith, Alan Kay, Andreas Raab, and David P. Reed. Croquet—A Collaboration System Architecture. `http://www.opencroquet.org/About_Croquet/whitepapers.html`. 42, 44

[159] David A. Smith, Andreas Raab, David P. Reed, and Alan Kay. Croquet: A Menagerie of New User Interfaces. `http://www.opencroquet.org/About_Croquet/whitepapers.html`. 42, 44

[160] Russell Smith. ODE Homepage. `http://www.ode.org`. 42

[161] Deyang Song and MichaeMichaell Norman. Nonlinear Interactive Motion Control Techniques for Virtual Space Navigation. In *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 111–117, 1993. 37, 68

[162] Computer Graphics staff. Status Report of the Graphic Standards Planning Committee. *SIGGRAPH Comput. Graph.*, 13(3):1–10, 1979. 84

[163] A. Steed and M. Slater. A Dataflow Representation for Defining Behaviours within Virtual Environments. In *VRAIS '96: Proceedings of the 1996 Virtual Reality Annual International Symposium*, pages 163–167, Washington, DC, USA, 1996. IEEE Computer Society. 48, 49, 56

[164] Randy Stiles and Michael Pontecorvo. Lingua Graphica: A Visual Language for Virtual Environments. In *Proceedings of the IEEE Workshop on Visual Languages*, pages 225–227, 1992. 48, 49, 56

[165] Paul S. Strauss and Rikk Carey. An Object-Oriented 3D Graphics Toolkit. In *SIGGRAPH '92: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, pages 341–349, 1992. 75

[166] Hideyuki Suzuki and Hiroshi Kato. Interaction-level Support for Collaborative Learning: AlgoBlock—An Open Programming Language. In *CSCL '95: The First International Conference on Computer Support for Collaborative Learning*, pages 349–355, Mahwah, NJ, USA, 1995. Lawrence Erlbaum Associates, Inc. 47, 48

[167] Swig Team. Swig Homepage. `http://www.swig.org`. 42

[168] Systems in Motion. Coin3D Homepage. `http://www.coin3d.org`. 75

[169] Peter Tandler. The BEACH Application Model and Software Framework for Synchronous Collaboration in Ubiquitous Computing Environments. *J. Syst. Softw.*, 69(3):267–296, 2004. 38

[170] TCL Team. TCL Homepage. `http://tcl.sourceforge.net/`. 42

[171] Technische Universität München. Janus Homepage. `http://www.ergonomie.tum.de/forschung/versuchseinrichtungen/sonstiges.htm`. 71, 96

[172] Matthew Thorne, David Burke, and Michiel van de Panne. Motion Doodles: An Interface for Sketching Character Motion. *ACM Trans. Graph.*, 23(3):424–431, 2004. 43, 57

[173] Marcus Tönnis, Christian Sandor, Gudrun Klinker, Christian Lange, and Heiner Bubb. Experimental Evaluation of an Augmented Reality Visualization for Directing a Car Driver's Attention. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 56–59, Vienna, Austria, October 2005. 138

[174] Shinji Uchiyama, Kazuki Takemoto, Kiyohide Satoh, Hiroyuki Yamamoto, and Hideyuki Tamura. MR Platform: A Basic Body on Which Mixed Reality Applications are Built. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, pages 246–255, Darmstadt, Germany, October 2002. 42

[175] B. Ullmer and H. Ishii. The metaDESK: Models and Prototypes for Tangible User Interfaces. In *UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 223–232, Banff, Alberta, Canada, October 14–17 1997. ACM Press. 80

[176] Brygg Ullmer. *Tangible Interfaces for Manipulating Aggregates of Digital Information*. PhD thesis, Massachusetts Institute of Technology, 2002. 46, 88

[177] Brygg Ullmer and Hiroshi Ishii. Emerging Frameworks for Tangible User Interfaces. *IBM Systems Journal*, 39(3–4):915–931, 2000. 22, 23

[178] Donald L. Vickers. *Sorcerer's Apprentice: Head-Mounted Display and Wand*. PhD thesis, Department of EE, University of Utah, Salt Lake City, UT, 1972. 103

[179] Viewpoints Research Institute. Squeakland Homepage. `http://www.squeakland.org`. 44

[180] Martin Wagner. *Tracking with Multiple Sensors*. PhD thesis, Technische Universität München, München, Germany, 2005. 7, 24, 59

[181] Mark Weiser. The Computer for the Twenty-First Century. *Scientific American*, pages 94–100, September 1991. 2

[182] Wikipedia. Definition of Dataflow. `http://en.wikipedia.org/wiki/Data_flow`. 39

[183] Wolfgang Wohlgemuth and Gunthard Triebfürst. ARVIKA: Augmented Reality for Development, Production and Service. In *DARE '00: Proceedings of DARE 2000 on Designing Augmented Reality Environments*, pages 151–152, New York, NY, USA, 2000. ACM Press. 38, 41, 44, 54

[184] Jürgen Zauner, Michael Haller, Alexander Brandl, and Werner Hartmann. Authoring of a Mixed Reality Assembly Instructor for Hierarchical Structures. In *ISMAR '03: Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 237–246, Tokyo, Japan, 2003. 50

[185] Shumin Zhai, Carlos Morimoto, and Steven Ihde. Manual and Gaze Input Cascaded (MAGIC) Pointing. In *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 246–253. ACM Press, 1999. 94

[186] Michelle X. Zhou and Steven K. Feiner. Automated Visual Presentation: From Heterogeneous Information to Coherent Visual Discourse. *J. Intell. Inf. Syst.*, 11(3):205–234, 1998. 138

Index

Numbers

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X