User Interfaces for Mobile Augmented Reality Systems

Tobias Hans Höllerer

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2004

Abstract

User Interfaces for Mobile Augmented Reality Systems

Tobias Hans Höllerer

In this dissertation, we present typical components of, useful services associated with, and user interactions possible with mobile augmented reality systems, based on a comprehensive series of hardware and software infrastructures and application prototypes we developed. We define a practical taxonomy of user interface components for such systems and establish methodology for adaptive mobile augmented reality interfaces that dynamically rearrange themselves in response to changes in user context.

The research contributions to the state-of-the-art in augmented reality begin with the author's participation in the design of the "Columbia Touring Machine" in 1997, the first example of an outdoor mobile augmented reality system, and his lead in developing later prototypes. We develop a series of hardware and software infrastructures for prototyping mobile augmented reality applications that allow multiple users to participate in collaborative tasks taking place indoors and outdoors.

We present exploratory user interfaces for many different applications and user scenarios, including the Situated Documentaries application framework for experiencing spatially distributed hypermedia presentations. Based on these explorations, we develop a taxonomic categorization of mobile augmented reality interface components and their properties. Virtual and real world objects alike are considered part of the interface. We tag each component with information about its purpose, its intrinsic properties, its relationship to other objects, and its capabilities and flexibility with regard to various manipulations.

Mobile augmented reality has until now faced a significant challenge: the complexity of the augmented views rapidly increases when many virtual objects fight for screen space to annotate physical entities in the dynamic views of multiple fast-paced roaming users. Responding to this, we develop user interface management techniques for mobile augmented reality. A rule-based reasoning architecture uses the taxonomic data classification mentioned above to automatically rearrange augmented reality views in dynamic situations; for example to remove clutter in the augmented view of the world or to react to infrastructural context changes, such as variations in tracking accuracy.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank the following people for their help, advice, and support:

- My advisor, Steven Feiner for academic guidance, support, trust, and the academic research and teaching opportunities he provided me.

- The members of my thesis committee: Ron Azuma, Chris Codella, John Kender, and Peter Allen, for valuable advice, comments, and discussions.

- Blair MacIntyre for designing Coterie, jump-starting the mobile augmented reality project, and for valuable advice and discussions.

- John Pavlik for his ideas and promotional activities regarding the application of mobile augmented reality to the field of Journalism.

- Andreas Butz for the collaboration during his PostDoc year at Columbia's Computer Graphics and User Interfaces Lab.

- Blaine Bell, for close collaboration on many lab-related software development and research projects.

- Various members of the Computer Graphics and User Interfaces Lab, who contributed greatly to the MARS project: Drexel Hallaway, Elias Gagas, Tachio Terauchi, Gus Rashid, Navdeep Tinna, and Ryuji "Kukky" Yamamoto for their work on the MARS infrastructure; Clifford Beshers, Simon Lok, Hrvoje Benko, Gábor Blaskó, Sinem Güven, and Tiantian Zhou for their valuable contributions. Everyone who shared with me the joy and effort of participating and helping in demos and exhibitions.

- Alan Crosswell for his initiative in setting up and maintaining a GPS base station, and in deploying a campus-wide WiFi network.

- My colleagues at the Naval Research Laboratory, who I collaborated with on many aspects of Mobile AR. In particular, Simon Julier, Yohan Baillot, Dennis Brown, Marco Lanzagorta, Ed Swan, Mark Livingston, and Larry Rosenblum.

- Tiberiu Chelcea for accommodating me at his place whenever I visited Columbia after I had moved away from New York City.

This work was funded in part by ONR Contracts N00014-97-1-0838, N00014-99-1-0249, N00014-99-1-0394, and N00014-99-0683; NSF Grants CDA-92-23009, EEC-94-44246, IIS-00-82961, and IIS-01-21239; the Advanced Network & Services National Tele-Immersion Initiative; and gifts from IBM, Intel, Microsoft, and Mitsubishi Electric Research Laboratories.

Für meine Eltern, Renate and Walter

Overture

Coppélius's aria from *Les Contes d'Hoffmann* by Jacques Offenbach.
Libretto by Jules Barbier. Passage translated by the author of this thesis.

COPPÉLIUS

Chacun de ces lorgnons
rend noir comme le jais,
ou blanc comme l'hermine.
Assombrit, illumine, éclaire,
ou flétrit les objets.
J'ai des yeux, de vrais yeux,
des yeux vivants, des yeux de flamme,
des yeux merveilleux
qui vont jusque au fond de l'âme
et qui même en bien des cas
en peuvent prêter
une à ceux qui n'en ont pas.
J'ai des yeux, de vrais yeux vivants,
des yeux de flamme.
J'ai des yeux, de beaux yeux! Oui!
Veux-tu voir le cœur d'une femme?
S'il est pur ou s'il est infâme!
Ou bien préfères-tu le voir,
le voir tout blanc quand il est noir?
Prends et tu verras ce que tu voudras.
Prenez mes yeux, mes yeux vivants,
mes yeux de flamme,
mes yeux qui percent l'âme.
Prenez mes yeux!

Each one of these eyeglasses
makes black jet black,
or white ermine white.
Obscures, illuminates, lights,
or fades the objects.
I have eyes, true eyes,
lively eyes, eyes of fire,
marvelous eyes,
which see to the depths of the soul
and in many cases
can even lend a soul
to those who lack one.
I have eyes, true lively eyes,
eyes of fire.
I have eyes, beautiful eyes! Yes!
Would you like to see into a woman's heart?
If it is pure or if it is infamous!
Or do you prefer to see it,
to see it all white when it is black?
Take and you will see whatever you wish.
Take my eyes, my lively eyes,
my eyes of fire,
my eyes, which penetrate the soul.
Take my eyes!

HOFFMANN

(mettant le lorgnon)
Plaisanterie!

COPPÉLIUS

Mais non!
(Il soulève brusquement la portière.)

HOFFMANN

Quoi?

COPPÉLIUS

Magie!

Chapter 1

Introduction

This dissertation presents research results pertinent to user interfaces for mobile augmented reality. In this chapter we give an introduction to the work and provide an overview of our research contributions. We begin by briefly presenting our definitions of *augmented reality* (AR) and *mobile augmented reality systems* (MARS) in Section 1.1. Section 1.2 gives an overview of the thesis work and its research contributions. Chapter 2 will provide a more thorough introduction to mobile AR in general.

1.1 Mobile Augmented Reality

The idea of AR is related to the concept of *virtual reality* (VR). VR attempts to create an artificial world that a person can experience and explore interactively, predominantly through his or her sense of vision, but also via audio, tactile, and other forms of feedback. AR also brings about an interactive experience, but aims to supplement the real world, rather than creating an entirely artificial environment around the user. The physical objects in the individual's surroundings become the backdrop and target items for computer-generated annotations, or even modifications. Different researchers subscribe to narrower or wider definitions of exactly what constitutes AR. While the research community largely agrees on most of the elements of AR systems, helped along by the exchange and discussions at several international conferences in the field, there are still small differences in opinion and nomenclature. For the purpose of this dissertation we follow the definition of Azuma (1997) and Azuma et al. (2001). We define an AR system as one that combines real and computer-generated information in a real environment, interactively and in real time, and that aligns virtual objects with physical ones. AR is a subfield of the broader concept of *mixed reality* (MR) (Drascic and Milgram, 1996), which further includes simulations predominantly taking place in the virtual domain and not in the real world.

While AR can potentially supplement the physical environment with information perceptible by all human senses, visual and auditory overlays are currently the most commonly applied augmentations. In the case of visual AR, computer-generated graphics are spatially registered with and overlaid on real objects, using display and tracking technologies that will be discussed in more detail in Chapter 2.

Mobile and wearable computing systems provide users access to computational resources even when they are away from the static infrastructure of their offices or homes. One of the most important aspects of these devices is their potential to support *location-aware* or *location-based* computing, offering services and information that are relevant to the user's current locale (Beadle et al., 1997). Such location-based computing and location-based services open up new possibilities in the way we interact with computers, gather information, find our way in unfamiliar environments, and do business.

Research and commercial location-aware systems have explored the utility of a variety of coarse position-tracking approaches, ranging from monitoring infrared signals emitted by "active badges" (Want et al., 1992), to getting location information from the nearest wireless phone base stations, in order to provide local weather and traffic updates, or tourist information (3G-LBS, 2001)

AR, which demands far more accurate position tracking combined with precise orientation tracking, can provide an especially powerful user interface for location-aware mobile computing; one might even say, the ultimate interface: to interact directly with the electronically enriched world around us. The world becomes the interface. Section 2.3.1 will discuss various prototype applications that mobile AR has been successfully employed for. MARSs apply the AR interface concept in truly mobile settings, that is, away from the carefully conditioned environments of research laboratories and special-purpose work areas. Quite a few technologies come together in making this possible: wearable computing, mobile displays, global tracking technologies, wireless communication, and location-based computing and services.

In our mobile AR prototypes, we augment the world with tracked optical see-through stereoscopic displays. Determining position and orientation of an object is often referred to as six–degree-of-freedom (6DOF) tracking for the six parameters sensed: position in x, y, and z, and orientation in yaw, pitch, and roll angles. Outdoors, we rely on GPS-based position tracking and hybrid inertial and magnetometer-based orientation tracking for head pose. Indoors, we employ ceiling-mounted 6DOF trackers and several hybrid tracking and dead-reckoning schemes, discussed in more detail in Section 3.3.1.

It should be noted that as an alternative or complement, the MARS unit could employ camera-based vision to determine the position of important objects that are to be virtually annotated, or even infer the user's 6DOF head pose. Such a tracking scheme, when perfected, might allow for precise image-based registration of virtual and real ma-

terial in unprepared environments. The work described in this thesis does not employ such tracking technologies, whose pursuit is beyond the scope of this dissertation, but it takes their future possibility into account.

1.2 Thesis Overview

The kind of user interfaces (UIs) possible in mobile AR systems differ substantially from the Windows-Icons-Menus-Pointing (WIMP) interface metaphor commonly employed in desktop and laptop systems. While WIMP interfaces are fairly well studied and understood, there does not yet exist much knowledge about typical components of, useful services for, and user interactions possible with MARS interfaces. At the same time, the dynamic nature of MARS interaction makes it desirable to design UIs that retain their usefulness under a wide variety of user (in particular, viewing) situations. The significant challenge that mobile AR has faced until now is illustrated by many of our early prototype UIs: the complexity of the augmented views rapidly increases when many virtual objects fight for screen space to annotate physical entities in the dynamic views of fast-paced roaming users. Other complicating factors, such as unexpected interruptions by incoming messages, or varying tracking accuracy, make it even harder for a MARS UI to maintain a usable composition. In response to these issues, we undertook the following research steps:

- We explored possible UIs for MARS through a series of prototypes (Feiner et al., 1997; Höllerer et al., 1999b; Höllerer et al., 2001a) and a set of example applications and interfaces (Höllerer et al., 1999a; Butz et al., 1999; Höllerer et al., 2001b; Bell et al., 2002a; Bell et al., 2002b).

- Based on these explorations and related work in the field, we developed a taxonomic categorization of mobile AR interface components and their properties. Virtual and real world objects alike are considered part of the interface. We developed data structures for MARS UI components that reflect the properties set forth in this taxonomy.

- In order to keep the UI as simple, informative, and well-structured as possible, we introduced and developed UI-management techniques for mobile AR, focusing on the visual UI composition (Julier et al., 2000b; Höllerer et al., 2001a; Bell et al., 2001).

- In order to have the system react dynamically to newly arising contingencies, we designed and implemented a rule-based reasoning architecture that uses the taxonomic data classification mentioned above to automatically rearrange AR views in

dynamic situations; for example, to ensure the visibility and accessibility of interface components deemed important for the current task, or to adjust for variations in tracking accuracy.

1.2.1 Properties of MARS Interfaces

Mobile AR presents a way for people to interact with computers and electronic information that is radically different from static desktop computing. It also differs substantially from other styles of computer interaction, such as VR, or various kinds of wearable computing interfaces, even though it may share some traits with several of these.

One of the key characteristics of MARS is that they exist in the real world. Both virtual and physical objects are part of the UI and can influence what kind of information the computer needs to present next. This raises several issues:

Control: Unlike a stand-alone desktop UI, where the only way the user can interact with the presented environment is through a set of well defined techniques, the UI for MARS needs to take into account the unpredictability of the real world. For example, a UI technique might rely on a certain object being in the user's field of view and not occluded by other information. Neither of these properties can be guaranteed, since the user is free to look away, and other information could easily get in the way, triggered by the user's own movement or an unforeseen event (such as another user entering the scene). Thus, to be effective, the UI technique either has to relax the non-occlusion requirement, or has to somehow guarantee non-occlusion in spite of possible contingencies.

Scene dynamics: In a mobile, head-tracked UI, the scene will be much more dynamic than for stationary ones. For a MARS this is especially true, since in addition to all the dynamics due to head motion, the system has to consider potentially moving objects in the real world that might interfere with the UI presented on the head-worn display. Because of these unpredictable dynamics, the screen composition for the UI needs to be flexible and the arrangement of visual elements may need to be changed. On the other hand, traditional UI design wisdom suggests to minimize dynamic changes in the UI composition (Shneiderman, 1998).

Consistency: People have internalized most of the laws of the physical world. When using a computer, users can learn the logic of a new UI. As long as these two worlds are decoupled (as they are in the desktop setting), there are no big problems even if the approaches are far from compatible. In the case of MARS, however, we need to be very

careful to design interfaces in which the physical and virtual world are consistent with each other.

Need for embedded semantic information: In MARS, virtual material is *overlaid* on top of the real world. Thus we need to establish concrete semantic relationships between virtual and physical objects in order to characterize UI behavior. In fact, since many virtual objects are designed to annotate the real world, these virtual objects need to store information about the physical objects to which they refer (or at least have to know how to access that information).

Display space: In terms of the available display space and its best use, MARS UIs have to deal with a much more complicated task compared to traditional 2D interfaces. Instead of one area of focus (e.g., a single CRT display), we have to deal with a potentially unlimited display space surrounding the user, only a very small portion of which is visible at any point in time. The representation of that portion of augmented space depends on the user's position, head-orientation, personal preferences (e.g., filter settings) and ongoing interactions with the augmented world, among other things. Usage of this space is made even more difficult by possible constraints that other pieces of information may impose. Other virtual or physical objects may, for example, need to be visible under all circumstances, and thus place restrictions on the display space that other elements are allowed to obstruct.

The display management problem is further complicated by the possibility of taking into account multiple displays. MARS, as a non-exclusive interface to whatever computing capabilities the augmented world may offer, may seamlessly make use of other kinds of displays, ranging from wall-sized to desk-top to hand-held or palm-top. If such display devices are available and accessible to the MARS, questions arise as to which display to use for what kind of information and how to let the user know about that decision.

1.2.2 MARS UI Management

Because of the dynamic nature and display space constraints of MARS UIs, we contend that automated computer support is needed in order to react adequately and flexibly to the dynamically changing context of MARS user situations. UI management addresses this need by having the system participate in the design and layout of the user interface.

MacIntyre and Feiner (1996a) coined the term *Environment Management* to denote the general problem of controlling the positions and interactions of many virtual and physical objects in a world of multiple users, multiple displays, multiple interac-

tion devices and multiple input and output media in general, and suggested the use of semi-automated behaviors to address it.

In Mobile AR, where the augmented environment is the user interface, Environment Management means first and foremost UI Management. This thesis focuses on a subset of Environment Management, namely the design and layout of the UI. In other words, we deal with the selection of UI components and their adequate presentation to the user. Our approach is to enable the computer to reason about the components of the UI, using object properties set forth in a taxonomy of UI elements, and then control them via a rule-based engine that is tightly integrated with the rest of the AR system.

1.2.3 Scope of this Research

A complete and exhaustive treatment of MARS UIs is not a realizable goal at this point in the maturity of the field. Instead, this research aims to advance the state-of-the-art in MARS in several respects:

First, we present a collection of MARS UI components that have been tried in a series of increasingly powerful MARS hardware prototypes. Second, starting from these UI components, and taking into account related research in the field, we categorize the UI elements in a MARS object taxonomy and suggest data structures to annotate MARS objects with taxonomic metadata. Third, we present a rule-based MARS architecture that can take advantage of such metadata in order to react to dynamic changes in user context.

This work focuses on mobile AR interfaces that employ see-through head-worn displays and head-pose tracking to graphically annotate the real world. As will be demonstrated, we also consider heterogeneous display and interaction environments, in which hand-held, lap-top, and wall-mounted displays play a part, but we explore these options in combination with our main focus on MARS: systems that augment a user's vision with tracked head-worn displays.

We investigate the capabilities of MARS UIs as a foundation for future systems. Thus, we do not want to restrict ourselves unnecessarily in terms of the functionality that we can support. Instead, we make compromises regarding the form factor of our overall system. Our prototypes, one example of which is depicted in Figure 1.1, consist of experimental backpack-based systems, based on commercial hardware that we have chosen for programmability and power at the expense of comfort and wearability. Ideally the bulky backpack system would be replaced by a wearable unit at most the size of a palmtop computer, the head-worn display would look much more like normal prescription eyewear, and the GPS antenna could be embroidered into the user's clothing. While theoretically possible with current technology, these are not currently economically viable

Figure 1.1: (a) The author with MARS backpack, tracked head-worn display, and hand-held computer, watching (b) an augmented campus scene (from the Situated Documentaries II application)

options.

With the exception of a novel technique for dead-reckoning tracking for indoor MARS, described in Section 3.3.1, this research is not focused on advancing the state-of-the-art in tracking technologies for mobile AR. Instead, our approach integrates several commercial sensors in order to do open-loop 6DOF tracking of a user's head pose. Vision-based tracking techniques are considered and reviewed in Chapter 2, but are not part of the research prototypes that evolved from this work.

We explore interface possibilities such as hand tracking, tracked tablet displays, and collaborative systems involving multiple users with head-tracked AR displays, but due to the challenges of accurately tracking multiple objects reliably in 6DOF using a truly mobile system outdoors, we conducted that research indoors, making use of commercial wide-area ceiling trackers. Also, while we constructed a series of outdoor MARS prototypes, we relied on only one functional system at each point in time.

As far as the taxonomy of UI components is concerned, any taxonomy first and foremost reflects the ideas and experience of the designers. We realize that the most general taxonomies have slowly evolved through a consensus process that considers input from multiple involved parties. The goal of the work presented here is not to provide an all-encompassing general definition of MARS UIs, but instead provide a working basis for the exploration of adaptive UIs that construct augmented scenes specifically tailored to user context. The main focus is on visual AR, as portrayed through tracked head-worn displays. In our taxonomy, we include certain input and output technologies, such as glove-based input devices and spatially registered audio, which have not actually been

implemented in any of our own MARS prototypes to date, but which have been discussed in the literature.

1.2.4 Summary of Contributions

This research makes a number of contributions to the fields of mobile AR, 3D interaction, and HCI:

- We present the system design of a series of MARSs, each of which demonstrates state-of-the-art hardware and software capabilities for its time. Our discussion starts with extensions to the Coterie-based Columbia Touring Machine, which was the first outdoor MARS, and leads up to the adaptive UI architecture Ruby, which is our most powerful and flexible MARS thus far. Over the described range of systems, we distinguish three major hardware platforms with increasing computational, graphical, and interaction capabilities (cf. Section 3.1 and Appendix A), and five different software environments that we implemented (Section 3.2), exploiting and managing these capabilities to the best possible extent. For our indoor MARS work, we describe a novel method for coarsely tracking a mobile person along corridors and pathways in an office building, using only worn sensors and knowledge of the building infrastructure (Höllerer et al., 2001b).

- We present and explore in detail the concept and application framework of Situated Documentaries, which we implemented on top of Coterie-based and Java-based software infrastructures. Situated documentaries form a spatially distributed hypermedia system, which lets users experience an interconnected web of multimedia news stories, presenting events in the exact places that are relevant to the story (Höllerer et al., 1999a).

  In other application prototypes, we demonstrate the interoperability of our Coterie- and Java-based MARS infrastructures with networked stationary UIs, including desktop and immersive indoor AR and VR interfaces and hand-held computers. We explore indoor-outdoor collaboration between roaming users and remote experts who see an overview visualization of the whole work area. We also investigate MARS UIs for navigational guidance (Höllerer et al., 1999b; Höllerer et al., 2001b).

- Based on these and additional UI explorations and also on related work in the field, we create a practical taxonomy of MARS UI components, their properties, and interaction concepts. We define this taxonomy with the idea of formalization and implementation in mind. We implement a significant subset of this taxonomy as part of the rule-based MARS *Ruby*.

- We define and explore the necessary steps for a UI management pipeline for MARS: *information filtering*, *UI component design*, and *view management*. In joint work with Blaine Bell, we designed interactive view-management techniques for AR interfaces. (Höllerer et al., 2001a).

- We present the design and implementation of *Ruby*, a rule-based MARS that stores all MARS objects in an internal knowledge base, tagged with the classification information outlined in our taxonomy. A forward-chaining expert system algorithm keeps the system's knowledge about all UI components up-to-date, and allows it to dynamically react to unforeseen changes in context, user behavior and availability of resources.

1.2.4.1 System Designs for Outdoor MARS

The Columbia *Touring Machine* (Feiner et al., 1997) was the first example of an outdoor MARS with head-tracked graphical overlays and is the starting point of the system designs discussed in this dissertation. The author of this dissertation assisted in the design of the prototype and subsequently became the lead architect of several MARS, for both hardware and software design.

Hardware Prototypes. Three major prototype hardware systems, following the original 1996–1997 *Touring Machine*, form the basis of our UI explorations, as described in more detail in Section 3.1 and Appendix A: MARS 1999, with new position- and orientation tracking, display, and handheld computing technologies, and an upgrade of the core processing power; MARS 2000, with a wearable computer assembled from electronic boards and components and new interaction and display hardware; and MARS 2001/2002, which is based on a 3D-graphics–capable laptop computer mounted with other components on a flat backpack board.

Software Infrastructures. Building on the experiences gathered with the Touring Machine, we have created a series of software infrastructures to further explore MARS UIs:

- The *Situated Documentaries Extensions* infrastructure, which extends the Touring Machine system with world-stabilized 3D graphics, multimedia capabilities, and better handheld computer integration (Höllerer et al., 1999a).

- *IN-N-OUT AR*, an indoor-outdoor environment that integrates several interfaces of different styles and platforms into a collaborative MARS (Höllerer et al., 1999b).

- *JABAR* (JAva-Based AR), a complete MARS framework implemented entirely in Java, which was our infrastructure of choice for the *Situated Documentaries III* prototype.

- In joint work with Blaine Bell, *CGUILab AR*, which unified various Java-based development efforts in our research lab. This was the infrastructure predominantly used for our *view management* work, which presents techniques to control the layout of MARS UI components on the view plane (Höllerer et al., 2001a; Bell et al., 2001).

- *Ruby*, a rule-based software architecture featuring a central knowledge base, implemented with the Java Expert System Shell (JESS), tightly integrated with the rest of the AR system (see Section 6.1 and Chapter 6).

In joint work with Drexel Hallaway and Navdeep Tinna, we created a novel method to track a MARS indoors, using a pedometer, orientation tracker, and knowledge of the environment in the form of spatial maps and accessibility graphs (Höllerer et al., 2001b). Section 3.3 presents the approach. We combine this tracking method with the much more accurate but limited-range 6DOF tracking of a commercial ceiling-mounted tracker, and design interfaces that adapt to the change in tracking accuracy (cf. Sections 5.3.1 and 6.2.3).

1.2.4.2 Situated Documentaries and Other Application Prototypes

Using these hardware and software infrastructures, we explore MARS UIs in a series of application examples. We designed mobile AR interfaces for such diverse application areas as a campus tour guide, historic architecture presentations, hidden infrastructure visualizations, indoor and outdoor navigational guidance systems, augmented group meetings, interactive documentaries, and a neighborhood restaurant guide. These were developed together with Blaine Bell, Andreas Butz, Drexel Hallaway, Gus Rashid, Tachio Terauchi, and Ryuji Yamamoto, with modeling and testing support from Elias Gagas, Simon Lok, Sinem Güven, Tiantian Zhou, Hrvoje Benko, and several groups of students enrolled in a series of classes on new media technologies.

The application framework we will focus on the most is a sequence of applications, dubbed *Situated Documentaries* (Höllerer et al., 1999a). Situated documentaries are very well suited for the purpose of exploring general MARS UI issues, since they provide a hypermedia interface for the physical world, enabling a wide variety of interaction mechanisms and presentations on arbitrary topics. We present three different implementations of this application concept, through which users can experience an interconnected web of multimedia stories, presenting documentaries in the physical places that are relevant to the respective story.

We present an overview of the MARS UIs for the different application prototypes mentioned above, focusing on UI support for collaboration and navigation tasks (Höllerer et al., 1999b; Höllerer et al., 2001b). We describe how successively more complex and

dynamic MARS interfaces lead to the need for managing the complexity of the UIs. We comment on the design process of the above MARS UIs, discuss the most serious problems we encountered, and describe lessons learned during our design cycle (Section 5.4).

1.2.4.3 Taxonomy of MARS UI Components

By way of all these application prototypes, we explore a range of MARS user interface elements, leading us to the definition of a taxonomy of MARS UI components. Related work by other researchers is also considered in our categorization. The resulting taxonomy can be used to compare and contrast UIs and explain how UIs achieve the usability they are created for. It provides a framework to evaluate and describe existing interfaces, and to stimulate creation of new interfaces. Last but not least, such a framework, if suitably formalized, allows a computer access to the building blocks of the MARS UIs and to make informed decisions about the UI (see Section 1.2.4.5).

1.2.4.4 MARS UI Management

Throughout our explorations, we focus on one of the most challenging conceptual problems of MARS UIs: the ever-increasing complexity of the augmented views in a dynamic world of many virtual objects competing for screen estate while annotating physical entities in the dynamic views of fast-paced roaming users. We propose the concept of MARS UI management to remedy this situation. Three stages of UI management help to remove complexity from the MARS UI. Information filtering, to focus on the relevant information; UI component design, to choose the right UI components for the user's context; and view management, to optimize layout of these components on the view plane (Höllerer et al., 2001a).

1.2.4.5 Rule-based AR System

We develop data structures for MARS UI components that reflect the entities and properties set forth in our taxonomy. We tag each component with information about its purpose, its intrinsic properties, its relationship to other objects and its capabilities and flexibility with regard to various manipulations. The purpose of this is to allow a MARS to make informed decisions about its user interface. We present Ruby, a rule-based system architecture that uses the meta-information stored with the MARS UI components to make dynamic UI design decisions, based on the user's context. Example applications illustrate how the system can adapt to changes in user activity or tracking accuracy, based on what the system knows about the current UI layout, and hence, the user situation.

The rest of this dissertation is organized as follows: In Chapter 2 we present a detailed overview of mobile augmented reality systems. We discuss the history of the field, review applications that have been pursued, and discuss system components, such as computing platforms, displays, tracking systems, and interaction technologies. We also touch on larger systems issues, such as global communication and data storage infrastructures, and the need for environmental modeling. We conclude that chapter with a review of existing MARS prototypes. Chapter 3 describes our various MARS designs, covering both hardware and software architectures that we created for implementing and testing mobile AR interfaces. Chapter 4 establishes our analytical foundation for MARS UIs. We discuss user interface concepts for MARS and introduce our taxonomy of MARS UI components. In Chapter 5, we report on different prototype applications that we developed, forming a growing set of MARS UI components, which in turn influences our taxonomy. The interfaces become increasingly dynamic and complex, leading to the need of automatic UI Management. We introduce the idea of a MARS UI-management pipeline, featuring the steps of information filtering, UI component design, and view management. In Chapter 6 we present a knowledge-based system architecture, which formalizes part of the taxonomy from Chapter 4. We explain how this rule-based system can dynamically control the UI design process in a MARS. We show several examples of the MARS UI being adapted to dynamic changes in user context. Chapter 7 finally presents our conclusions and discusses future work.

Chapter 2

Background and Related Work

This chapter provides a detailed introduction to mobile AR technology, covering central topics, such as wearable display and computing hardware, tracking, registration, and user interaction. We begin by pointing out the main components of a MARS in Section 2.1, followed by a brief account of the history of mobile AR in Section 2.2. Section 2.3 discusses the potential and possibilities of MARS technology, with a detailed overview of prototype application areas, and reviews the challenges that impede immediate widespread commercial adoption. In Section 2.4, we take a closer look at the requirements and specific components of MARS, one at a time. Section 2.5 presents an overview of existing MARS prototypes. The specific testbed systems we built to enable the UI explorations presented in this thesis are described in Chapter 3.

2.1 Components of Mobile AR Systems

Several components are needed to create a mobile AR experience. To begin with, one needs a *computational platform* that can generate and manage the virtual material to be layered on top of the physical environment, process the tracker information, and control the AR display(s).

Next, one needs *displays* to present the virtual material in the context of the physical world. In the case of augmenting vision, these can be head-worn displays, mobile hand-held displays, or displays integrated into the physical world (e.g., flat-panel displays built into walls and furniture, or video projection on arbitrary surfaces). All these different display types might also be used concurrently. Other senses (e.g., hearing, touch, or smell) can also be potentially augmented. Spatialized audio in particular is often used to convey localized information, either complementing or completely substituting for visual elements (Sawhney and Schmandt, 1998).

We must also address *Registration*: aligning the virtual elements with the physical

objects they annotate. For visual and auditory registration, this can be done by *tracking* the position and orientation of the user's head and relating that measurement to a model of the environment and/or by making the computer "see" and potentially interpret the environment by means of cameras and computer vision.

Wearable input and interaction technologies enable a mobile person to work with the augmented world (e.g., to make selections or access and visualize databases containing relevant material) and to further augment the world around them. They also make it possible for an individual to communicate and collaborate with other MARS users.

Wireless networking is needed to communicate with other people and computers while on the run. Dynamic and flexible mobile AR will rely on up-to-the-second information that cannot possibly be stored on the computing device before the fact.

This brings us to another item in the list of requirements for MARS: *data storage and access technology*. If a mobile AR system is to provide information about a roaming individual's current environment, it needs to get the data about that environment from somewhere. Data repositories must provide information suited for the roaming individual's current context. Data and service discovery, management, and access all pose several research questions that are being examined by researchers in the database, middleware, and context-based services communities. From the user's point of view, the important questions are how to get to the most relevant information with the least effort and how to minimize information overload.

2.2 Historical Overview

The first fully functional AR system dates back to the late 1960s, when Ivan Sutherland and his colleagues built a mechanically tracked 3D see-through head-worn display, through which the wearer could see computer-generated information mixed with physical objects, such as signs on a laboratory wall (Sutherland, 1968). For the next few decades much research was done on getting computers to generate graphical information, and the emerging field of *interactive computer graphics* began to flourish. Photorealistic computer generated images became an area of research in the late 1970s, and progress in tracking technology furthered the hopes to create the ultimate simulation machine. The field of VR began to emerge. Science fiction literature, in particular the early 1980s movement of *cyberpunk*, created visions of man-computer symbiosis, originally contemplated by Licklider (1960). The entertainment industry jumped in with movies such as the *Terminator* series, which presented one version of what it could look like when a computer annotates your vision. During the 1970s and 80s, AR was a research topic at just a few institutions, including the U.S. Air Force's Armstrong Laboratory, the NASA Ames Research Center, the Massachusetts Institute of Technology, and the Uni-

versity of North Carolina at Chapel Hill. As part of the US Air Force *Super Cockpit* project, Tom Furness developed a high-resolution heads-up overlay display for fighter pilots, supported by 3D sound (Furness, 1986).

It was not until the early 1990s, with research at the Boeing Corporation that the notion of overlaying computer graphics on top of the real world received its current name. Tom Caudell and David Mizell at Boeing worked on simplifying the process of conveying wiring instructions for aircraft assembly to construction workers, and they referred to their proposed solution of overlaying computer presented material on top of the real world as *augmented reality* (Caudell and Mizell, 1992). Even though this application was conceived with the goal of mobility in mind, true mobile graphical augmented reality was out of reach for the available technology until a few years later. Also during the early 1990s, Jack Loomis and colleagues at the University of California, Santa Barbara, developed a GPS-based outdoor system, presenting navigational assistance to the visually impaired with spatial audio overlays (Loomis et al., 1993).

Since about the mid-1990s computing and tracking devices have become sufficiently powerful, and at the same time small enough, to support registered computer-generated graphical overlays in a dynamic mobile setting. The Columbia *Touring Machine* (Feiner et al., 1997), developed in 1996, is an early prototype of an outdoor mobile augmented reality system that presents 3D graphical tour guide information to campus visitors, registered with the buildings and artifacts the visitor sees.

At about the same time AR research experienced a renaissance, around 1990, Mark Weiser and fellow researchers at Xerox PARC conceptualized the idea of *ubiquitous computing* (Weiser, 1991), an environment in which computing technology is embedded into all kinds of everyday physical objects, (such as appliances, doors, windows, or desks) which results in the "computer disappearing into the background." In other words, computational devices and user interfaces blend into the environment and are no longer a primary factor in the user's consciousness. Related to these ideas is the concept of *tangible user interfaces*, which attempts to let people control digital information by handling seemingly non-electronic physical objects. Ishii and colleagues at the MIT media lab prototyped numerous tangible interfaces since the early 1990s, using abstract components (wooden blocks, plastic triangles), concrete everyday objects (bottles, dolls, plexiglas models), and sculpting material (clay, sand) (Ishii and Ullmer, 1997).

Another research field, *wearable computing* (Mann, 1997; Starner et al., 1997a), took off in the 1990s, when personal computers were becoming small enough to be carried or worn at all times. The earliest wearable system was a special purpose analog computer for predicting the outcome of gambling events, built in 1961 (Thorp, 1998). On the commercial front, palmtop computers embody the trend towards miniaturization. They date back to the Psion I organizer from 1984 and later became commonplace after the in-

troduction of the Apple Newton MessagePad (1993) and the Palm Pilot (1996). Since the mid 1990s, wearable computing has received ever increasing commercial backing, and the miniaturization and more cost-effective production of mobile computing equipment have resulted in several companies offering commercial wearable computing products (E.g., Xybernaut, Charmed, Via, Antelope Technologies).

In terms of the technologies necessary for a mobile AR experience, we will look briefly at the historical developments in the fields of *tracking and registration*, *wireless networking*, *display technology*, and *interaction technology* in Section 2.4. Now that the technological cornerstones of mobile AR have been placed, it might seem that it is purely a matter of improving the necessary components, putting it all together, and making the end result as reliable as possible. However, there are more challenges lying ahead, and, after discussing various applications of MARS in the following subsection, we will come back to these challenges in Section 2.3.2.

2.3 Mobile AR: Applications, and Challenges

Mobile AR would be particularly applicable whenever people require informational support for a task while needing to stay focused on that task. It could allow people to interact with computer-supported information (which might come from databases or as a live feed from a remote expert), without getting distracted from the real world around them. This is a very important feature for the mobile worker, or for anybody who needs or wants to use their hands, and some of their attention, for something other than controlling a computer. The next section gives a few examples of such occupations and summarizes application areas for which mobile AR prototypes have been employed.

2.3.1 Applications

Many of the early AR publications are "application papers," describing potential applications of the new technology in varied fields. In the rest of this section, we give an overview of such potential uses for mobile AR systems.

2.3.1.1 Assembly and Construction

Over a nine year period, researchers at Boeing built several iterations of prototypes for AR-supported assembly of electrical wire bundles for aircraft. AR can overlay schematic diagrams, as well as accompanying documentation, directly onto the wooden boards on which the cables are routed, bundled, and sleeved. The computer can lead (and potentially talk) assembly workers through the wiring process. Since the resulting wire bundles are long enough to extend through considerable portions of an aircraft, stationary AR

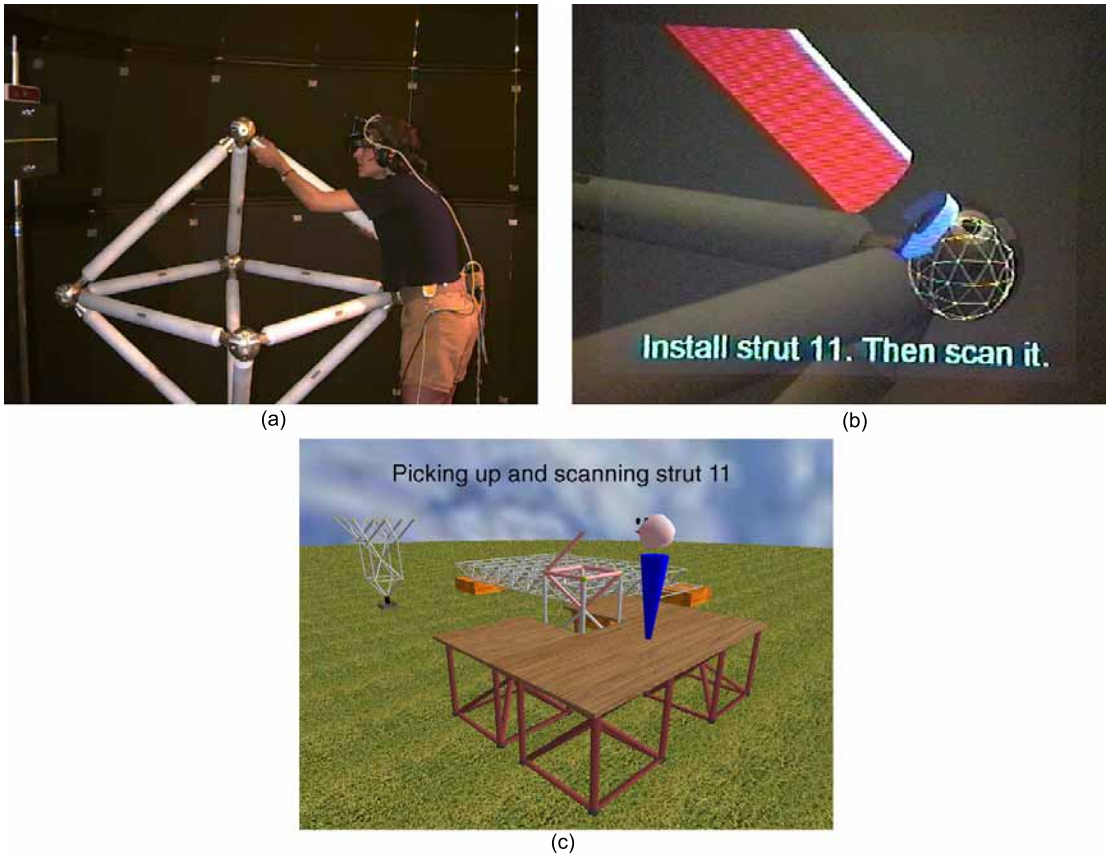


Figure 2.1: Columbia University project on AR for construction. (a) User installing a spaceframe strut. (b) View through the head-worn display. (c) Overview visualization of the tracked scene.

solutions were not sufficient, and the project became an exercise in making mobile AR work for a specific application scenario (Mizell, 2001). *Augmented Reality for Construction* (Feiner et al., 1999), is a prototype for the construction of spaceframe structures. As illustrated in Figure 2.1, the user would see and hear through their head-worn display where the next structural element is to be installed. The user scans the designated element with a position-tracked barcode reader before and after installation to verify that the right piece gets installed in the right place. The possibility of generating virtual overview renderings of the entire construction scene, as indicated in a small-scale example via live-updated networked graphics in one of the prototype's demonstrations (see Figure 2.1a), is a side benefit of tracking each individual worker and following their actions, and could prove immensely useful for the management of large complex construction jobs. Such construction tasks would ultimately take place in the outdoors.

2.3.1.2 Maintenance and Inspection

Apart from assembly and construction, inspection and maintenance are other areas in manufacturing that may benefit greatly from applying mobile AR technologies. Sato et al. (1999) propose a prototype AR system for inspection of electronic parts within the boundaries of a wide-area manufacturing plant. Their mobile AR backpack is tracked with a purely inertia-based (gyroscope orientation tracker plus acceleration sensor) tracking system, and calibration has to be frequently adjusted by hand. Zhang et al. (2001) suggest the use of visual coded markers for large industrial environments. Klinker et al. (2001) present the system architecture of a MARS prototype for use in nuclear power plant maintenance. AR is well suited for situations that would benefit from some kind of "x-ray vision," an ability to see through solid structures. Using direct overlays of hidden infrastructure, AR can assist maintenance workers who are trying to locate a broken cable connection within the walls of a building, or the location of a leaking pipe beneath a road's surface. The exact position may either have been detected (e.g., by installed sensors) before the maintenance worker arrives, in which case direct visualization of the problem area via AR could be the fastest way to direct the worker's attention to the right area, or AR may be used as a supporting tool for actually determining the problem, by instantaneously and directly visualizing any data the worker might gather, probing the environment with various sensors.

2.3.1.3 Navigation and Path Finding

Consider some other outdoor-oriented uses for mobile AR. An important application area for wearable systems is their use as navigational aids. Wearable computers can greatly assist blind users (Loomis et al., 1993; Petrie et al., 1996; Loomis et al., 1998) through audio and tactile feedback. If auditory information relating to real world waypoints and features of the environment is presented to the position-tracked wearer of the system via spatialized stereo audio, this clearly matches our definition of a MARS from Section 1.1.

Visually augmented reality can aid navigation by directly pointing out locations in the user's field of view, by means of directional annotations, such as arrows and trails to follow (see Figure 2.2), or by pointing out occluded infrastructure, either directly by visually encoded overlays (Furmanski et al., 2002), or indirectly through 2D or 3D maps that are dynamically tailored to the situation's needs and presented in the person's vision (see Figures 2.9 and 5.17).

2.3.1.4 Tourism

Taking navigational interfaces one step further, including more and more information about objects in a mobile user's environment that might be of interest to a traveler, leads

Figure 2.2: Navigational AR interfaces, imaged through head-worn displays. (a) Indoor guidance using overview visualization and arrows. (b) Virtual trails and flags outdoors (viewed from the roof of a building).

naturally into applications for tourism (Feiner et al., 1997; Cheverst et al., 2000). In this case, AR is not only used to find a certain destination, but also to display a lot of background information, helping the visitor to make decisions and get informed. For example, instead of looking up a description and historical account of a cathedral in a guide book (or on a laptop computer in the hotel room, or even on a wirelessly connected palm-sized computer at the site), AR can bring the actual air around the church alive with information: three-dimensional models of related pieces of art or architecture, the life and work of the architect, or architectural changes over the centuries can be documented in situ with overlays. The possibilities are limited only by the amount and type of information available to the AR-enabled individual and the capabilities of the AR device the individual is wearing.

Figure 2.3 shows an example of a mobile AR restaurant guide developed at Columbia University. This prototype MARS provides an interface to a database of restaurants in the Morningside Heights area. Information about restaurants is provided either through an overview 3D map, so that the user can be guided to a specific place of his or her choice, or as direct annotations at the actual restaurant locations themselves. Having selected an establishment, the user can bring up a popup window with further information on it: a brief description, address and phone number, an image of the interior and, accessible at a mouse click, the menu and, if available, reviews of the restaurant, and its external web page.

2.3.1.5 Architecture and Archaeology

AR is also especially useful to visualize the invisible: architects' designs of bridges or buildings that are about to be constructed on a particular site, historic buildings, long

Figure 2.3: Mobile AR restaurant guide. (a) User with MARS backpack, looking at a restaurant. (b) Annotated view of restaurant, imaged through the head-worn display.

torn down, in their original location, or reconstructions of archaeological sites. Figure 2.5, which was imaged through AR eyewear showing a situated documentary (Höllerer et al., 1999a) of the history of Columbia's campus, shows a model of the Bloomingdale Asylum, which once occupied the grounds of today's main administrative building. The European sponsored project ARCHEOGUIDE (Augmented Reality based Cultural Heritage On-site Guide (Vlahakis et al., 2002)) aims to reconstruct a cultural heritage site in an augmented reality and let visitors view and learn about the ancient architecture and customs.

2.3.1.6 Urban Modeling

AR is not solely used for passive viewing or information retrieval via the occasional mouse-button (or shirt-button) click. Many researchers are exploring how AR technologies could be used to enter information to the computer (Rekimoto et al., 1998). One practical example is 3D modeling of outdoor scenes: using the mobile platform to create 3D renderings of buildings and other objects that model the very environment to be used later as a backdrop for AR presentations (Baillot et al., 2001; Piekarski and Thomas, 2001b).

2.3.1.7 Geographical Field Work

Field workers in geography and regional sciences could use AR techniques to collect, compare, and update survey data and statistics in the field (Nusser et al., 2001). By

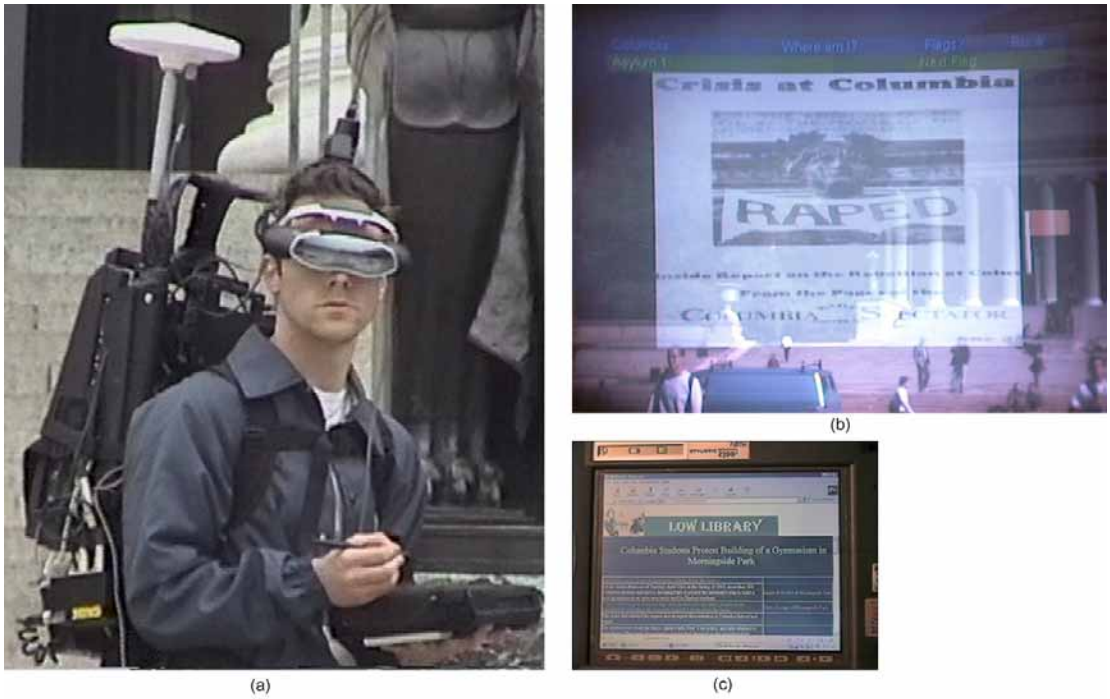


Figure 2.4: Situated Documentaries I. (a) User with backpack MARS. (b) View through head-worn display, showing multimedia story about Columbia 1968 student revolt. (c) Additional information on complementary hand-held display.

assisting data collection and display, an AR system could enable discovery of patterns in the field, not just the laboratory. Instant verification and comparison of information with data on file would be possible.

2.3.1.8 Journalism

Journalism is another area where mobile AR techniques might prove beneficial. Pavlik (2001) discusses the use of wireless technology for the mobile journalist, who covers and documents a developing news story on the run. AR techniques could be used to leave notes in the scene for other collaborating journalists and photographers to view and act upon. The *Situated Documentaries* project at Columbia University (Höllerer et al., 1999a), illustrated in Figures 2.4 and 2.5, is a collaboration between computer science and journalism, and uses a MARS device for storytelling and presentation of historical information.

Figure 2.5: Situated Documentaries II: Historic building overlaid on its original location on Columbia's campus.

2.3.1.9 Entertainment

The situated documentaries application also suggests the technology's potential for entertainment purposes. Instead of delivering 3D movie "rides," such as the popular *Terminator-2* presentation at Universal Studios, to audiences in special purpose theme park theatres, virtual actors in special effects scenes could one day populate the very streets of the theme parks, engaging AR outfitted guests in spectacular action. As an early start in this direction, several researchers have experimented with applying mobile AR technology to gaming (Thomas et al., 2000; Szalavari et al., 1998; Starner et al., 2000).

2.3.1.10 Medicine

Augmented reality has important application possibilities in medicine. Many of these, such as surgery support systems that assist surgeons in their operations via live overlays (Fuchs et al., 1998), require very precise registration, but do not require that the surgeon be extremely mobile while supported by the AR system. There are, however, several possible applications of mobile AR in the medical field. In the hospital or nursing home, doctors or nurses on their rounds of visits to the patients could get important information about each patient's status directly delivered to their glasses (Hasvold, 2002). Out in the field of emergency medicine, paramedics could assess a situation quicker with wearable sensing and AR technology: They would apply the wearable sensors to the patient and would from then on be able to check the patient's status through AR eyewear, literally at a glance. Also, a remote expert at a distant hospital can be brought into the loop and communicate with the field worker through the AR system, seeing through camera feeds what the field worker is seeing, which could help in preparing for an imminent operation

at the hospital.

Monitoring the health information of a group of people at the same time can be advantageous for trainers or coaches during athletic training or competition. The coaches would be able to get an overview of the athletes' health conditions while the training program is under way, with the possibility of making individual adjustments to the program. Note, however, that providing AR overlays onto a moving person would require that the target person is position-tracked with high accuracy. The faster the movements of the monitored person, the faster the update rate of the position sensing has to be. For a long time, commercial differential GPS receivers provided update rates of up to 5Hz, which is sub-optimal for the described application. Newer products provide update rates of up to 20 Hz (Trimble Navigation Ltd, 2002). Obviously, miniaturization of all sensing devices is also a key requirement. The military also has potential medical uses for mobile AR technologies. The health status of soldiers on the battlefield could be monitored, so that in case of any injuries the commanding officer can be informed of location and status of the wounded.

2.3.1.11 Military Training and Combat

Military research led to the development of satellite navigation systems and heads-up displays for war-fighter pilots. Military research laboratories have also been exploring the potential of mobile AR technology for land warriors for some time now (Tappert et al., 2001). There is considerable overlap with potential civilian applications on a general level. Navigational support, enhancement of communications, repair and maintenance, and emergency medicine, are all important topics in civilian and military life. There are, however, specific benefits that AR technology can bring to the military user. Most missions take place in unfamiliar territories. Map views, projected naturally into a warrior's limited view of the battle scene in front of him, can give him additional information about terrain that cannot easily be seen. Furthermore, reconnaissance data and mission planning information can be integrated into these information displays, clarifying the situation and outlining specific sub-missions for individual troops. Ongoing research at the Naval Research Laboratory is concerned with how such information displays can be delivered to warriors most effectively (Julier et al., 2000b). Obviously, all military communication during a military operation, especially reports about the location of friendly troops, need to be handled with utmost security when relaying them to a command center or to officers in the field. Apart from its use in combat, mobile AR might also prove a valuable military tool for training and simulation purposes. For example, several large scale combat scenarios could be tested with simulated enemy action in real training environments.

2.3.1.12 Personal Information Management and Marketing

It is difficult to predict which endeavors in mobile AR might eventually lead to commercial success stories that pave the way for widespread sale of integrated AR systems. Some small companies already offer specialized AR solutions (e.g., TriSense). Whatever might trigger widespread use, the biggest potential market for this technology could prove to be personal wearable computing. AR can serve as an advanced and immediate UI for wearable computing. In personal, daily use, AR could support and integrate common tasks, such as email and phone communication with location-aware overlays, provide navigational guidance, enable individuals to store personal information coupled with specific locations, and provide a unified control interface for all kinds of appliances in the home (Feiner, 2002). Of course, such a personal platform would be very attractive for direct marketing agencies. Stores could offer virtual discount coupons to bypassing pedestrians. Virtual billboards could advertise products based on the individual's profile. Virtual 3D product prototypes could be sent to the customer to pop up in their eyewear (Zhang et al., 2000). To protect the individual from spam and other unwanted information overload, an AR platform would incorporate appropriate filtering and view-management mechanisms (see Section 5.4.3).

2.3.2 Challenges

In spite of the great potential of mobile AR for many application areas, progress in the field has so far almost exclusively been demonstrated in a growing number of research prototypes. The time is not quite ripe yet for commercialization. When asking for the reasons why, one has to take a good look at the dimension of the task. While increasingly better solutions to the technical challenges of wearable computing are being introduced in new systems, a few problem areas remain, such as miniaturization of input/output technology and power supply, and improved thermal dissipation, especially in small high-performance systems. Ruggedness is required. With some early wearable systems the phrase 'wear and tear' seemed to rather fittingly indicate the dire consequences of usage. On top of these standard wearable computing requirements, mobile AR imposes a lot of additional needs, including: reliable and ubiquitous wide-area position tracking; accurate and self-calibrating (head-) orientation tracking; extremely light, bright, and transparent display eyewear with wide field of view; and fast 3D graphics capabilities. We will look at current technologies addressing these problem areas in the following section.

Outdoor AR is particularly challenging, since there is a wide range of operating conditions the system must handle. Moreover, in contrast to controlled environments indoors, one has little influence over outdoor conditions: for example, lighting can range from direct sunlight, possibly exacerbated by a reflective environment (e.g., snow), to ab-

solute darkness without artificial light sources during the night. Outdoor systems should withstand all possible weather conditions, including wind, rain, frost, and heat.

The list of technological challenges does not end with the user's side. Depending on the tracking technology, AR systems need to have access to a model of the environment they are supposed to annotate, or they require that such environments be prepared (e.g., equipped with visual markers or electronic tags). Vision-based tracking in unprepared environments is currently not a viable general solution, but research in this field is building up an arsenal of solutions for future systems. The data to be presented in AR overlays needs to be paired with locations in the environment. A standard access method needs to be in place for retrieving such data from databases responsible for the area the mobile AR user is currently passing through. This requires mechanisms such as automatic service detection and the definition of standard exchange formats that both the database servers and the mobile AR software support. It is clear from the history of protocol standards, that without sufficient demand and money-making opportunities on the horizon, progress on these fronts can be expected to be slow. On the other hand, the World Wide Web, HTML and HTTP evolved from similar starting conditions. Some researchers predict that location-aware computing on a global scale will be the legitimate successor of the World Wide Web as we know it today (Spohrer, 1997; Spohrer, 1999).

In the movie industry, special effects seamlessly merge computer-generated worlds with real scenes. Currently these efforts take days and months of rendering time and very carefully manually assisted integration of the virtual material into real-world footage. In AR, not only does the rendering need to be performed in real time, but also the decisions about what to display and the generation of this material must be triggered and controlled on the fly. Making the visuals as informative as possible, and, in some cases, as realistic as possible, with the correct lighting as compared to the physical environment to provide a seamless experience, is an open-ended challenge for visual AR.

Finally, as we discussed in Chapter 1, even if all technological problems were to be solved, there are conceptual difficulties associated with a UI that places a virtual layer of information on top of the real world, wherever you go. The crucial difference between such a UI and proven existing interfaces, such as the WIMP metaphor for desktop computing, or form-based command-line interfaces for travel agency software, is that here the interface has to adapt to the real world. The interface designer does not have control over, and cannot possibly predict, what situations the users will be in when they access the provided functionality. The interface behavior should be dynamic and flexible enough to adjust to and correctly take into account the user's location, motion, viewing direction, social situation, and any outside factors such as intermittent noise, objects obscuring what the user focuses on, and so on. This is a very difficult and complicated task. In this thesis, we take small steps in the direction of such a flexible dynamic UI.

2.4 Mobile Augmented Reality Systems

In this section, we review in greater depth the basic components and infrastructure required for mobile AR systems, as outlined before in Section 2.1. We take a look at mobile computing platforms, displays for mobile AR, tracking and registration issues, environmental modeling, wearable input and interaction techniques, wireless communication, and distributed data storage and access. We will give brief overviews of important historical developments in these areas, and point out technologies that have successfully been employed in mobile AR system prototypes, or that have great potential to be employed in future systems.

2.4.1 Mobile Computing Platforms

Mobile computing platforms have seen immense progress in miniaturization and performance over recent years, and are sold for increasingly lower prices. Today, high-end notebook computers catch up in computing power with available desktop solutions very shortly after a new processor model hits the market. The trend towards more mobility is clearly visible. The wearable technology market, even though still in its infancy, has a growing customer base in industry, government, and military. Wearable computing solutions for personal use can now be purchased from various sources.

There are several decision factors when choosing a computing platform for mobile AR, including the computing power needed, the form factor and ruggedness of the overall system, power consumption, the graphics and multimedia capabilities, availability of expansion and interface ports, available memory and storage space, upgradeability of components, the operating system and software development environments that run on the device, availability of technical support, and last but not least, price. Quite clearly, many of these are interdependent factors. The smaller the computing device, the less likely it is to have the highest computing power and graphics and multimedia capabilities. Expansion and interface ports normally come at the price of an increased form factor. So does upgradeability of components: if you have shrunk functionality, such as the graphics capabilities, to the level of special-purpose integrated circuits, you no longer have the luxury of being able to easily replace that component with a newer model. Additionally, it takes a lot of effort and ingenuity to shrink down any kind of technology to a size considerably smaller than what the competition is offering, so one can expect such equipment to be sold at a higher price. A concrete example of some of the tradeoffs involved in choosing a mobile AR platform is the Columbia University MARS project (Höllerer et al., 1999b). The hardware platform for this project, illustrated in Figures 2.3 and 2.4, was assembled from off-the-shelf components for maximum performance, upgradeability, ease of maintenance, and software development. These choices were

made at the cost of the form factor and weight of the prototype system, whose parts were assembled on a backpack frame. From 1996 to 2002, every single component of the prototype was upgraded multiple times to a more powerful or otherwise more advantageous version, something that would not have been possible if a smaller, more integrated system had been chosen initially. Details of our series of system prototypes over the years can be found in Chapter 3.

The smallest wearable computing platforms currently available (Windows CE-based Xybernaut Poma, or the higher performance, but pre-production OQO, Tiqit eighty-three, and Antelope Technologies mobile core computer) do not provide any hardware-accelerated 3D graphics and their main processors do not have excessive power to spare for software renderings of complex 3D scenes at interactive speeds. Decision factors in choosing a 3D graphics platform for mobile AR include the *graphics performance* required, video and texture memory, graphics library support (OpenGL or Direct-X), availability of stereo drivers, power consumption, and price. The most practical solution for a mobile AR system that can support complex 3D interactive graphics comes in the form of small notebook computers with integrated 3D graphics chips. The display can be disassembled if the computer is exclusively used with a near-eye display. However, it can be put to good use in prototype systems for debugging purposes and for providing a view for onlookers during technology demonstrations.

Specific application requirements can drastically limit the choices for a mobile AR computing platform. For example, there are currently no integrated wearable computing solutions available that support rendering and display of complex graphical scenes in stereo. A system designer targeting such applications either has to assemble their own hardware to create a small form-factor solution, or resort to the smallest available power notebook that exhibits sufficient graphics performance and a graphics chip supporting stereo.

Mobile systems do not necessarily have to follow the pattern of one standalone device, carried or worn, that generates and presents all the information to the user. Instead, there can be varying degrees of "environment participation," making use of resources that are not necessarily located on the user's body. There exists a spectrum of mobile computation, with the differentiating factor being how much computing and sensing power the mobile person is wearing or carrying, and how much of it is provided by the environment.

At one extreme, all information is generated and displayed on a single device that the user wears or carries. Examples include portable audio players and hand-held organizers without wireless communication option. Departing one step from this approach, functionality can be distributed over multiple devices. Wireless connectivity technologies, such as IEEE 802.11b or Bluetooth enable data exchange between different devices. For example, a personal organizer or wearable computer can send data over a wireless

connection to an audio/video headset. With the addition of a Bluetooth-enabled cell phone for global communication purposes such a combination would constitute a complete wireless mobile computing solution.

Not all devices need to be carried by the user at all times. Let us assume that we want to minimize the wearable computing equipment's size and power consumption. Lacking the computational power to generate and process the information that is to be displayed, the wearable device turns into a *thin client*, relying on outside servers to collect and process information and feed it to the portable device as a data stream that can be comfortably presented with the limited resources available. Displays can reside with the mobile person, as in the case of head-worn, body-worn, or hand-held displays, but they can also be embedded into the environment (e.g., loudspeakers at every street corner, or video walls).

In the extreme case, a mobile user would not need to wear or carry *any* equipment and still be able to experience mobile AR. All the computation and sensing could happen in the environment: A grid of cameras could be set up so that multiple cameras would cover any possible location the person could occupy. Information could be stored, collected, processed, and generated on a network of compute servers that would not need to be in view, or even nearby. Displays, such as loudspeakers, video walls, and projected video could bring personalized information to the people that are standing or moving nearby. However, the infrastructural overhead to facilitate such a scenario is quite high.

The task of making mobile AR work in "unprepared environments" requires solutions closer to the "one-device" end of the spectrum. It should not be overlooked, however, that environment participation often takes place without the user even being aware of it. GPS tracks a person's location by means of a prepared environment on a global scale, namely with the help of satellites orbiting the earth. Wireless communication only works if there are receivers or transceivers in the environment (e.g., point-to-point communication partners, base stations, or satellites).

2.4.2 Displays for Mobile AR

There are many approaches to display information to a mobile person and a variety of different types of displays can be employed for this purpose: personal hand-held, wrist-worn, or head-worn displays; screens and directed loudspeakers embedded in the environment; image projection on arbitrary surfaces, to name but a few. Several of these display possibilities may also be used in combination.

In general, one can distinguish between displays that the person carries on the body and displays that make use of resources in the environment. Wearable audio players and personal digital organizers use displays that fall in the first category, as do wear-

able computers with head-worn displays. An example for the second category would be personalized advertisements that are displayed on video screens that a person walks by. For such a scenario, one would need a fairly sophisticated environmental infrastructure. Displays would need to be embedded in walls and other physical objects, and they would either have to be equipped with sensors that can detect a particular individual's presence, or they could receive the tracking information of passersby via a global computer network. Such environments do not yet exist outside of research laboratories, but several research groups have begun exploring ubiquitous display environments as part of *Smart Home* or *Collaborative Tele-Immersion* setups, such as the Microsoft Research EasyLiving project (Brumitt et al., 2000), or UNC's Office of the Future (Raskar et al., 1998). As part of the latter project, computer generated imagery is projected onto wall-sized displays or even arbitrary surfaces, whose shape is detected using structured light. Such projection displays can be viewed by everyone in that particular room, even though for stereo viewing only one person, whose head pose is accurately tracked, can view the material in an undistorted fashion.

Another display type that is being explored in AR research is the *head-mounted projective display* (Hua et al., 2001). This type of head-worn display consists of a pair of micro displays, beam splitters, and miniature projection lenses. It requires that retroreflective sheeting material be placed strategically in the environment. The head-worn display projects images out into the world, and only when users look at a patch of retroreflective material, do they see the image that was sent out from their eyewear. This approach aims to combine the concept of physical display surfaces (in this case: patches of retroreflective material) with the flexibility of personalized overlays with AR eyewear. A unique personalized image can be generated for each person looking at the same object with retro-reflective coating, as long as their viewing angles are not too close to each other.

One of the most promising approaches for mobile AR might be to combine different display technologies. Head-worn displays provide one of the most immediate means of accessing graphical information. The viewer does not need to divert his or her eyes away from their object of focus in the real world. The immediateness and privacy of a personal head-worn display is complemented well by the high text readability of hand-held plasma or LCD displays, and by the collaboration possibilities of wall-sized displays. One cannot realistically assume that large environments, on the scale of entire cities, will become densely populated with displays of all kinds for the explicit purpose of facilitating AR applications any time soon. However, it is conceivable that personal AR devices may be enabled to make use of specific displays, wherever these already exist, for example in shopping malls or conference centers. Perhaps, the use of multiple displays will start with more sophisticated electronic infrastructures in our homes. Until

then, personal display devices appear to be the option of choice for mobile AR. Mobile AR research at Columbia experimented from early on with head-worn and hand-held displays used in synergistic combination (Feiner et al., 1997; Höllerer et al., 1999a), as shown in Figure 2.4. In the following we will take a look at some noteworthy personal displays.

In 1979, a mobile audio display changed the way people listened to music: the Sony Walkman, which immediately became a hugely popular sales item. It was one of the three most successful fashion products of the 1980s, with the other two being roller skates and digital watches (another kind of mobile display). This commercial success paved the way for other mobile devices, among them personal digital organizers. The original Sony Walkman weighed in at 390g, not counting batteries and audio tape. Today, MP3 players are available, which weigh less than 40 grams, including batteries.

As mentioned in the historical overview in Section 2.2, the concept of *see-through head-worn computer graphics displays* dates back to Ivan Sutherland's work on a head-mounted three-dimensional display (Sutherland, 1968). Some time before that, in 1957, Morton Heilig had filed a patent for a head-worn display fitted with two color TV units (Heilig, 1960). In later years, several head-worn displays were developed for research in computer simulations and the military, including Tom Furness's work on heads-up display systems for fighter pilots (Furness, 1986). VPL Research and Autodesk introduced a commercial head-mounted display for VR in 1989. In the same year, Reflection Technologies started selling a small personal near-eye display, the P4 Private Eye. This display is noteworthy, because it gave rise to a number of wearable computing and VR (Pausch, 1991) and AR (Feiner and Shamash, 1991) efforts in the early 1990s. It had a resolution of 720x280 pixels with 1-bit intensity, using a dense column of 280 red LEDs and a vibrating mirror. The display was well suited for showing text and simple line drawings.

Display technology for computer graphics evolved from simple oscilloscopes to vector graphic terminals to raster graphics terminals with at first limited bitmap resolutions. When memory prices started to decrease by the late 70's and early 80's, and later dropped considerably, raster graphics with increasing pixel resolutions were introduced, and opened the market for different types of high-resolution displays, where plain television cathode ray tubes had been sufficient earlier. RCA made the first experimental liquid crystal display (LCD) in 1968, a technology that should steadily develop and later enable a whole generation of small computing devices to display information. Today, various different technologies are explored for displays of a wide variety of sizes and shapes. *Plasma displays* provide bright images and wide viewing angles for medium-sized to large flat panel form factors. *Organic light emitting diodes* (OLED) can be used to produce ultra-thin displays. Certain types of OLED technology, such as *light emitting*

Figure 2.6: Binocular and monocular optical see-through head-worn displays. (a) Sony Glasstron LDI-D100B, (b) Microvision Nomad, (c) Minolta Forgettable Display.

polymers, might one day lead to display products that can be bent and shaped as required. Of great interest for the development of personal displays are display technologies that are so small that optical magnification is needed to view the images. These are collectively referred to as *microdisplays.* OLED on silicon is one of the options to produce such miniature displays, which increasingly find their way into camera viewfinders, portable projectors, and near-eye displays. Non-emissive technologies for microdisplays include transmissive *poly-silicon LCDs*, and several reflective technologies, such as *liquid crystal on silicon* (LCoS) and *digital micro-mirror devices* (DMD).

One technology that is very interesting for mobile AR purposes is the one employed in MicroVision's Nomad *retinal scanning display.* It is one of the few displays that can produce good results in direct sunlight outdoors. It works by pointing a red laser diode towards an electromagnetically controlled pivoting micro-mirror and diverting the beam via an optical combiner through the viewer's pupil into the eye, where it sweeps across the retina to recreate the digital image. This technology produces a very crisp and bright image, and exhibits the highest transparency any optical see-through display offers today. As of the end of 2002, the Nomad is available as a single-eye monochrome device, shown in Figure 2.6. A full-color and optionally stereoscopic model, dubbed Spectrum, has been announced to enter serial production soon.

When choosing a head-worn display for mobile AR, several decision factors have to be considered. One of the more controversially discussed questions within the AR research community is whether to prefer optical see-through or video feed-through displays.

Optical see-through glasses are transparent, the way prescription glasses or sunglasses are. They use optical combiners, most commonly so-called beam-splitters, to layer the computer generated image on top of the user's view of the environment. Figure

Figure 2.7: (a) Optical- and (b) video-see-through indoor AR. Both images are from indoor AR work performed at Columbia University.

2.7(a) shows an image shot through such glasses. In contrast, video feed-through glasses present a more indirect, mediated view of the environment. One or two small video cameras, mounted on the head-worn display, capture video streams of the environment in front of the user, which are displayed on non-transparent screens right in front of the user's eyes. The computer can modify the video image before it is sent to the glasses to create AR overlays. An example is shown in Figure 2.7(b). More details and a discussion of the advantages and disadvantages of either approach are given in Azuma (1997) and Feiner (2002) . Yet another head-worn display option, which is often employed in non-AR wearable computing applications, is a monocular display. Even if the display is non-transparent, the user is able to see the real world with the non-occluded eye. However, perceiving a true mixture of computer overlay and real world can be somewhat of a challenge in that case.

For mobile augmented reality work, the author of this thesis prefers optical see-through displays. It is his belief that a person walking around in the environment should have as little as possible, preferably nothing, subtracted from their vision. In his opinion, several drawbacks of current video feed-through technology stand in the way of their adoption in truly mobile applications: Seeing the real world at video resolution and at the same small field of view angle used for the graphical overlays, having to compensate for image distortions introduced by the cameras, the risk of any kind of latency in the video feed to the glasses, and safety concerns about seeing the world solely through cameras.

In our experience, monocular displays can yield acceptable results for AR if the display is see-through to make it easier for the user to fuse the augmented view with the other eye's view of the real world (as is the case with the MicroVision Nomad). A bigger field of view is also helpful. It is hard to discuss such display properties in isolation,

however, since quite a few display factors influence the quality of mobile AR presentations, among them monocular vs. binocular, stereo-capability of binocular displays, resolution, color depth, luminance, contrast, field of view, focus depth, degree of transparency, weight, ergonomics, and looks. Power consumption is an additional factor with extended mobile use.

Stereo graphics greatly enhance the AR experience, since virtual objects can then be better perceived at the same distance as the real world objects they annotate. Note though, that even though the stereo option accounts for objects being displayed with the correct interocular distance, all currently available displays have a fixed focus depth, and hence require the viewer's eyes to accommodate at that particular distance, which leads to an accommodation-convergence conflict. Unless the physical object is located at that particular fixed focus distance, the viewer needs to adjust accommodation in order to see either the physical object or the virtual one in perfect focus. Currently, the market for optical see-through head-worn displays is quite limited. If stereoscopic information display is a necessity, the options are even more restricted. The Columbia mobile AR prototypes employed several stereo capable optical see-through displays over the years, none of which are on the market anymore. Figure 2.6(a) shows the Sony LDI-D100B, a display that was discontinued in June 2000.

Display appearances are for the most part still bulky and awkward today. Smaller monocular displays, such as the MicroOptical SV-3 or the Minolta 'Forgettable Display' prototype (Kasai et al., 2000), pictured in 2.6(c), provide a more inconspicuous look, but do not afford the high field-of-view angles necessary for true immersion nor the brightness of, for example, the MicroVision Nomad. Meanwhile, manufacturers are working hard on improving and further miniaturizing display optics. Microdisplays can today be found in a diverse set of products including viewfinders for cameras, displays for cell phones and other mobile devices, and portable video projectors. Near-eye displays constitute a growing application segment in the microdisplay market. The attractiveness of mobile AR relies on further progress in this area.

Apart from the display technology, the single most important technological challenge to general mobile AR concerns tracking and registration.

2.4.3 Tracking and Registration

Augmented Reality requires very accurate position and orientation tracking in order to align, or register, virtual information with the physical objects that are to be annotated. It is highly difficult to fool the human senses into thinking that computer-generated virtual objects actually live in the same physical space as the real world objects around us. In controlled environments of constrained size in indoor computing laboratories, re-

searchers have succeeded in creating environments in which a person's head and hands can be motion-tracked with sufficiently high spatial accuracy and resolution, low latency, and high update rates, to create fairly realistic interactive computer graphics environments that seemingly co-exist with the physical environment. Doing the same in a general mobile setting is a disproportionately bigger challenge. In the general mobile case, one cannot expect to rely on any kind of tracking infrastructure in the environment. Tracking equipment needs to be light enough to wear, fairly resistant to shock and abuse, and functional across a wide spectrum of environmental conditions, including lighting, temperature, and weather. Under these circumstances, there does not currently exist a perfect tracking solution, nor can we expect to find one in the near future. Compromises in tracking performance have to be made, and the applications will have to adjust.

Tracking technology has made big advances since the early days of head-tracked computer graphics. Sutherland's original head-mounted display was tracked mechanically through ceiling-mounted hardware, and, because of all the equipment suspended from the ceiling, was humorously referred to as "the Sword of Damocles." Sutherland also explored the use of an ultrasonic head-tracker (Sutherland, 1968). The introduction of the Polhemus magnetic tracker in the late 1970s (Raab et al., 1979) had a big impact on virtual and augmented reality research, and the same technology, in improved form, is still in use today. During the 1990s, commercial hybrid tracking systems became available, based on different technologies, all explored in experimental tracking systems over the previous decades, such as ultrasonic, magnetic, and optical position tracking, and inertial and magnetometer-based orientation tracking. With respect to global navigation systems, the U.S. Navy experimented with a series of satellite navigation systems, beginning with the *Transit* system in 1965, which was developed to meet the navigational needs of submarines (cf. Pisacane (1998)). The idea for today's *NAVSTAR* Global Positioning System (GPS) (Getting, 1993) was born in 1973. The first operational GPS satellite was launched in 1978, and the 24-satellite constellation was completed in 1993. Satellites for the Russian counterpart constellation, *Glonass* (Langley, 1997), were launched from 1982 to 1998. The European Union has plans underway to launch a separate 30-satellite GPS, called *Galileo* (European Commission, Energy and Transport, 2002).

In the remainder of this section, we will focus on the tracking technologies most suited for mobile augmented reality. For a more comprehensive overview of tracking technologies for AR and VR, we refer the reader to existing surveys of motion tracking technologies and techniques, such as (Rolland et al., 2001) or a recent special tracking issue of IEEE Computer Graphics and Applications (Julier and Bishop, 2002).

Visual registration of virtual and physical material can be achieved in several ways. The common approach is to determine the person's head pose in some global

coordinate system, and relate it to a computer model of the current environment. Note that in this case a computer model of the environment has to be created in a step called *environmental modeling*. This model should use the same global coordinate system as the tracking system, or the necessary conversion transformation has to be known.

Absolute position and orientation of the user's head and the physical objects to be annotated do not necessarily need to be known. In one of the most direct approaches to visual registration, cameras observe specific unique landmarks (e.g., artificial markers) in the environment. If the camera's viewing parameters (position, orientation, field of view) coincide with the display's viewing parameters (e.g., because the display is showing the camera image, as in the case of video-see-through displays), and stereo graphics are not employed, the virtual annotations can be inserted directly in pixel coordinates without having to establish the exact geometric relationship between the marker and the camera (Rekimoto and Nagao, 1995). On the other hand, if the precise locations of the landmarks in the environment are known, computer vision techniques can be used to estimate the camera viewing parameters. The use of cameras mounted on the display together with landmark recognition is sometimes referred to as *closed-loop tracking*, in which tracking accuracy can be corrected to the nearest pixel, if camera image and graphics display coincide. This is in contrast to so-called *open-loop tracking*, which tries to align the virtual annotations with the physical objects in the real world purely relying on the sensed 6DOF pose of the person and the computer model of the environment. Any inaccuracies in the tracking devices or the geometrical model will result in the annotation being slightly off from the intended position in relation to the physical world.

An important criterion for mobile AR tracking is how much tracking equipment is needed on the user's body, and in the environment. The obvious goal is to wear as little equipment as possible, and to not be required to prepare the environment in any way. Note that a system such as GPS meets this requirement for all intended purposes, even though a 'prepared environment' on a global scale is needed, in form of a satellite constellation. Several tracking approaches require some knowledge about the environment. In order to create any useful AR annotations, either the objects to be annotated have to be modeled or *geo-referenced* in absolute terms, or their location must be able to be inferred from a known relationship to pre-selected and identifiable landmarks.

The tracking accuracies required for mobile AR depend very much on the application and the distance to the objects to be annotated. If we are annotating the rough outlines of buildings, we can afford some registration error. When trying to pinpoint the exact location of a particular window, we have to be more accurate. When registration errors are measured as the pixel distance on the screen between the physical target point and the point where the annotation gets drawn, the following observation holds with respect to overall registration error: The further away the object that is to be annotated,

the less of an impact errors in position tracking have, but the more of an impact errors in orientation tracking have. Since most targets in outdoor mobile AR tend to be some distance away from the viewer, one can assume that errors in orientation tracking contribute quite a bit more to overall misregistration than do errors in position tracking (Azuma, 1999). Since there are no standalone sensors that afford general reliable 6DOF tracking in unprepared environments outdoors, mobile AR systems normally resort to hybrid approaches, often employing separate mechanisms for position and orientation tracking.

Position tracking via GPS is a natural candidate for outdoor environments, since it is functional on a global scale, as long as signals from four or more satellites can be received. While use of GPS navigation has long been restricted to areas that afford direct visibility to the satellites, newer reception circuits raise hopes that the area of coverage for commercially available GPS tracking devices can be extended to some regions without direct line-of-sight to the satellites, including urban canyons and indoor areas (Global Locate, 2002). Plain GPS without selective availability is accurate to about 10–15 meters. GPS using the wide area augmentation system (WAAS) is typically accurate to 3–4 meters in the US and other countries that adopt this technology. Differential GPS typically yields a position estimate that is accurate to about 1–3 meters. Real-time kinematic (RTK) GPS with carrier-phase ambiguity resolution can produce centimeter accuracy position estimates. The latter two options require the existence of a nearby base station from where a differential signal can be sent to the roaming unit. Therefore, one cannot really speak of an unprepared environment anymore in that case. Commercial differential services are available, though, with base stations covering most of North America.

Another position-tracking system applicable for wide-area mobile AR involves calculating a person's location from signal quality measures of IEEE 802.11b (WiFi) wireless networking. This obviously also requires the deployment of equipment in the environment, in this case the WiFi access points, but if such a wireless network is the communication technology of choice for the mobile AR system, the positioning system can serve as an added benefit. Several research projects, and at least one commercial product, are exploring this concept. The RADAR system uses multilateration and precomputed signal strength maps for this purpose (Bahl and Padmanabhan, 2000), while (Castro et al., 2001) employ a Bayesian networks approach. The achievable resolution depends on the density of access points deployed to form the wireless network. Ekahau (Ekahau, 2002) offer a software product that allows position tracking of WiFi enabled devices after a manual data collection/calibration step. They report tracking accuracies of 1–3 meters for a typical setup over a wide area indoor environment.

Two additional means of determining position are often employed in mobile AR systems, mostly as part of hybrid tracking systems: Inertial sensors and vision-based approaches. Accelerometers and gyroscopes are self-contained or *sourceless* inertial sen-

sors. Their main problem is drift. The output of accelerometers needs to be integrated once with respect to time, in order to recover velocity, and twice to recover position. Hence, any performance degradations in the raw data lead to rapidly increasing errors in the resulting position estimate. In practice, this approach to position estimation can only be employed for very small time periods between updates gathered from a more reliable source. Inertial sensors can also be used to detect the act of a pedestrian taking a step, the principle of *pedometers*, which, when combined with accurate heading information, can provide a practical *dead-reckoning* method (Höllerer et al., 2001b).

Vision-based approaches (Koller et al., 1997) are a promising option for 6DOF pose estimation in a general mobile setting. One or two tiny cameras are mounted on the glasses, so that the computer can approximately see what the user sees. Model-based vision techniques require an accurate model of the environment with known landmarks that can be recognized in the image feeds. In contrast, move-matching algorithms track dynamically chosen key points along the image sequence, leading to relative, rather than absolute, tracking solutions, which means that further registration of the image sequence coordinate system with the physical world needs to be established to enable 3D graphical overlays. Simultaneous reconstruction of the camera motion and scene geometry is possible, but such computations are highly computationally expensive, and existing algorithms require a 'batch bundle adjustment,' a global offline computation over the entire image sequence. Finally, 2D image-based feature tracking techniques measure so-called 'optical flow' between subsequent video images. Such techniques are comparatively fast, but by themselves cannot estimate 3D camera motion. Combinations of all these approaches are possible. Recent research reports promising results for some test cases (Julier and Bishop, 2002). However, in general, computer vision algorithms still lack robustness and require such high amounts of computation that only a few specific pure vision solutions have been applied in real-time so far. For the time being, hybrids of vision based tracking and other sensing technologies show the biggest promise.

Orientation tracking also benefits greatly from hybrid approaches. The basic technologies available for orientation sensing are electromagnetic compasses (magnetometers), tilt sensors (inclinometers), and gyroscopes (MEMS and optical). Hybrid solutions have been developed, both as commercial products and research prototypes. The IS300 and InertiaCube2 orientation sensors by InterSense (InterSense, 2001) combine three MEMS gyroscopes, three accelerometers (for motion prediction), and an electromagnetic compass in one small integrated sensor. Azuma et al. (1999a) present a hybrid tracker that combines a carefully calibrated compass and tilt sensor with three rate gyroscopes. You et al. (1999) extended that system by a move-matching vision algorithm, which did not, however, run in real time. Behringer (1999) presents a vision-based correction method based on comparing the silhouette of the horizon line with a model of

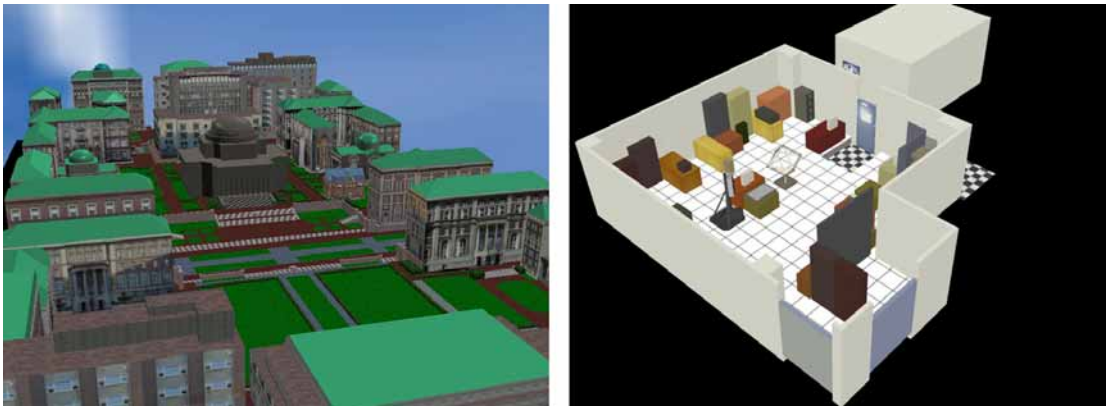


Figure 2.8: Environmental Modeling: (a) Model of Columbia's campus. (b) Model of the Computer Graphics and User Interfaces Laboratory.

local geography. Satoh et al. (2001b) employ a template matching technique on manually selected landmarks in a real-time algorithm that corrects for the orientation drift of a highly accurate fiber optic gyro sensor.

In summary, the problem of tracking a person's pose for general mobile augmented reality purposes is a hard problem with no single best solution. Hybrid tracking approaches are currently the most promising way to deal with the difficulties of general indoor, and in particular outdoor, mobile AR environments.

2.4.4 Environmental Modeling

For AR purposes, it is often useful to have access to geometrical models of objects in the physical environment. As mentioned above, one use of such models is in registration. If you want to annotate a window in a building, the computer has to know where that window is located with regard to the user's current position and field of view. Having a detailed hierarchical 3D model of the building, including elements such as floors, rooms, doors, and windows, gives the computer the utmost flexibility in answering such questions and working out the annotations correctly. Some tracking techniques, such as the model-based computer vision approaches mentioned in Section 2.4.3, rely explicitly on features represented in more or less detailed models of the tracking environment. Geometrical computer models are also used for figuring out *occlusion* with respect to the observer's current view. For example, if portions of a building in front of the observer are occluded by other objects, only the non-occluded building parts should be annotated with the building's name (Bell et al., 2001), since otherwise the observer might easily be confused as to which object is annotated by the label.

For the purposes mentioned so far, an environment model does not need to be

Figure 2.9: Context-Overview: World-in-Miniature map displayed in Columbia MARS 2000 demonstration.

photorealistic. One can disregard materials, textures, and possibly even geometric detail. In fact, in model based tracking, often only a 'cloud' of unconnected 3D sample points is used.

More realistic geometric models of real-world objects are sometimes used for annotation purposes, or for giving an overview of the real environment. For example, a building that is occluded from the user's view can be displayed in its exact hidden location via augmented reality, enabling the user, in a sense, to see through walls. Somebody looking for a specific building can be shown a virtual version of it on the AR display. Having gotten an idea of the building's shape and texture, the person might watch the model move off in the correct direction, until it coincides with the real building in physical space. A 3D map of the environment can be presented to the user to give a bird's-eye overview of the surroundings. Figures 2.9 and 5.17 show examples of such worlds in miniature (WiM) (Stoakley et al., 1995) being used in AR.

Creating 3D models of large environments is a research challenge in its own right. Several automatic, semi-automatic, and manual techniques can be employed, among them 3D reconstruction from satellite imagery, 3D imaging with laser range finders, reconstruction from a set of overlapping photographs, surveying with total stations and other telemetry tools, and manual reconstruction using 3D modeling software. Even AR techniques itself can be employed for modeling purposes, as mentioned in Section 2.3.1.6. Abdelguerfi (2001) provides an overview of 3D synthetic environment reconstruction. The models in Figure 2.8 were reconstructed by extruding 2D map outlines of Columbia's campus and a research laboratory, refining the resulting models by hand, and texture mapping them selectively with photographs taken from various strategic positions. The WiM of Figure 2.9 shows the currently selected building (Dodge Hall) as a 3D model in the context of a satellite image of the current environment.

3D spatial models can be arbitrarily complex. Consider, for example, the task of

completely modeling a large urban area, down to the level of water pipes and electricity circuits in walls of buildings. Not just the modeling task, but also the organization and storage of such data in spatial databases and data structures optimized for specific queries, is linked to several lines of active research. Finally, environmental modeling does not end with a static model of the geometry. Most environments are dynamic in nature, or at least have considerable dynamic components. Changes in the geometric models (due to moving objects, construction, or destruction) need to be tracked, and reflected in the environmental model. The databases may need to change quite rapidly, depending on what levels of updates are considered.

2.4.5 Wearable Input and Interaction Technologies

How to interact with wearable computers effectively and efficiently is another open research question. The WIMP desktop interface metaphor is not considered a good match for mobile and wearable computing, mostly because it places unreasonable motor skill and attention demands on mobile users interacting with the real world. Similarly, there is no obvious best solution for text input to wearable and mobile systems. Instead, users employ different technologies and techniques in different situations. As a general UI principle, augmented reality can provide a very immediate computing interface to a user engaged with the physical world. Visual attention does not need to be divided between the task in the physical world and a separate computer screen. But interacting seamlessly with such a computing paradigm is a challenge. In this section we review interaction technologies that have been tried with mobile augmented reality systems.

Basic interaction tasks that graphical computer interfaces handle, include selection, positioning, and rotation of virtual objects, drawing paths or trajectories, the assignment of quantitative values, referred to as quantification, and text input (Foley et al., 1984). AR interfaces deal as much with the physical world as with virtual objects. Therefore, selection, annotation, and, possibly, direct manipulation of physical objects also play an important role in these kind of interfaces.

We already mentioned one class of input devices, namely the sensors that afford tracking and registration. Position tracking determines the user's locale, and head-orientation tracking assists in figuring out the user's focus. Establishing the user's context in this fashion can very effectively support user interaction. The user interface can adapt to such input by limiting the choices for possible courses of action to a context-relevant subset. Both position and head orientation tracking can also be employed for object selection. Suppose that the task is to select a building in an urban neighborhood. With position tracking only, the respectively closest building, or a list of the n closest ones, might be listed on the display for direct selection via a button or scrolling input device.

With head orientation, the user can point the head into the direction of the object to be selected. Selection can take place by dwelling for a certain time period on the object in view, or by active selection via button-like devices. Höllerer et al. (1999a) have discussed several tracking-prompted selection mechanisms for a mobile AR system. Additional orientation trackers can provide hand tracking, which can be used to control pointers or manipulate virtual objects on the AR screen. Tracking hand or finger position for full 6DOF hand tracking, as is common in indoor virtual or augmented environments, would be a great plus for mobile AR, but is hard to achieve with mobile hardware in a general setting. Research prototypes for this purpose have experimented with vision-based approaches (Starner et al., 1997b), and ultrasonic tracking of finger-worn acoustic emitters using three head-mounted microphones (Foxlin and Harrington, 2000).

Quite a few mobile input devices tackle continuous 2D pointing. The tasks commonly performed by mice in desktop systems are covered in the mobile domain by trackballs, track-pads, and gyro-mice, many of which wirelessly transmit data to the host computer. It should be mentioned, though, that these devices owe their popularity to the fact that in absence of a better mobile interface standard, many people run common WIMP interfaces on their mobile and wearable platforms for the time being. Accurate 2D pointing poses a big challenge for a mobile user's motor skills. However, the 2D pointing devices can also be used to control cursor-less AR interfaces (Feiner et al., 1997). When user interaction mostly relies on discrete 2D pointing events, such as selecting from small lists of menu items, then small numeric keypads with arrow keys, or arrow keys only, might provide a solution that is more easily handled on the run, and more easily worn on the body.

Mobile interfaces should try to minimize the burden of encumbering interface devices. The ultimate goal is to have a free-to-walk, eyes-free, and hands-free interface with miniature computing devices worn as part of the clothing. As becomes clear from our overview so far, this ideal cannot be reached yet with current mobile computing and interface technology. Some devices already nicely meet the size and ergonomic constraints of mobility: auditory interfaces for example, can already be realized in a pretty inconspicuous manner, with small wireless earphones tucked barely noticeably into the ear, and microphones worn as part of a necklace or shoulder pad. There is a growing body of research on wearable audio interfaces, dealing with topics such as speech recognition, speech recording for human-to-human interaction, information presentation via audio, and audio dialogue (Degen et al., 1992; Roy and Schmandt, 1996; Mynatt et al., 1997; Sawhney and Schmandt, 1998). It is clear, though, that a standalone audio interface cannot offer the best possible solution for every situation. Noisy environments and places that demand silence pose insurmountable problems to such an approach. Audio can be a valuable medium for multimodal and multimedia interfaces, though.

Other devices are more impractical for brief or casual use, but have successfully been employed in research prototypes exploring next-generation interfaces. *Glove-based* input devices, for example, using such diverse technologies as electric contact pads, flex sensors, accelerometers, and even force-feedback mechanisms, can reliably recognize certain hand gestures, but have the drawback of looking awkward and impeding usage of the hands in real-world activities. Nevertheless, the reliability and flexibility of glove gestures has made the computer glove an input device of choice with some mobile AR prototypes (Thomas and Piekarski, 2002). Starner et al. (1997b) on the other hand, explore vision-based hand gesture recognition, which leaves the hands unencumbered, but requires that a camera be worn on a hat or glasses, pointing down to the area in front of the user's body, in which hand gestures are normally made.

We already discussed the use of cameras for vision-based tracking purposes (Section 2.4.3). Apart from that purpose, and the potential of finger and hand gesture tracking, cameras are used to record and document the user's view. This can be useful as a live video feed for teleconferencing, or for informing a remote expert about the findings of AR field-workers, or simply for documenting and storing everything that is going on in front of the mobile AR user. Recorded video can be an important element in human-to-human interfaces, which AR technology nicely supports.

A technology with potential for mobile AR is gaze control (Jacob, 1991). Eye trackers observe a person's pupils with tiny cameras in order to figure out where that person's gaze is directed. Drawbacks are the additional equipment that needs to be incorporated into the headset and eyewear, the brittleness of the technology (the tracker needs to be calibrated and the cameras are not allowed to move with respect to the eye), and the fact that there is a lot of involuntary eye movement that needs to be correctly classified as such. With the right filters, however, gaze control could provide a very fast and immediate input device. As a pointing device, it leaves out an entire step of coordinated muscle activity that is needed with other pointing devices to move a mouse pointer to a location that was selected via eye movement in the first place. As an additional benefit, gaze control provides information about what a user's attention is directed at, at each point in time. As computers gather more and more knowledge about the user's interest and intention, they can adapt their interfaces better to suit the needs of the current context.

Other local sensors that can contribute to more knowledge about the user's state include biometric devices that measure heart-rate and bioelectric signals such as galvanic skin response, electroencephalogram (neural activity), or electromyogram (muscle activity) data. Employing such monitored biological activity for computer interface purposes is a fairly ambitious research endeavor, but the hopes and expectations for future applicability are quite high. Affective computing (Picard, 1997) aims to make computers more

aware of the emotional state of their users and adapt accordingly.

As we can see, user interface technology can be integrated with the user more or less tightly to various degrees. While the previous paragraph hinted at possible future man-machine symbioses, current wearable computing efforts aim to simply make computing available in as unencumbered a form as possible. One item on this agenda is to make clothes more computationally aware, for example by embroidering electronic circuits (Farringdon et al., 1999). On the other hand, not every interface technology needs to be so tightly integrated with the user. Often, different devices that the user would carry instead of wear on the body, can support occasionally arising tasks very efficiently. For example, hand-held devices such as palmtop organizers or hand-held tablet computers are good candidates for reading devices, and are well suited for pen-based input, using handwriting recognition and marking gestures. Hybrid interfaces, as Feiner et al. (1997) explored them for mobile AR purposes, aim to employ different display and input technologies and reap the benefits of each technology for the purposes it is best suited for.

Fast and reliable text input to a mobile computer is hard to achieve. The standard keyboard, which is the proven solution for desktop computing, requires too much valuable space and a flat typing surface to be considered as an option. Still, small, or foldable, or even inflatable keyboards, or virtual ones that are to be projected by a laser device onto a flat surface (May, 2003), are current commercial options or product prototypes. Chording keyboards, which require key combinations to be pressed to encode a single character, such as the one-handed Twiddler (Handykey, 2001), are very popular choices for text input in the wearable computing community. Cell phones provide their own alphanumeric input techniques via a numeric keypad. We already mentioned speech recognition, which boosted major improvements in accuracy and speed over the last ten years, but cannot be applied in all situations, handwriting recognition and pen-based marking interfaces. Soft keyboards enable text input via various software techniques, but they use up valuable display screen space for that purpose. Glove-based and vision-based hand gesture tracking do not provide the ease of use and accuracy necessary for serious adoption yet. It seems likely that speech input and some kind of fallback device (currently, the odds favor pen-based, or special purpose chording or miniature keyboards) will share the duty of providing text input to mobile devices in a wide variety of situations in the future.

This section reviewed various input devices and technologies for wearable computing in general, and mobile AR in particular. As a final remark, the applicability of such a wide variety of input technologies is supported nicely by *multimodal user interface* techniques (Cohen et al., 1998). Such techniques employ multiple input and output modes in combination (gestures, speech, vision, sound, and haptics), using different media to present the user with a more natural, yet still predictable and robust, interface.

2.4.6 Wireless Communication and Data Storage Technologies

We already discussed the mobility of a computing system in terms of its size, ergonomics, and input/output constraints. Another important question is how connected such a system is in the mobile world. This question concerns the electronic exchange of information with other, mobile or stationary, computer systems. The degree of connectivity can vary from none at all to true global wireless communication. Most likely is a scenario where the mobile client has different options to get connected to the internet, ranging in covered area and connection speed from a fast direct cable connection when used in a stationary office environment, to slightly slower wireless local area networks (WLANs), which offer full connectivity in building- or campus sized networks of wireless access points, to wireless phone data connections with close to nationwide coverage but much slower transmission speeds.

Wireless data transmissions date back to the invention of radio by Marconi in 1896, but the first packet-based WLAN was the ALOHANET at the University of Hawaii in 1971. Today, WLANs provide bandwidths ranging between 2 and 54 Mbps, and are quite common for providing coverage in campuses and homes. At least one telecommunications company has announced plans for a nationwide rollout of IEEE 803.11b (WiFi) networks (Cometa Networks, 2002), and started implementing it. During the first two years of the current century, phone service providers began to roll out new nationwide networks based on third generation wireless technology (at bandwidths of 144 Kbps, and higher in some selected test areas), which nicely complement smaller sized community WLANs.

For point-to-point connections between different devices, the Bluetooth consortium (Ericsson et al., 1998) has established an industry standard for low-power close-range radio frequency communication. Using this technology, wearable computers connect with respectively enabled input/output devices that a user can carry or wear on the body, or walk up to, as in the case of stationary printers. Bluetooth-enabled cellular phones provide access to nationwide wireless connectivity whenever faster networking alternatives are not available.

From the perspective of mobile AR systems, the integration of location based services with communication systems is an important issue. Where it might be sufficient for special purpose AR systems to store all related material on the client computer, or retrieve it from one single task-related database server, this is not true anymore in the mobile case. For true mobility, the AR client will need to connect to multiple distributed data servers in order to obtain the information relevant to the current environment and situation. Among the data that need to be provided to the client computer are the geometrical models of the environment (see Section 2.4.4), annotation material (object names, descriptions, and links), as well as conceptual information (object categorization and

world knowledge) that allows the computer to make decisions about the best ways to present the data. Some formalism is needed to express and store such meta-knowledge. Markup languages, such as various XML derivatives (MRML, 2002; GML, 2003), are very well suited for this purpose. XML, with its wide, and further increasing, adoption all over the World Wide Web, offers the advantages of a common base language that different authors and user groups can extend for their specific purposes.

For interactive applications, as required by AR interface technology, all or most of the data that is to be displayed in world-overlays should be stored, or cached on the local (worn or carried) client computer. That prompts the question of how to upload and "page in" information about new environments that the mobile user is ready to roam and might want to explore. Such information can be loaded preemptively from distributed databases in batches of relative topical or geographical closure (e.g., all restaurants of a certain neighborhood close to the user's current location). We would like to emphasize that currently no coherent global data repository and infrastructure exists that would afford such structured access to data. Instead, different research groups working on mobile AR applications established their own test infrastructures for this purpose. For the purpose of our AR work, data and meta-data is stored and accessed dynamically in relational databases, and distributed to various clients via a data replication infrastructure. The database servers effectively turn into *AR servers*, responsible for providing the material used by the client for overlays to particular locations. More details can be found in Section 3.2.

Research from grid computing, distributed databases, middleware, service discovery, indexing, search mechanisms, wireless networking, and other fields comes together to build examples of new communication infrastructures that enable such semantically prompted data access. The internet offers the backbone to experiment with such data distribution on virtually any level of scale.

2.5 Existing Mobile AR Systems

As we mentioned in Section 2.2, truly mobile visual AR systems first became possible in the mid 1990s. Early indoor prototypes, such as the Sony CSL *Navicam* (Rekimoto and Nagao, 1995) experimented with portable display/camera combinations overlaying simple annotations onto an environment tagged by visual markers recognizing markers, but the video processing had to be done on more powerful stationary computers, which required a cable connection or wireless video transmission to nearby servers (Starner et al., 1997a). During the second half of the 1990s portable computing hardware became powerful enough to do 3D graphics processing and video processing on the wearable host, and more widespread wireless networking further supported computational mobil-

ity. While in 1998 there existed only a few experimental MARS (Feiner et al., 1997; Billinghurst et al., 1997; Azuma et al., 1998; Thomas et al., 1998), in recent years an increasing number of researchers started to tackle the various problems associated with mobile, and in particular outdoor, AR systems.

MARS systems with a focus on tracking accuracy have been demonstrated by Azuma and colleagues at HRL Laboratories and Behringer and colleagues at Rockwell Scientific (Azuma et al., 1999a; Azuma et al., 1999b; Behringer, 1999; Behringer et al., 2000). In related recent work, several research groups from around the globe strive to improve the process of natural feature tracking as a component of mobile AR registration (Lee et al., 2002; Chia et al., 2002; Simon and Berger, 2002; Genc et al., 2002; Behringer et al., 2002; Satoh et al., 2001a).

Researchers at the University of South Australia have implemented a series of MARS systems for terrestrial navigation (Thomas et al., 1998; Thomas et al., 2000; Piekarski and Thomas, 2001a), also experimenting with a stationary indoors VR facility (Piekarski et al., 1999). Their more recent prototypes employ a glove-based interface to creating and manipulating augmented material in the outdoors.

Billinghurst and colleagues at the Human Interface Technology Lab of the University of Washington developed a series of collaborative wearable information spaces and evaluated different UI ideas in simple user studies (Billinghurst et al., 1998a; Billinghurst et al., 1998b; Billinghurst et al., 1998c; Billinghurst et al., 1998d).

In a research collaboration with Columbia University, a group at the Naval Research Laboratory has built a backpack MARS for the Battlefield Augmented Reality System (BARS) project (Julier et al., 2000a; Baillot et al., 2001). NRL and Columbia coupled their systems to explore collaborative MARS UI issues. (Julier et al., 2000b; Höllerer et al., 2001a).

As part of the Mixed Reality Systems Project (Tamura et al., 2001), the TOWNWEAR project (Towards Outdoor Wearable Navigator With Enhanced and Augmented Reality) (Satoh et al., 2001b; Satoh et al., 2001a) has presented an outdoor MARS that uses high-precision fiber-optic gyroscopes and vision-based corrections for head orientation tracking. Fully general position tracking yet has to be demonstrated.

Reitmayr and Schmalstieg at the Technical University Vienna adapted their *Studierstube* environment for use with a mobile system, and explore collaborative user interface issues (Reitmayr and Schmalstieg, 2001b; Reitmayr and Schmalstieg, 2001a; Reitmayr and Schmalstieg, 2003). Researchers at the University of Munich developed a component-based MARS and explore its use for the maintenance of power plants (Klinker et al., 2001)

The Ubiquitous Communications project at the Delft University of Technology has developed a wireless augmented reality terminal that is low-power and truly wearable

(Pouwelse et al., 1999; Pasman and Jansen, 2001).

The Archeoguide system (Dähne and Karigiannis, 2002; Vlahakis et al., 2002) is an outdoor AR prototype for visualizing virtual reconstructions in situ at archeological sites. Because of their image-matching approach to registration, the mobility of the observer is constrained to certain viewing areas. Researchers at the University of Nottingham implemented a mixed reality system that shows the historical context of a medieval site on a mobile video display (Schnädelbach et al., 2002).

The Nokia Research Center in Tampere, Finland, has experimented with several augmented reality interfaces for palmtop- and wearable computer based navigational guidance (Suomela and Lehikonen, 2000; Suomela et al., 2001).

Sato et al. (1999) propose a Mixed Reality system for inspection of electronic parts to be used within the boundaries of a wide manufacturing plant. Their MARS backpack is tracked with a purely inertia-based (gyroscopic orientation tracker plus acceleration sensor) tracking system and calibration has to be frequently adjusted by hand. A group at the Electronics and Telecommunication Research Institute in South Korea (Jang et al., 1999) mount a GPS system and a wireless CCD on a helicopter and fuse the resulting data with information from an existing 3D GIS database of the covered area, thus producing AR images of the area captured by the CCD.

Finally, there is much synergy between the fields of wearable computing (Mann, 1997) and augmented reality. Steve Mann uses the term *mediated reality* to emphasize the active role the wearable computer plays in enhancing and altering a user's impressions of the world, constantly experienced through cameras (Mann, 1998). Starner and colleagues use augmented reality in several indoor wearable computing applications (Starner et al., 1997a; Starner et al., 2000).

Chapter 3

System Designs

To explore the possibilities of mobile AR interfaces in general outdoor scenes, we built a series of MARS prototypes, starting with extensions to the 1997 "Touring Machine" and leading up to the most recent system, more prosaically christened "MARS 2002." Our experimental backpack-based systems were all assembled using commercial hardware that we have chosen mostly for high performance and flexibility, as weighed against comfort and wearability. The rationale behind this "functionality before form" approach is that it allows us to explore the potential of the most powerful available AR technologies in the field over an extended period of time, upgrading specific components whenever new technologies become available. While it is theoretically possible to create a much smaller version of a MARS backpack system, this would mean losing the flexibility of easy replacement of parts, as well as foregoing the software development benefits of a powerful standard operating system with a rich choice of software development tools and existing libraries. Minimizing weight and overall size would also have been a vastly more expensive choice.

It is our observation and expectation that miniaturization and integration of the components necessary for MARSs will continue to take place, helped along and accelerated by the pressures of the marketplace, driven by already promising advancements of wearable computing technology. In this light, we want to be able to explore the interface possibilities of tomorrow using today's technology; hence our efforts to stay on top of the technological possibilities, given the constraints of a system that can realistically be worn and carried over extended periods of time. At the same time, it is clear that for anybody to wear and use such a system on a regular basis for an extended period of time, major adjustments in size, weight, and comfort would have to be made. Section 3.1 presents an overview of our explorations in hardware design, as we extended and improved our testbed platform for outdoor AR interfaces. Appendix A reports the details of our designs.

Regarding this overall system development strategy, it becomes clear that the evolution of our MARS prototypes is to be seen as a step-wise but steady and continuous improvement of a starting configuration, at least as far as the computing hardware is concerned. In terms of software architectures, our approach enabled us to try considerably different solutions, and the sequence of five software architectures we will describe in Section 3.2 is in fact a series of different snapshots witnessing the transition from one major system architecture, based on Modula-3 (Harbison, 1992), COTERIE (MacIntyre and Feiner, 1996b), and Repo-3D (MacIntyre and Feiner, 1998), to another, based on Java (Gosling et al., 1996), Java3D (Sowizral et al., 1997), and several other Java-based libraries. Having said that, each of the five different software architectures presented here, actually forms a complete system, each tested and put to use by various new user interface techniques, which will be reported upon in detail in Chapter 5, after Chapter 4 gives a taxonomic foundation of MARS UIs in general.

We conclude the chapter with a presentation of the system design and tracking research we carried out to support MARS interfaces indoors. This work is presented in Section 3.3.

3.1 System Architectures, Hardware

We mentioned above that our philosophy of hardware system design supports gradual improvements of computing hardware. Still, over the last six years there have been noticeable milestones, brought about by several new technologies enabling novel mobile 3D graphics solutions. In this section, supported by Appendix A, we summarize four different MARSs, which reveal considerable differences in terms of the specific parts and technologies employed, while still following the same overall formula of a backpack-based system with hardware-accelerated 3D graphics overlaid on the physical world by optical see-through head-worn displays, and tracked by GPS and head-orientation trackers. We start our hardware systems review by recapitulating the components of such a generic system, and then talk about the different specific components used over the years, and the progress in computing, tracking, and display power and accuracy that they brought with them.

Figure 3.1 gives a schematic overview of a generic MARS, implemented using a see-through and hear-through head-worn display, head-orientation tracking and differential GPS, possibly assisted by dead-reckoning sensors, wireless networking, and a choice of interaction peripherals, connected to the backpack computer in a wireless fashion if possible. Among these, we have additional computing devices to choose from, such as a pen-controlled handheld PC or a palm-size computer or PDA.

Considering the arsenal of MARS technologies depicted (discussed at length in

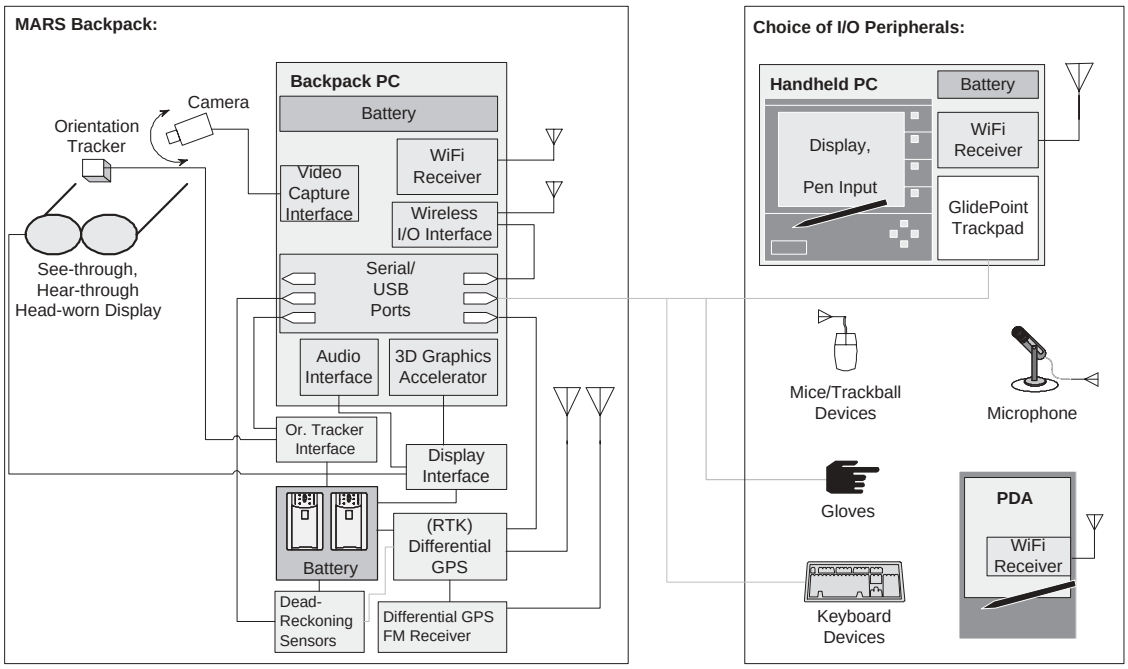


Figure 3.1: General MARS hardware diagram.

the preceding chapter), the most notable elements missing from the MARS prototypes we actually implemented, as will be described in the following subsections, are camera-based vision and some specific input devices.

The reason for not considering camera vision for the UIs we have been developing so far, is mainly that employment of computer vision for mobile AR purposes is a highly challenging research topic all by itself. As such, it is beyond the scope of this thesis, which focuses on the UI design aspect of MARSs. It should also be mentioned that video feeds can be a useful interface mechanism for human-to-human interaction using MARSs, and video camera support for this (and for scene documentation and system debug purposes) has been implemented in our later MARS prototypes (cf. Sections A.4 and Section 3.2.3).

Other input devices that we do not currently support in any of our prototypes (e.g., glove-based devices) have simply not been a sufficiently good match with our general mobile application settings. Gloves, for example, have the disadvantage of encumbering the user's hands, which are important for nearly all everyday activities. However, we can imagine future finger-tracking technologies that would limit such an encumbrance to a minimum. In its current form, glove devices can be advantageously employed in specific mobile tasks in which the user repeatedly performs extensive 3D interactions with virtual material, and we consider related research on such interfaces in our MARS component taxonomy (Chapter 4).

Reviewing the series of MARSs we implemented over the years, the improvements in performance are quite striking. The processor on the main backpack computer changed from a 133 MHz Pentium in 1997 to a 2.2 GHz Mobile Pentium 4 in 2002, an estimated 25-fold increase in processing power, taking into account clock frequency and optimized processor architectures.

The progress in graphics rendering speed has been even more drastic: From 1997 to early 2003 the claimed rendering performance skyrocketed by a factor of over 150 from 300,000 triangles per second on a GLINT 500DTX chip to 50 million triangles on a Quadro4 700Go. At the same time the claimed fill rate still rose by a factor of almost 50 from 16.5 megapixels per second on the 1997 GLINT chip to 880 megapixels per second on the Quadro4 Go.

We also witnessed good progress in display hardware resolution and tracker accuracy, even though it has not been as pronounced, and especially not as steady and continuous as the above performance improvements. Interestingly, even though it was not our main target for optimization, the weight of the overall system dropped from about 40 pounds in 1997 to about 25 pounds in 2002. Much more improvement would be possible in this arena, if one were to sacrifice some of the flexibilities associated with off-the-shelf components.

The details on the different hardware prototypes we developed can be found in Appendix A. The Touring Machine hardware is discussed in A.1. Section A.2 presents the MARS 1999 updates to the architecture. A.3 describes the MARS 2000 that was based on a custom-assembled main computer, and A.4 details the latest prototype, the MARS 2001/2002, which utilizes the power of newly available graphics notebooks.

3.2 System Architectures, Software

We will now take a look at the software architectures that we developed to utilize the hardware platforms just described. The software architectures, or infrastructures, form the foundation (in terms of modules, libraries, device drivers, prototyping facilities, etc.) for UI development on our MARSs. We will briefly present five major architectures that we built over the years in the Computer Graphics and User Interfaces Laboratory (CGUI Lab). SitDoc, the Situated Documentary Extensions to COTERIE; IN-N-OUT AR, a COTERIE/Java hybrid bridging the in- and outdoors; JABAR, the JAva Based AR environment; CGUI Lab AR environment, a shared central Java and Java3D-based infrastructure, most of which was implemented by Blaine Bell; and our extension of this infrastructure to *Ruby*, a Rule-Based AR Environment.

There is not exactly a one-to-one relationship between hardware platforms and software infrastructures. Table 3.1 lists how the different infrastructures map to the hard-

Software Infrastructure	Years	Hardware Platform(s)
COTERIE	1996–1997	Touring Machine
COTERIE w. SitDoc extensions	1998–1999	Touring Machine
IN-N-OUT AR (Java/Java3D + SitDoc)	1999–2000	MARS 1999, MARS 2000
JABAR (Java/Java3D)	2000–2001	MARS 2000, MARS 2001
CGUI Lab AR	2000–2002	MARS 2001/2002
Ruby (Rule-based AR: CGUI AR/Jess)	2001–2003	MARS 2001/2002

Table 3.1: MARS software infrastructures and hardware platforms.

ware platforms described in Section 3.1. The table also illustrates how the different infrastructures form a transition from the COTERIE environment, designed and implemented by Blair MacIntyre, to several infrastructures built on top of Java and Java3D. With the exception of the distribution architecture used in IN-N-OUT AR, which was implemented by Drexel Hallaway, and the CGUI Lab AR infrastructure, which was mostly implemented by Blaine Bell, all software architectures starting from the SitDoc Extensions were mainly designed and implemented by the author of this thesis.

We make the distinction between new infrastructures and mere extensions of existing environments based on the code bases of the software in question. The Situated Documentary extensions were built on top of a large existing code base (COTERIE, Repo-3D). Each of the other infrastructures listed here consists mainly of new or substantially restructured code. Each of these infrastructures has served as the basis for various MARS UI techniques, which will be described in detail in Chapter 5.

3.2.1 COTERIE-based MARS

From 1997 to 1999, COTERIE with its interpreted prototyping language Repo (MacIntyre, 1997) and distributed graphics library Repo-3D (MacIntyre and Feiner, 1998) formed the main development environment for our MARS interfaces.

COTERIE, developed by MacIntyre (1998), is a prototyping infrastructure that provides language-level support for distributed virtual environments. It runs on Windows NT/95, Solaris, and IRIX, and includes the standard services needed for building virtual environment applications, including support for assorted trackers, etc. The software is built on top of Modula-3 (Harbison, 1992) and Repo (MacIntyre, 1997), which is an extended variant of the lexically scoped interpreted language Obliq (Cardelli, 1995).

As a graphics package the system used Repo-3D, an extended version of Obliq-3D (Najork and Brown, 1995). The display-list based 3D graphics package Obliq-3D was modified and extended both to provide additional features needed for virtual environment

applications and to achieve better performance.

A great advantage of using the COTERIE environment is the programming support that it offers for the development of distributed graphical environments. COTERIE is based on the distributed data-object paradigm for distributed shared memory. Any data object in COTERIE can be declared to be a shared object that either exists in one process, and is accessed via remote-method invocation, or is replicated fully in any process that is interested in it. The replicated shared objects support asynchronous data propagation with atomic serializable updates, and asynchronous notification of updates. The main disadvantage of COTERIE, and the major reason that we eventually turned to a Java-based development environment, was the small user base and lack of support for this development environment, which made it impossible to maintain the code base and keep up with innovations in the field. However, at the time we started this research, Java was not available yet, and COTERIE offered sufficient advantage over existing C/C++-based systems to make it the platform of choice for the Touring Machine and first Situated Documentaries application environments.

Since in their original form, Repo and Repo-3D did not support the multimedia capabilities we needed for our work on Situated Documentaries (cf. Section 5.1), and also lacked support for the new devices that were part of MARS 1999 (cf. Section A.2), we extended the infrastructure with multimedia libraries, a new scripting language to put together multimedia presentations, and new device drivers. We refer to all these extensions collectively as the SitDoc extensions, described in Section 3.2.1.2

3.2.1.1 Campus Tour Architecture

The COTERIE base architecture for the Touring Machine and Situated Documentaries extensions was implemented on top of various Microsoft Windows platforms, in order to benefit from its support for assorted commercial peripherals. Initially, we ran Windows NT on the backpack computer. We ran Windows 95 on our first handheld computer because it did not support Windows NT, and we switched to Windows 98 on our second handheld, because it offered better pen extensions than did Windows NT.

Information on the handheld computer is presented entirely via a web browser. We selected Netscape because of its popularity at the time and the ease with which it can be controlled from another application. Note that the use of a web browser as the main application interface does not constrain us much in terms of the general type of interfaces that we can provide on the handheld, since we can make use of Java applets and Active-X components to give interaction the face we want it to have. The “time-machine” interface of Section 5.1.1 illustrates this approach.

Figure 3.2 gives an overview of the main software architecture for the COTERIE-based campus tour. It was designed and implemented by Blair MacIntyre, with program-

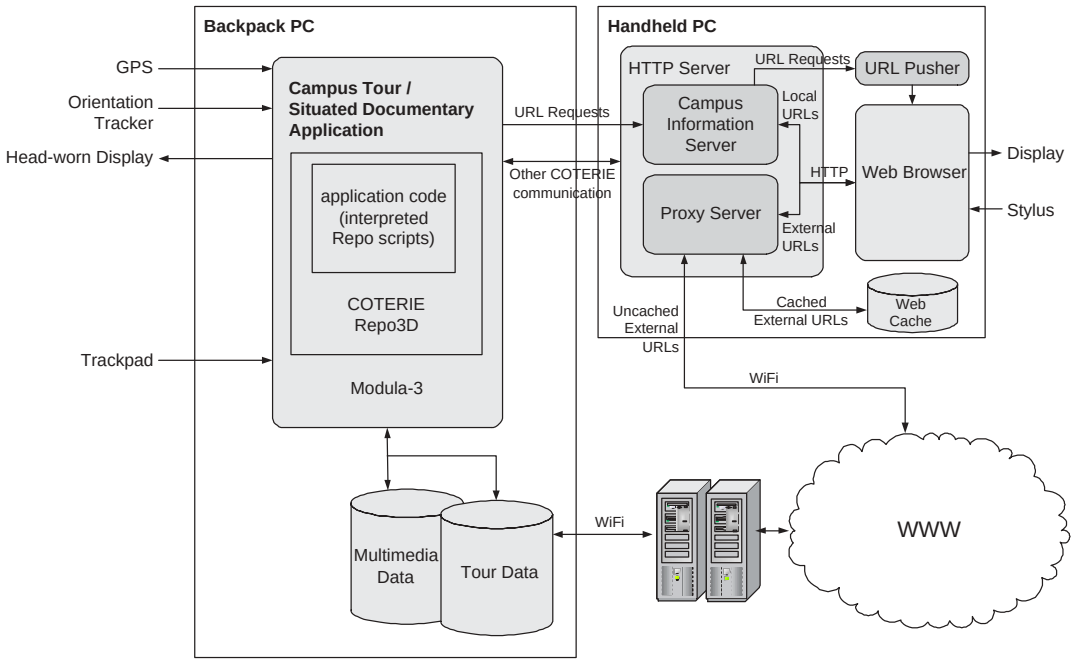


Figure 3.2: MARS 1997–1999 software architecture.

ming support from the author of this thesis. The application software consists of two main modules, one running on each machine, implemented in a total of approximately 3600 lines of commented Repo code.

The tour application running on the backpack computer continuously receives input from the GPS position tracker and the orientation tracker. It also takes user input from the trackpad that is physically attached to the back of the handheld PC. Based on this input and a database of information about campus buildings, it generates the graphics that are overlaid on the real world by the head-worn display.

The application running on the handheld PC is a custom HTTP server in charge of generating web pages on the fly and also accessing and caching external web pages by means of a proxy component. The HTTP server can talk back to the tour application through a distributed server architecture.

3.2.1.2 Situated Documentaries Extensions

This architecture, which extends the COTERIE-based system architecture above, served as the basis for the Situated Documentaries UIs, which we will discuss in Section 5.1. Two main contributions are associated with these extensions to the campus tour architecture. First, we extended the above environment by providing playback and synchronization support for various new media, which made it possible to present context-dependent

```
let low1 = proc(urlobj, menu)
  prepareSounds("l",1,4);
  prepareSlides(1,2);

  urlobj.sendurl(menu, urlprefix & "low/lowh.html");

  showText(menu.data, "Columbia Students Protest Building of Gymnasium");
  playSound("l1");
  removeText();

  (* =============== DEBUG output: =================
     wr_putText(wr_stdout, "played first sound bit\n");
     wr_flush(wr_stdout);
   *)

  showSlide(1);
  playSound("l2");
  removeSlide(1);

  showSlide(2);
  urlobj.sendurl(menu,urlprefix & "videos/lv1.mpg");
  playSound("l3");
  removeSlide(2);
  showText(menu.data, "Video is playing on your browser!");

  playSound("l4");
  removeText();

  cleanupSlides(1,2);
  cleanupSounds("l",1,4);
end;
```

Figure 3.3: Example Repo script, choreographing a Situated Documentaries multimedia presentation.

narrated multimedia presentations. Second, we provided authoring tools that enabled non-programmers to put together multimedia narratives that can be presented as *in situ* AR stories. We also extended the driver architecture to better accommodate a new set of tracking devices (real-time kinematic GPS and hybrid orientation tracking) and better monitor the quality of the tracking data.

We designed and implemented system support for different media, including text, audio, still images, video, still and animated 3D graphics, and omnidirectional panoramas, providing unified Repo APIs for playback and coarse-grain synchronization. The multimedia information to be conveyed through the AR UI had to be arranged and locally distributed over the target region. For this purpose we designed several authoring tools.

To create the multimedia presentations, we developed a simple extension to the interpreted language Repo. Each multimedia presentation is stored as a Repo script, referencing by filename the multimedia "chunks" (images, video segments, audio snippets, 3D animations, omnidirectional views) it uses. Each chunk is stored on the computer (backpack or handheld) on which it is to be played; additional material to be presented on the handheld computer can be obtained from the web using a wireless network interface.

Students in two graduate Journalism classes on digital media used our prototyping environment to create multimedia narratives. While some of the students had previ-

ous programming experience, for others it was the first contact with digital multimedia programming. A short introduction to the scripting language was enough for them to assemble synchronized multimedia presentations of one to four minutes length. They collected historical multimedia footage, broke it into chunks, and wrote scripts, such as the one in Figure 3.3, to choreograph the presentation.

This specific script describes a multimedia presentation that uses audio narrations, short text messages, "slides" (still images, which enter the AR screen in a slideshow animation), and webpages, including one that plays video footage, to form a story about student revolts on Columbia's campus. The script defines a procedure, `low1`, through which the multimedia presentation can be called from other Repo code. Sounds and slides are preloaded, in order to enable instant access when it is their turn to be delivered in the presentation. The commands to show and remove various media are executed in sequence. Synchronization takes place purely at the level of the relatively coarse-grain media chunks the students created. When the script is executed, Repo messages are exchanged between the main server on the backpack computer and the HTTP server on the handheld computer, ensuring a timely execution of the presentation on the head-worn audio and video display and the hand-held web browser.

The author of this thesis provided all the mechanisms for control flow, and integrated the multimedia scripts into the "physical hypermedia" UI described in Section 5.1.1. All location-based information is stored in a campus database on the backpack computer. This database contains the complete structure of the situated documentaries, including the contents of all context-menus and links to the multimedia presentation scripts. We used an early version of a map-based tool we developed to place UI objects at specified physical locations, where they can be used to trigger multimedia stories. For this purpose, we scanned in a high-resolution map of Columbia's campus that provides a placement resolution of about six inches in latitude or longitude.

3.2.2 IN-N-OUT AR – an Indoor/Outdoor Java/COTERIE Hybrid Infrastructure

The major contribution of this system architecture is to have integrated several disjoint Java- and COTERIE-based subsystems to a coherent whole, which enables AR-, VR- and desktop-based indoor/outdoor communication. Important features of this architecture are the central database and networking server for inter-process communication and the relational database backend for persistent data storage. For the first time in our explorations, we built an infrastructure that placed a MARS in the larger framework of a collaborative service and storage infrastructure (Spohrer, 1999).

A wearable UI alone is not enough to fully capture the potential of a world-wide

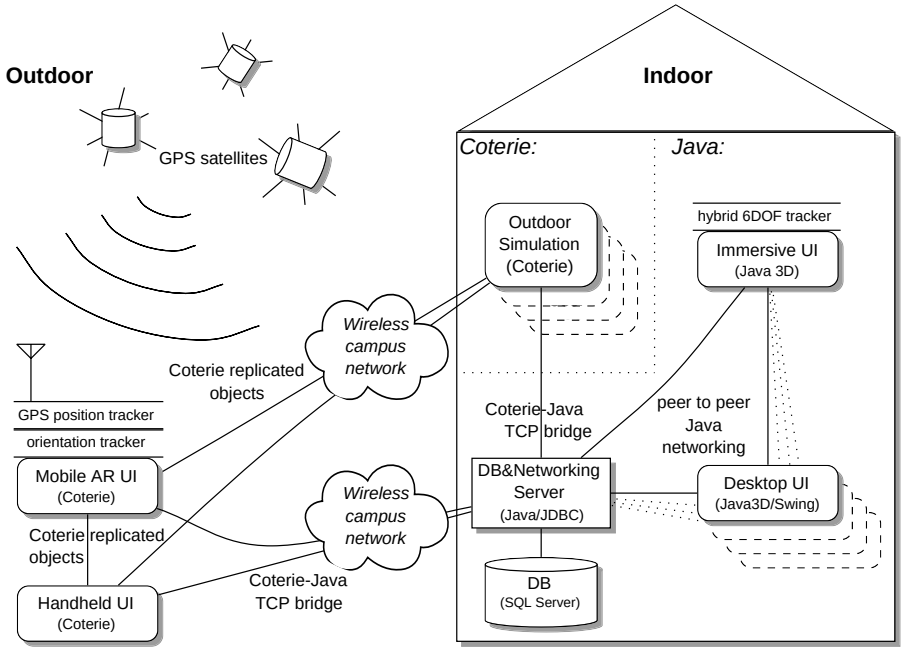


Figure 3.4: MARS 1999 distributed software architecture.

layer of spatialized information. For various tasks, a stationary computer system will be more adequate, especially for those applications whose UIs work best with physically large displays. Among these applications are tools, especially collaborative ones, for authoring the information layer, for obtaining a broad-scale overview of relevant information, and for playing back logs of user interactions with the augmented world.

The IN-N-OUT AR environment facilitates indoor–outdoor collaboration (Höllerer et al., 1999b). Indoors, a desktop or immersive UI, based on a 3D environment model, lets users create virtual objects and highlight and annotate real objects for outdoor users to see, and maintain histories of outdoor users' activities; in turn, outdoor users point out interesting objects and events for indoor users to view. We will discuss the user interface concepts we implemented on this hybrid platform in Section 5.2.1.

Figure 3.4 shows an overview of the system architecture for this heterogeneous MARS environment. The following four types of UI clients are interconnected through the networking infrastructure and have access to the database backend: a COTERIE-based outdoor MARS of a similar architecture as described in the previous section; an arbitrary number of COTERIE-based outdoor MARS simulation engines, running on desktop computers; an indoor desktop UI implemented in Java, and a stationary immersive VR or AR UI, implemented using Java and Java3D.

Tachio Terauchi implemented the Java-based indoor UIs, and Drexel Hallaway was the main architect of the UDP-based peer-to-peer communication infrastructure

(TCP-based for the COTERIE–Java link). The author of this thesis authored formulated the overall system design, authored the COTERIE-based components, and integrated the different parts to a working infrastructure.

As emphasized in Figure 3.4, all applications in our testbed have access to a main database that contains a model of the physical environment and all the virtual information that has been added over time. When each of our current UIs is started up, it reads the most recent state of the database. Internally, the data is organized in a relational database, currently maintained using Microsoft SQL Server. A database server process, written in Java using the JDBC API, provides client processes (multiple users and UIs) with access to this data. To make this possible, we developed a client-server database access protocol.

Not surprisingly, the latency of these calls is too great for real-time graphics updates (e.g., rendering a moving outdoor user in an indoor system). To address this, Drexel Hallaway's peer-to-peer communication infrastructure emulates a distributed shared memory model for the objects that need to be updated rapidly.

As far as the outdoor MARS is concerned, our hand-held display, upgraded in processing power, could now run a map-based UI coded in COTERIE, alternatively to the web-browser UI from Section 3.2.1.1. It could now be used either in conjunction with the backpack, or standalone.

3.2.2.1 Development Tools

We developed several tools and techniques to make the development and testing of new collaborative outdoor and indoor interfaces easier and more efficient.

To test new outdoor UI components without actually having to take the backpack system outside, we designed a COTERIE application that simulates an outdoor user indoors (cf. Figure 3.4). This program runs in two modes: free navigation mode on a common 2D display, which supports mouse-based navigation over the whole terrain with controls similar to first-person action games, and immersive look-around mode, which uses the same head-worn display and orientation tracker we use outdoors. For this latter mode, we assume a static position in our environment and use either a 360° omnidirectional image taken from that position as a "backdrop" (Höllerer et al., 1999a) or our 3D environment model.

Since both real and simulated outdoor users are treated alike, as far as interaction with other processes is concerned, this allowed us to do tests with multiple roaming users, even though we only maintained one physical backpack system at a time.

Under the author's supervision, Gus Rashid developed an authoring tool for offline creation of new environment models and placement of new 3D geometry. This is different from the online interaction mechanisms possible in the indoor desktop and immersive UI in that it is intended for creating base models of new environments. The

offline tool uses a 2D map of the area to be modeled with longitude/latitude coordinates. It allows us to trace over the 2D map, which is loaded as a background image, and offers the geometrical primitives of a typical 2D drawing program. Arbitrary 2D outlines drawn by the user can be extruded into the third dimension to create simple models of buildings and saved for use in our MARS environment.

3.2.3 JABAR — A Java-Based AR Infrastructure

JABAR is our first MARS architecture entirely built on Java. Making a transition to a Java-based AR system had already been planned when we started work on the IN-N-OUT infrastructure. The main disadvantage of our COTERIE platform was, and had always been, the small user base of Modula-3, and the resulting lack of further library development, the difficulties in maintaining our own libraries for use on new hardware platforms, and difficulties in convincing project students to learn several new programming languages (Modula-3 and Repo). Java brought with it the momentum, third-party libraries, cross-platform compatibility, and programming ease to make it a very attractive programming environment. The only open issue was the performance of interpreted Java byte code when comparing it to the speed of natively compiled code. Java's just-in-time compilation technology removed this last obstacle. In a comparison of interactive graphics applications, a solution based on Java 1.3 performed better than the same graphical scene implemented using Repo and the COTERIE environment. We were aware that with C or C++ we would have been able to gain some extra speed even on top of our Java results, but because of the advantages of Java mentioned above, and the considerable Java driver library that we had already developed for some of our tracking and interaction hardware, we committed to Java. One of Java's features is automatic memory allocation management via so-called garbage collection. In early Java implementations this would have led to noticeable delays in rendering at irregular intervals, but with the 1.3.1 Java Virtual Machine, built-in support for incremental garbage collection resolved most of the issues, and was further improved by three new garbage collection algorithms in Java release J2SE 1.4.1.

For our JABAR (JAva-Based AR) architecture, we reused and extended code from the IN-N-OUT environment, in particular the object distribution architecture which continued to be maintained and improved by Drexel Hallaway, and the device drivers we had already developed for our indoor interfaces. Ryuji Yamamoto contributed camera support that allowed snapshots of annotated world scenes to be taken and sent to other clients.

Figure 3.5 shows the hierarchy of libraries for this architecture. On top of the Java SDK we rely on Java 3D and the Java Media Framework (JMF) as the main support

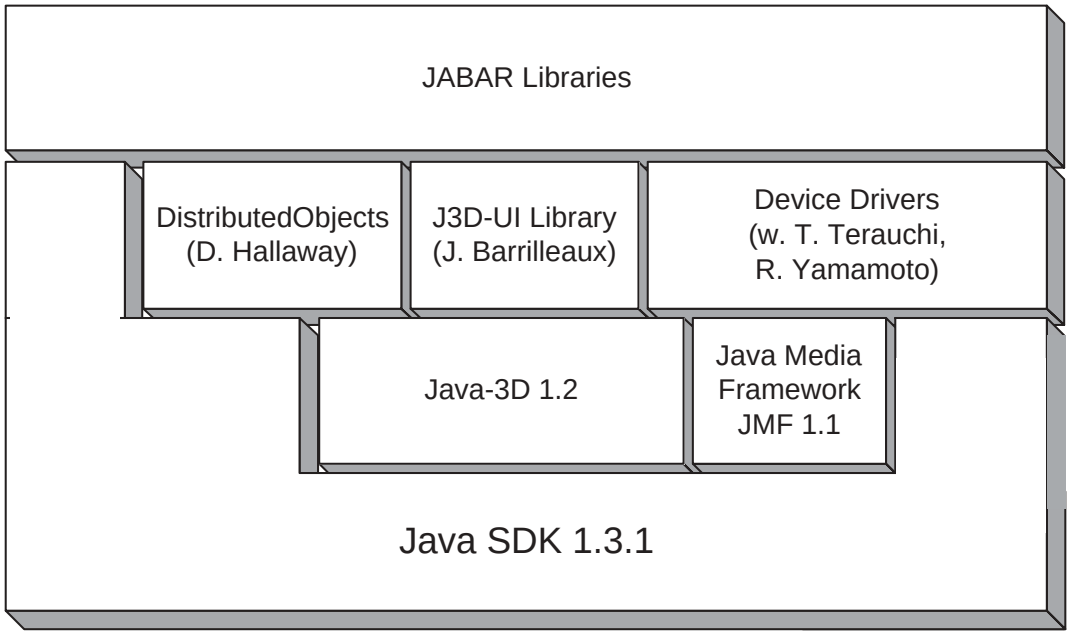


Figure 3.5: JABAR library hierarchy.

libraries to enable us to re-implement the kind of multimedia AR interfaces we had explored using the COTERIE architecture. We also make use of a new framework of 3D interaction techniques, implemented on top of Java 3D (Barrilleaux, 2000). This library offers some interesting world- and display-space visualization and interaction support, but has the disadvantage that it is fairly complex and hard to integrate with other libraries. Therefore, we decided to just use a small subset of the offered functionality, and implement our own visualization and interaction infrastructure with a focus on AR requirements. This is represented as the top box labeled 'JABAR libraries' in Figure 3.5.

JaBAR also provides support for an integrated web browser that can pop up in the AR view, in response to selections in the 3D scene.

All in all, this new architecture supersedes the capabilities of the COTERIE-based development environment in many respects, and has proved perfectly usable in the areas where the COTERIE environment had had advantages (e.g., in data distribution mechanisms). The new capabilities led to several new MARS interface mechanisms we designed, described in Sections 5.1.2 and 5.3.1.

3.2.4 CGUI Lab AR — A Central Architecture for AR and View Management

Our efforts on JaBAR merged with other ongoing efforts in the CGUI Lab. This new joint infrastructure, implemented to a large extent by Blaine Bell, was the basis for our work on view management for virtual and augmented reality (Bell et al., 2001).

One of its most important features was the way it circumvented flaws in Java 3D's view model by discarding the view model's HMD mode and re-implementing that functionality using the standard desktop mode. We adapted our tracker drivers for use with this generalized view model approach.

3.2.5 Ruby — Rule-based Control of AR interfaces

One major lesson we learned in our experiments with mobile AR interfaces was that static interfaces are often inadequate. Too many situations can arise that have an impact on UI effectiveness in an unforeseen fashion. For example, the more annotations augment the real world, the more likely it becomes that these annotations interfere with each other, partially overlap, and appear to annotate the wrong physical object. As another example, changes in the availability of resources can disrupt a UI and render it useless. An adaptive interface could counteract these problems. For example, if the user lost GPS tracking capabilities upon entering a building, the system should be able to adapt and present a UI on the head-worn display that does not require this kind of position tracking.

In order to make dynamic decisions about UI composition and layout, we decided to take a rule-based approach, making use of an established forward chaining expert system algorithm, RETE (Forgy, 1982), and in particular, its implementation in the Java Expert System Shell (Jess) from Sandia National Laboratories (Friedman-Hill, 1998). This architecture radically shifted the control of the user interface from application-dependent control code to dynamic rules governing over the consistency of the UI as a whole.

This is a major departure from the previous architectures, which had a decidedly imperative programming language approach to UI design: There, the UI was hardcoded for a particular application, and changing situations had to be anticipated and provided for by means of nested if-statements in the main behavior loop. Figure 3.6 highlights the central knowledge-based core functionality of the new architecture: The Jess knowledge base, a library of rules, the knowledge manager API that forms the interface to much of the rest of the infrastructure and the application code, and the object/scene directory that holds representations of all the objects playing a role in the AR user interface. These objects change their state depending on rules firing in the knowledge base, and callbacks cause the desired changes to be reflected in the UI.

The whole architecture will be described in detail in Chapter 6 after we have in-

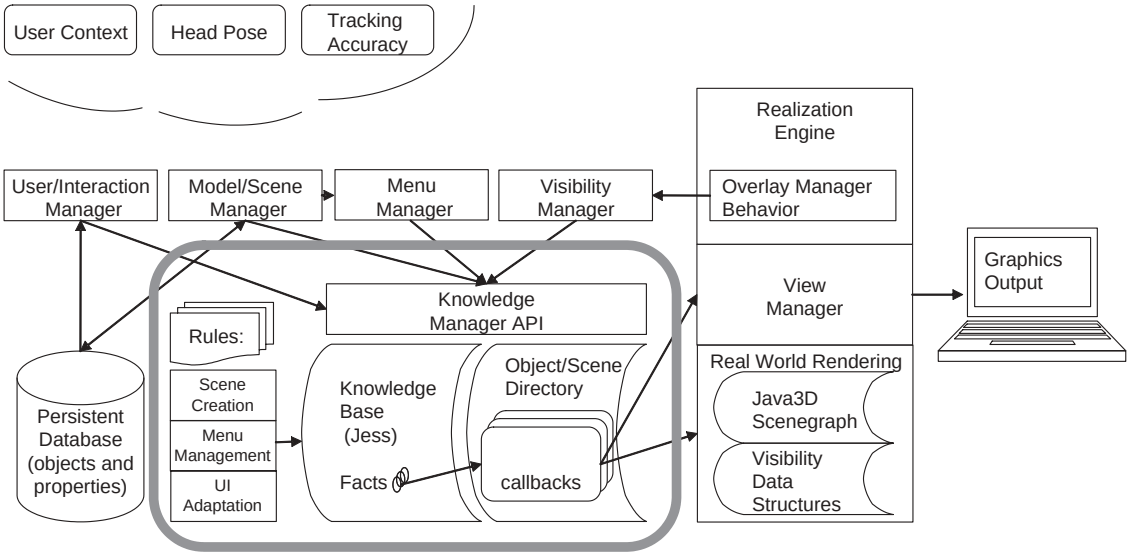


Figure 3.6: Ruby rule-based software architecture.

troduced our taxonomy of MARS UIs in Chapter 4 and presented a series of increasingly adaptive user interfaces in Chapter 5.

3.3 Indoor Tracking for MARS

In this section we present research results on tracking approaches for indoor MARSs. This work explored how to track a user in the hallways and laboratories of our research building, based purely on worn sensors and knowledge of the environment. The author of this dissertation designed and iteratively refined the overall tracking concept and algorithms, which were implemented and tested by Navdeep Tinna, using data structures and geometrical models provided by Drexel Hallaway.

For this work we used the MARS 2000 hardware described in Section A.3, omitting the differential GPS hardware to obtain a smaller system for indoor-only use. Orientation tracking is done with an InterSense IS300 Pro hybrid inertial/magnetic tracker. We can track both the user's head and body orientation by connecting head-mounted and belt-mounted sensors to the unit. When walking around indoors, we have to switch off the magnetic component of the tracker to avoid being affected by stray magnetic fields from nearby labs (see Section 3.3.1). In that case we obtain orientation information from the inertial component of the tracker.

For indoor position tracking, we use a Point Research PointMan Dead-Reckoning Module (DRM) and an InterSense IS600 Mark II SoniDisk wireless ultrasonic beacon. The system can detect whether the beacon is in range of an InterSense IS600 Mark II

ceiling tracker. The tracker is connected to a stationary tracking server and the position updates of the roaming user's SoniDisk beacon are relayed to the user's wearable computer using the Java-based distributed augmented reality infrastructure described in Section 3.2.2. Navdeep Tinna developed the driver support for the DRM, building on earlier code by Phil Gross.

Our augmented reality user interface for navigational guidance, co-developed by Drexel Hallaway, adapts to the levels of positional tracking accuracy associated with different tracking modes (Höllerer et al., 2001b). We will show examples of the user interface in Section 5.3.1. In the following we present our indoor dead-reckoning tracking approach.

3.3.1 Wide-Area Indoor Tracking using Dead-Reckoning

Whenever the user is not in range of an appropriate ceiling tracker, our system has to rely on local sensors and knowledge about the environment to determine its approximate position. Unlike existing hybrid sensing approaches for indoor position tracking (Golding and Lesh, 1999; Van Laerhoven and Cakmakci, 2000; Clarkson et al., 2000), we try to minimize the amount of additional sensor information to collect and process. The only additional sensor is a pedometer (the orientation tracker is already part of our mobile augmented reality system). Compared with Lee and Mase (2001) who use digital compass information for their heading information, we have a much more adverse environment to deal with (see discussion below). Therefore, we decided to rely on inertial orientation tracking and to correct for both the resulting drift and positional errors associated with the pedometer-based approach by means of environmental knowledge in the form of spatial maps and accessibility graphs of our environment.

Our dead-reckoning approach uses the pedometer information from the DRM to determine when the user takes a step, but uses the orientation information from the more accurate IS300 Pro orientation tracker instead of the DRM's built-in magnetometer. We do this because the IS300 Pro's hybrid approach is more accurate and less prone to magnetic distortion. Furthermore, we have the option to use the IS300 Pro in inertial-only tracking mode. Figure 3.7(a) illustrates the problems that our indoor environment poses for magnetometer-based tracking. The plot corresponds to a user walking around the outer hallways of the 6th floor of our research building, using the IS300 Pro tracker in hybrid mode. The plot reflects substantial magnetic distortion present in our building. In particular, the loop in the path on the left edge of the plot dramatically reflects the location of a magnetic resonance imaging device for material testing two floors above us.

For indoor environments with magnetic distortions of such proportions, we de-

cided to forgo magnetic tracker information completely and rely on inertial orientation data alone. Figure 3.7(b) shows the results for a user traveling the same path, with orientation tracking done by the IS300 Pro tracker in purely inertial mode. The plot clearly shows much straighter lines for the linear path segments but there is a linear degradation of the orientation information due to drift, resulting in the "spiral" effect in the plot, which should have formed a rectangle.

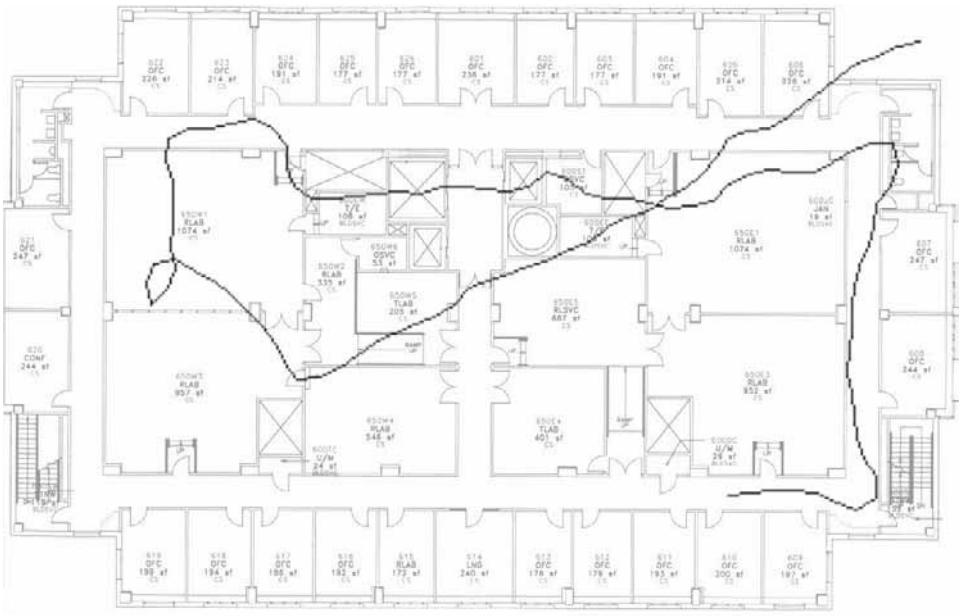
Figure 3.8(a) and (b) show the results after correcting the method of Figure 3.7(b) with information about the indoor environment. Plot 3.8(a) shows a similar path through the outer hallway as those of Figure 3.7. In contrast, plot 3.8(b) shows an "S"-shaped path from our lab door at the southeast, around the outside hallway at the east and north, down through the center corridor to the south hallway, then heading to and up the west hallway, and across the north hallway back to the north end of the center corridor. To perform these corrections, we use two different representations of the building infrastructure in conjunction: spatial maps and accessibility graphs.

Spatial maps accurately model the building geometry (walls, doors, passageways), while *accessibility graphs* give a coarser account of the main paths a user usually follows. Figure 3.9 compares the two representations for a small portion of our environment. Both the spatial map and the accessibility graph were modeled by tracing over a scanned floor plan of our building using the modeling program mentioned at the end of Section 3.2.2.1.

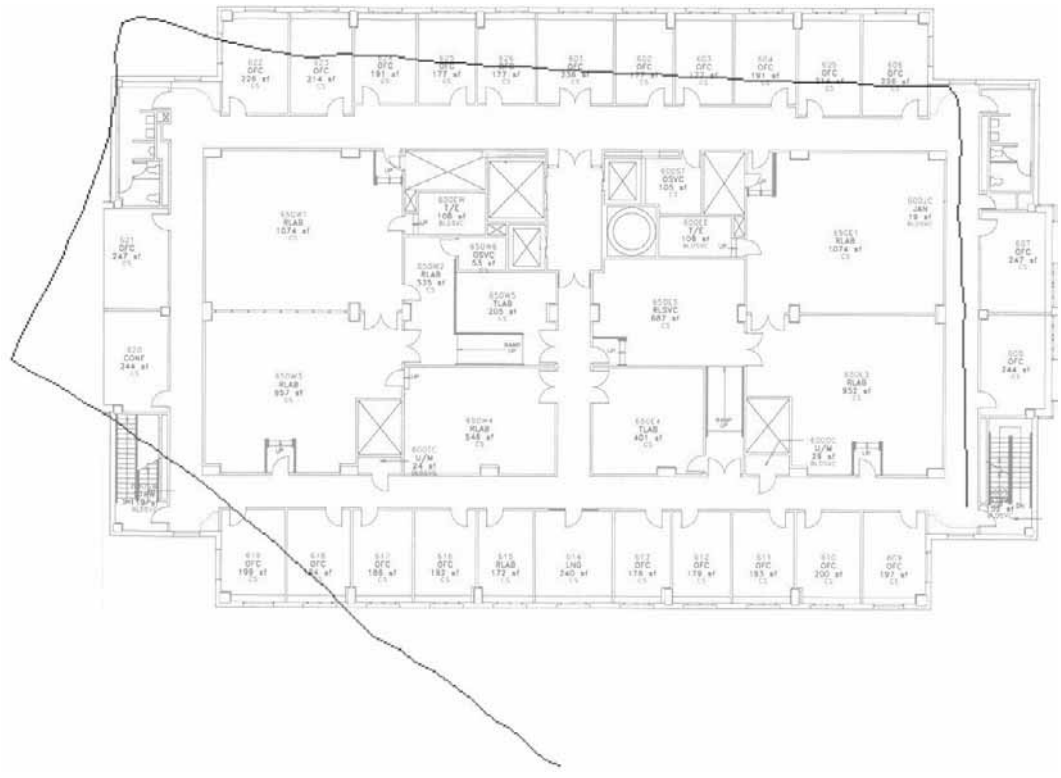
The spatial map models all walls and other obstacles. Doors are represented as special line segments (as denoted by the dashed lines connecting the door posts). In addition to its role in tracking correction, the accessibility graph is also the main data structure used by the path planning component mentioned in Section 5.3.1.

For each step registered by the pedometer, and taking into account the heading computed by the orientation tracker, our dead-reckoning algorithm checks the spatial map to determine if the user will cross an impenetrable boundary (e.g., a wall). If that is the case, then the angle of collision is computed. If this angle is below a threshold (currently 30°), the conflict is classified as an artifact caused by orientation drift and the directional information is corrected to correspond to heading parallel to the obstacle boundary.

If the collision angle is greater than the threshold, the system searches for a segment on the accessibility graph that is close to the currently assumed position, is accessible from the currently assumed position (i.e., is not separated from it by an impenetrable boundary, which is checked with the spatial map data structure), and is the closest match in directional orientation to the currently assumed heading information. The system assumes that the user is really currently located at the beginning of that segment and changes the last step accordingly to transport the user there.

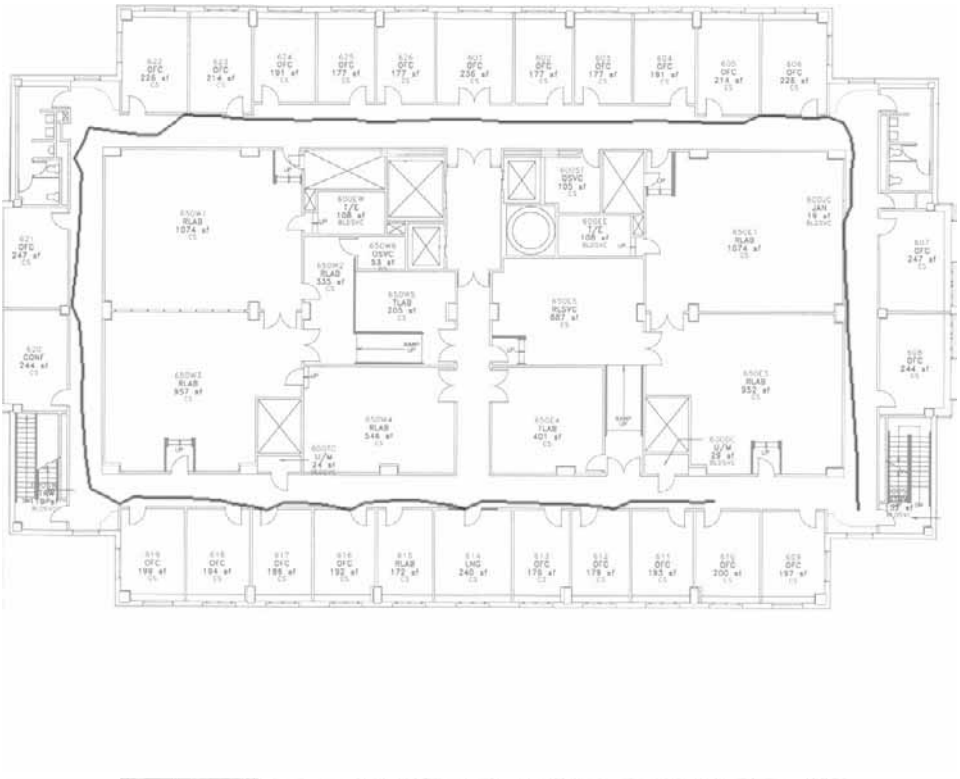


(a)



(b)

Figure 3.7: Tracking plots using the DRM in our indoor environment. (a) Pedometer and magnetic orientation tracker. (b) Pedometer and inertial orientation tracker.



(a)

(b)

Figure 3.8: Tracking plots using the pedometer, inertial orientation tracker, and environmental knowledge. (a) Path around the outer hallway. (b) More complicated path, passing through doors.

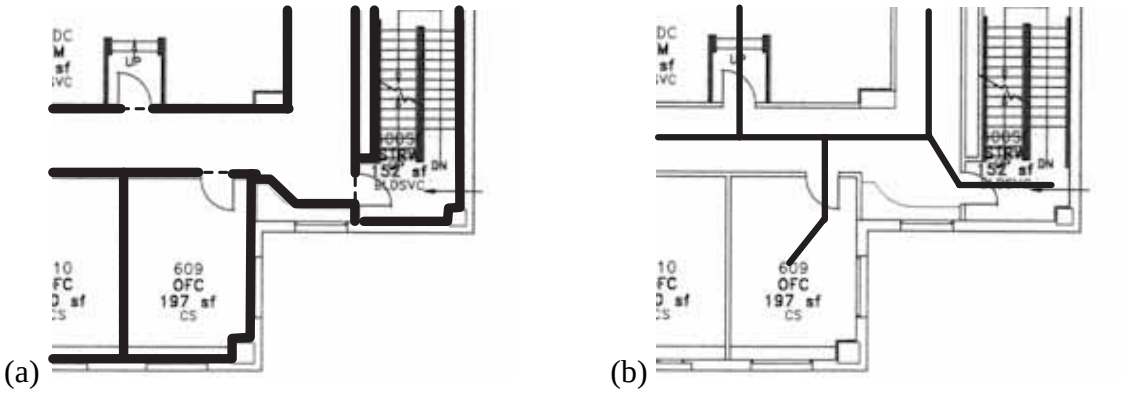


Figure 3.9: Two different representations of a small part of our building infrastructure, as used in the dead-reckoning–based tracking approach: (a) Spatial map. (b) Accessibility graph.

Doors are handled as special cases. First, the sensitive door area is assumed to be larger than the doorframe itself (currently, all walls in the immediate continuation of the door 1 m to either side will trigger door events if the user attempts to cross them). In case of a door event, the angle of collision is determined. If the angle is below our 30 degree threshold, the system behaves as if the door were a simple wall segment and no passage occurs. If the angle is greater than 60 degrees, the system assumes that the user really wanted to enter through that door and proceeds correspondingly. If the angle is in between the two thresholds, the system continues with the accessibility graph search described above.

Our results with this approach were very promising. The plot in Figure 3.7(d) for example corresponds to a path along which the user successfully passed through three doors (the lab door at the east end of the south corridor, and two doors at the north end and middle of the center corridor), and never deviated far from the correct position. The rules concerning drift correction and passing through doors work well for the described environment, as evidenced in numerous test runs. We have not applied the method yet in any other environments.

Chapter 4

Analytical Foundation of MARS UIs

In this chapter we present a taxonomy of MARS UI components. In doing so we keep several goals in mind: First, to provide a theoretical foundation of MARS interfaces. Second, a taxonomic overview of the concepts and interface components relevant to MARSs enables researchers to better understand the system and UI domain of this interface paradigm. Third, and most importantly, we designed this taxonomy to be implementable. What we mean by that is that the categories and attributes in this taxonomy form a specific knowledge representation that enables MARS interfaces to maintain semantic information about their components and purpose and allows for knowledge-based adaptation of the UI in dynamic situations. Chapter 6 will present Ruby, our rule-based implementation of such an adaptive system, which formalizes large parts of the taxonomies presented here.

We begin by discussing in Section 4.1 the scope of MARS UIs and the main properties that set them apart from non-AR situated and mobile interfaces on one hand, and from VR interfaces on the other. In Section 4.2, we present our taxonomy of MARS UI components. We discuss MARS objects and their attributes, give a broader view of MARS concepts in general, and end with a discussion and categorization of MARS interaction.

4.1 Scope of MARS User Interfaces

The kind of computer interaction that MARSs make possible is quite different from what we are used to in the computing environments of today's office or home environments.

There have been various explorations of alternatives to the classic WIMP, which all offer their own respective advantages on certain computing platforms and for certain application areas for which WIMP interfaces can be inefficient to use. Figure 4.1 gives an overview of some of the avenues that researchers are exploring, and shows how MARS

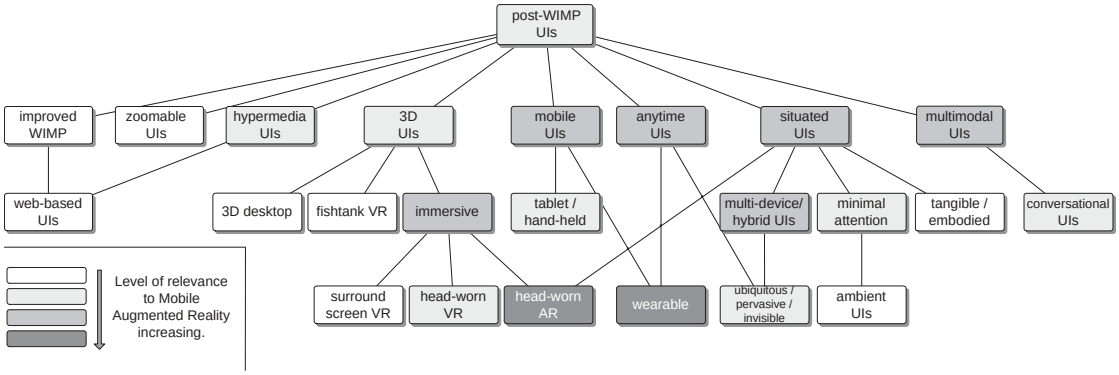


Figure 4.1: Research areas of relevance to MARS UIs in the hierarchy of post-WIMP UIs

interfaces fit in with these lines of UI research.

As indicated in the figure, MARS UIs combine elements from immersive 3D UIs with mobile and multimodal UIs. MARSs are examples of *situated* or *context-aware* user interfaces, meaning that they have the ability to detect, interpret, and respond to aspects of the user's local environment (Hull et al., 1997; Beadle et al., 1997). MARS UIs provide a particularly immediate interface for context-aware computing, linking the physical world in front of a mobile person to computer augmentations.

Many of the features that distinguish MARS UIs from desktop interfaces are related to MARS's kinship to mobile, situated, anytime, and especially wearable (Rhodes, 1997) computing on one hand, and immersive 3D graphics on the other:

- They are always immediately available to the user, independent of the user's current location or occupation.
- They are location-aware and can provide links to the physical world surrounding the user.
- They can react to head gestures.
- Information can be *registered* with real-world objects, resulting in world-stabilized annotations.
- Due to head-tracking and world-stabilized annotations, the screen composition is much more dynamic than for stationary UIs, where all changes in screen composition are usually consciously controlled by the user.
- The use of near-eye displays enables instantaneous and inconspicuous information access.

- An infinite display space surrounds the user, but only a small window into this space is visible at each instant.

- The display space has an inherently 3D component, since the real world is three-dimensional.

The affinity of MARS UIs with VR interfaces becomes especially apparent when it comes to interacting with computer-generated material. Research in VR, which immerses a user in completely computer-generated environments, has a long history of exploring and categorizing the possibilities of 3D interaction (Hinckley et al., 1994a; Poupyrev et al., 1996; Bowman and Hodges, 1997; Pierce et al., 1997; Mine et al., 1997; Pierce et al., 1999; Bowman, 1999).

In contrast to MARSs, however, VR simulations mostly take place in controlled environments where the user's real-world mobility is restricted. VR simulations aim to replace the real world with computer graphics in a realistic fashion. The graphics are commonly viewed on tracked head-worn displays, or on stereo projection displays, such as a surround-view CAVE (Cruz-Neira et al., 1993), or a Responsive Workbench (Krueger and Froehlich, 1994). The most important properties that set MARS UIs apart from VR interfaces, can be listed as follows:

- The UI consists of both virtual and physical elements, and virtual material is registered with the physical world.

- Positional proximity to physical objects can trigger changes in the virtual UI layer.

- The user roams about freely in large physical environments.

- Since the computer graphics elements annotate the physical world rather than replace it, image realism plays a less important role.

- MARSs have to work in non-controlled environments (weather, lighting, background noise, constraints imposed by social situations, ...)

- Cameras might be used for image registration and human-to-human communication purposes.

- MARS applications need to stay informed about the current physical environment (e.g. via dynamic information retrieval from application servers).

- Tracking accuracy may vary substantially depending on available resources.

With MARSs, the world becomes the interface, which means that in dealing with mobile AR we rarely focus exclusively on the computer anymore. In fact, while we go about our day to day activities, most of the time we would not be able, let alone want, to pay attention to the computer. At the same time, however, we will expect the system to provide assistance and augmentation for many of our tasks. In (Broll et al., 2001) we describe a futuristic scenario of using such an unobtrusive mobile helper interface.

Mobile AR agrees well with the notion of non-command interfaces (Nielsen, 1993), in which the computer is reacting to sensed user context rather than explicit user commands. For a person trying to focus on a real-world task, and not on how to work a particular computer program, it is desirable that computers understand as much as possible about the task at hand without explicitly being told. Often, much of the needed interaction can be reduced to the user's selecting one of several prompted alternatives, leading to a Minimal Attention User Interface (MAUI) (Pascoe et al., 2000). Kristoffersen and Ljungberg (1999) suggest three principles for mobile palmtop computer interfaces: Little or no visual attention; structured, tactile input (in their case using four hardware buttons); and audio feedback. In mobile AR UIs we can operate with more visual feedback, since the division of attention between the physical world and the computer display is much less disruptive. In fact, in principle it is possible to focus simultaneously at a real-world scene and computer-generated feedback in form of world-stabilized annotations.[1] Schmidt and colleagues (2000) propose a mobile AR interface that presents menus and reflects user choices in display segments at the periphery of the user's field of view.

Many tasks on mobile computing platforms can be completed applying the principles of MAUIs. Sometimes, however, it is desirable to focus on and interact with computer-generated information directly. Here, MARS UIs reveal their kinship to VR UIs by providing 3D interaction techniques that facilitate full control over world-placed virtual objects. Tasks that require such extended user interaction with the AR interface include, for example, placing or moving virtual objects in the environment, or modeling them from physical examples.

Interfaces that have been tried for such tasks range from using a 2D cursor and head motion (Baillot et al., 2001) to a tracked glove (Thomas and Piekarski, 2002), to exerting control indirectly using a tracked graphics tablet, on which interface elements can be overlaid (Reitmayr and Schmalstieg, 2001a). Simple interaction with virtual material has also been achieved using vision-based hand tracking (Kurata et al., 2001).

Mobile AR interfaces invite collaboration. Several users can discuss and point

[1] Shortcomings in tracking accuracy and display technology can hamper experiencing virtual annotations as really being part of a physical environment. Currently available head-worn displays, for example, cannot automatically adjust focus planes depending on where a person is looking. Still, the user can switch back and forth between the head-worn display context and the real world easily and without invoking head motion.

to virtual objects displayed in a shared physical space (Billinghurst et al., 1997; Butz et al., 1999; Reitmayr and Schmalstieg, 2001a). Still, every participant can view their own private version of the shared data, for example to see annotations optimized for their specific viewing angle (Bell et al., 2001). Multiple users can collaborate in the field, and remote experts with a top-down overview of the user's environment can communicate and share information with the field worker (Höllerer et al., 1999b).

In order to optimize the UI for different user contexts, we want the mobile AR system to have knowledge about the UI components that form the interface at any given moment, and of the alternatives and flexibilities in placement and appearance. In this chapter, we will categorize MARS UI components using type, state, semantic, and dynamic constraint attributes. In deriving such a taxonomy, we will draw from our own experience in designing MARS user interfaces, as well as from existing research in the field. Our objective for developing such a taxonomic formulation is to explain the working principles of current MARS interface techniques, including the interface methodologies that will be described in Chapter 5. This is the first step in building user interfaces that are aware of the type and function of their components, and that can adapt to contingencies encountered during run-time by rearranging their composition dynamically. Chapter 6 describes an implementation, in which part of this taxonomy is formalized using a knowledge-base data format for representing UI components and their properties. A rule-based reasoning engine monitors changes in the user's context and causes the MARS UI to automatically adapt to dynamic situations.

4.2 Taxonomy of MARS UI components

Over the years the HCI community has developed taxonomies in order to formalize various kinds of UIs. Many taxonomies related to visualization and graphical design refer back to Bertin (1983), who defines basic elements of information graphics and their properties in a systematic classification of visual elements to display data and relationships. He proposes a visual semantics for linking data attributes to visual elements.

Foley et al. (1984) focus on computer graphics subtasks and interaction techniques and review experimental evaluations of techniques based on different input devices. Buxton (1986) critiques and extends the Foley taxonomy; for example, to better mirror the cognitive relevance of physical gestures. Card et al. (1991) analyze the design space of input devices.

During the 1980s and early 90s, when WIMP UIs became fairly standardized, which was partly due to the amount of analytical exploration invested in them, the concept of user interface management systems (UIMS) became popular (Kasik, 1982; Olsen, Jr. et al., 1987; Green, 1987; Olsen, Jr., 1992). The ultimate goal of UIMSs was to au-

tomate the production of WIMP-style user interface software. While that goal proved to hard to achieve in all completeness, taxonomies of WIMP UI components lead to the development of various 2D interface toolkits and development environment that cemented the success of the WIMP interface (Marcus (1992) gives an overview).

Myers (1988) presents a taxonomy of window manager user interfaces. Some researchers have begun to tackle the 3D domain and have started classifying 3D widgets and higher level interaction techniques (Green, 1990; Conner et al., 1992; Hinckley et al., 1994b; Jacob et al., 1999).

Milgram and Kishino (1994) and Milgram and Colquhoun Jr. (1999) explore taxonomies of *Mixed Reality* (MR) systems, a generalization of AR systems that also includes applications which are set predominantly in the virtual domain, possibly enriched by real life imagery (VR and *Augmented Virtuality*). Their taxonomies try to classify whole MR systems according to, among other things, their degree of virtuality, the extent of world knowledge they posess, their reproduction fidelity, and their extent of presence metaphor. They do not discuss specific MR UI components at all.

Research on the automated design of information presentations has produced extensive taxonomies of graphical design, extending the categories of Bertin (1983). In particular, Feiner (1985) addresses the case of pictorial explanations, Mackinlay (1986) explores the use of multiple encodings for graphical presentations, Seligmann (1993) classifies intent-based illustration techniques, and Zhou and Feiner present classifications for heterogeneous data and atomic visualization tasks (Zhou and Feiner, 1996; Zhou and Feiner, 1997).

As the first step on the road to a taxonomy of MARS UI components that can facilitate the automatic management of information spaces, we need to characterize the objects that define a MARS UI.

4.2.1 MARS Objects

MARS objects are the entities that form the AR user interface. As shown in Figure 4.2, we distinguish between objects that have a perceptible embodiment (*environment objects*), objects that represent blueprints for virtual environment objects (*UI templates*), and *container objects* that group and organize other objects into collections.

Objects can be real or virtual, or they can be abstract concepts that help to structure the world. They are either atomic or composite. They have attributes describing their type, state, semantics, and realization parameters. Section 4.2.2 will present a complete picture of the properties of MARS objects.

Environment objects are perceptible objects in the augmented environment surrounding the mobile user that are sufficiently relevant to the user or to other objects in

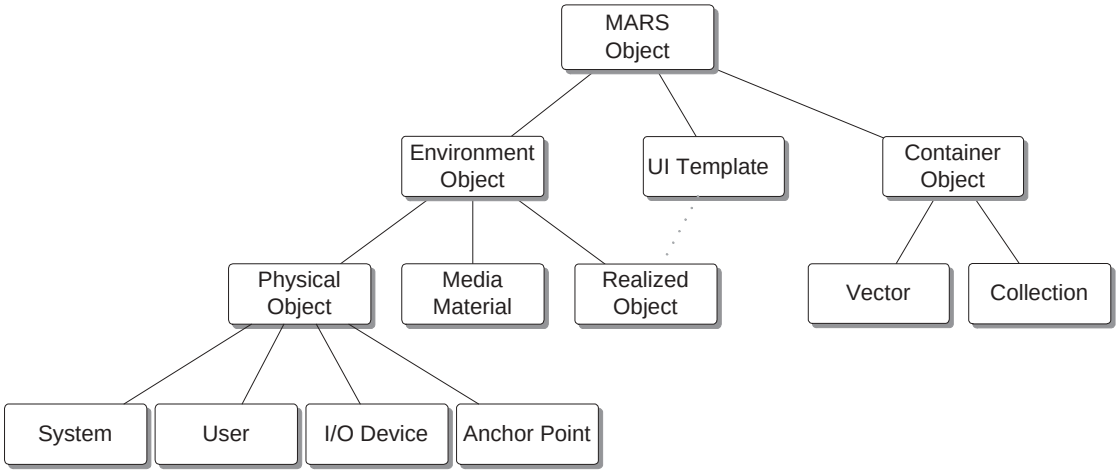


Figure 4.2: MARS Objects. The dotted line signifies that every realized object is based on a UI template. All other lines denote inheritance.

the environment to be represented by the system. They can be physical objects in real space or virtual objects that appear on different display technologies (e.g., AR glasses, various 2D and 3D displays, or audio displays). All of these objects share the property of having a perceptible embodiment, either in real life or on one of the output devices that are part of the system.

UI templates represent the blueprints for virtual objects that the system creates as part of the user interface. All the virtual UI elements, such as menus, labels, or audio messages originate from UI templates that describe the general structure of these elements.

Container objects serve as explicit representations for groups of objects, such as a set of menu items or a multimedia story collection.

In the following subsections we will look at the different subcategories shown in Figure 4.2.

4.2.1.1 Environment Objects

An environment object is an object that the MARS actively represents and that has a perceptible embodiment in the augmented environment, either in the real world, or in any of the output media the MARS supports. Environment objects can play an active part in the UI and can get annotated or referenced. Environment objects are further subdivided into *physical objects*, *media materials*, and *realized objects*.

Physical Objects Physical objects are real objects in the environment, about which the MARS has some information. These object representations are created from entries

in the MARS's data repository (cf. Section 3.2). The amount of information stored with these objects depends on how accurately the physical world was modeled. In most cases, the MARS knows at least the physical object's location and dimensions. Not every physical object that is visible through the head-worn display of a MARS is an environment object. Objects that the MARS does not know about escape this category. In absence of sensory inputs and sensemaking tools, a MARS can only represent what it is informed about.

Realized Objects Realized objects are virtual environment objects. They form the virtual layer of the AR interface. Every virtual element presented to the user on one of the MARS's output devices is either a realized object or part of a realized object. Realized objects are concrete instantiations of *UI templates* (see Section 4.2.1.2). Realized objects can be stored in the data repository as well, in which case they become *persistent realized objects*, as opposed to the volatile realized objects that are simply part of a specific interface and discarded when that UI has served its purpose. Examples for *persistent* realized objects are the situated documentaries flags of Section 5.1.1, which denote virtual points of interest in a physical environment. These objects are preserved across separate runs of the program and may even be shared by other programs.

Media Materials The elements of the media materials object category represent multimedia content stored in the main data repository. Media materials are raw media snippets, such as 3D models, images, unformatted text, audio or video segments. They can be used by the MARS to form *realized* virtual objects by combining them with the blueprints from the *UI template* category.

4.2.1.2 UI Templates

UI templates describe the general structure of UI components. For each *realized object* there is a UI template that describes the overall format of that component. All virtual elements that the MARS creates have a formal description in the UI template category. UI templates are the blueprints for realized objects, in that they describe the overall structure of a component without specifying the specific content and context.

Consider the example of a screen-stabilized in-place menu offering choices about a physical object, say a statue in the real world, connecting the menu with the statue using a leader line. The UI template for such a menu encodes the overall functionality of the menu (e.g., screen-stabilized set of menu items realized on top of a semi-transparent polygon with a leader line pointing to an anchor position in the real world), but provides parameters, such as the number of menu entries, size and appearance of the menu, default screen position, etc., to be filled in by the MARS when the menu is created. Content and

context materials, such as the text for the menu items, the actions that are executed when menu items are selected, and the anchor location (the statue), are also not part of the UI template but have to be filled in by the system when creating the realized object.

4.2.1.3 Container Objects

The *container object* category is used to express the concept of groups or collections of objects in UI templates, and thereby also in realized objects. Container objects represent groupings of objects either as an unordered *collection*, such as the set of all interactive elements (buttons, sliders, menus) on a specific panel, or as an ordered *vector* of objects, such as the items in a menu, or an ordered set of multimedia stories. Grouping objects in vectors implies a sequential order among the elements. In order to rearrange the listing of objects, one changes the order within the vector that represents the list.

In general, the ability to group objects has the benefit that property changes such as re-colorings can be applied to the whole group instead of single elements at a time.

4.2.1.4 Special Objects and Examples

In this subsection we point out some specific objects that play a special rôle in the MARS UI. We also illustrate the function of different object categories through UI examples.

Special Physical Objects There are a few physical objects that stand out among the set of environment objects. Among these special objects are the *user* object, the *system* object, the sets of *input and output device* objects, and the concept of an *anchor point*.

User objects represent MARS users. On one hand these are represented as ordinary MARS objects, but on the other hand there is much additional information that can be stored to describe the user's context and preferences. If the MARS supports dynamic user modeling (Kobsa, 1990), the user model is stored here, too.

Input and output devices are physical objects, and as such are part of the augmented environment. Additional information that is stored with the entries representing these devices includes the type of input/output they afford, their intrinsic parameters (e.g., screen size and resolution for displays), and perhaps their usage patterns. Input and output devices are discussed in more detail in Section 4.2.4.

The *system* object keeps track of the current configuration of the MARS. A mobile system can flexibly change its configuration while it is being used. New devices, such as a nearby wall-sized display, can be dynamically added to the set of available resources. The system object also keeps track of resources such as tracking availability and remaining battery power for all devices currently part of the system, as long as such information can be determined by the MARS.

Anchor points are a borderline case in the category of physical objects. They represent coordinates in the physical world. As such, they do not actually represent perceptible entities. However, because of the parallel usage of anchor points and objects as the targets of links or annotations, it is convenient to represent anchor points as environment objects, instead of as an abstract concept.

Example Media Materials Media materials are stored in the data repository as context relevant information that can be presented to the user when the need arises. These media bits are stored in the format of raw media snippets, such as 3D models, images, unformatted text, audio snippets, video sequences, etc. No formatting or presentation information is stored with them. In order to present any of these media bits to the user, the system has to pick a UI template to combine them with. For example, images can be displayed in AR in a multitude of ways: as a screen stabilized object, as a world-stabilized object pasted onto a real world object, as a virtual billboard in physical space, always orienting itself towards the user, etc. Even different output media can be used to realize the media snippet. Unformatted text, for example, can be presented as 2D text on a hand-held display, as 3D text in AR, or can even be pronounced by a speech synthesizer and presented via loudspeakers or earphones.

By representing media materials separately from the objects responsible for their final presentation (which is done by *realized objects* referencing these media snippets), the MARS can flexibly alter the presentation of multimedia material based on available resources. For example, an AR tourist guide might show images of a restaurant's interior to a user either on a 2D hand-held/palm-top display, if that is available, or embed them in the augmented view of the scene via AR glasses.

Media materials have a potential perceptual embodiment in one or more media. The parameters that describe what objects the media materials annotate, and in what way, are encoded as semantic and realization properties, as described in Section 4.2.2.

UI Templates and Realized Objects Examples The set of UI templates determines the range of possible UI elements. UI templates resemble the concept of classes in object-oriented programming, with realized objects resembling object instantiations of those classes.

As a simple example consider the concept of a label that is to annotate an object, say a statue, in the physical world, viewed through AR glasses. In the simplest case, a text string is to be placed at a specific fixed 3D position in the augmented world. On closer examination, there are a lot of design decisions involved. To name just a few: Should the text be 2D or 3D (i.e., should the characters have an extension in depth)? Should it be placed at a fixed angle with respect to the statue, or should it always be oriented

towards the viewer? Should the apparent font size be fixed regardless of the distance to the object, or should the text behave like a physical sign posted on the statue (risking lack of readability when viewed from a distance)? Should the label be selectable by the user? What font type, color, and attributes should be used? Also, should the label really be fixed at a specific 3D position, or should it have the freedom to update its position and appearance slightly to ensure the correspondence with the statue. For example, if the viewer's position is such that another sculpture is partially occluding the statue, should not the AR system make sure that the label could not be misinterpreted (assuming the MARS knows about the occluding sculpture)?

Design decisions such as these are settled using the UI template concept. UI templates describe the general structure and behavior of UI elements. UI templates also provide a number of parameters (in the case of our label for example the text string to be conveyed and the object or location to be annotated). A realized object gets created by filling in these parameters.

Returning to the above example of labeling a real-world statue, there are two extreme positions regarding implementation of different label versions: the first one is to have a distinct UI template for each desired permutation of the above (and more) realization possibilities. The other extreme is to have just one UI template representing the concept of a generic label, and handling all design alternatives as additional parameters for that template. The best implementation strategy depends on many different factors, such as the number of representation alternatives desired, the frequency of occurrence for each of these alternatives, differences in semantics among the alternatives, and plans for extensibility and modular nesting of UI templates. In our example, it might make sense to create different UI templates for selectable and static labels, but to express all the other representation alternatives as parameters of the templates.

UI templates can contain links and references to any media materials, physical or realized objects, or collections or vectors of objects. These can be provided as parameters when creating a realized object from such a template. The result is a composite object that contains references to other objects, possibly hierarchically, with an atomic object forming the leaf of the hierarchic structure. For example, a UI template for a world-stabilized pop-up menu might take as parameters the physical object it annotates, as well as realized objects for each of the menu entries, which in turn were created from a menu entry UI template that could require the provision of a second-level menu, and so on.

Once the UI templates and their parameters are settled, *realized objects* can be created dynamically from them in order to populate the MARS UI. These realized objects form the virtual layer of the MARS UI. Many of the attributes that are specified as parameters to the UI template during the creation of each object are also reflected in a hierarchy of formal MARS object attributes. Such attributes, provided for every object

the MARS deals with, allow the MARS to reason about the objects during run-time. The following section presents a taxonomy of MARS object attributes.

4.2.2 Object Attributes

In the previous section we have introduced the notion of different categories of MARS objects. In order to allow the MARS to store information about each of these objects in a systematic manner, we need to define a general set of MARS object properties. This enables the MARS to retain sufficient knowledge about its UI components to make dynamic design decisions during run-time. From the perspective of a reasoning system governing the composition of the MARS UI, a MARS object is fully described by the sum of its attributes.

Figure 4.3 gives an overview of the hierarchy of attributes. We group MARS object attributes into four different categories: type attributes, state attributes, realization attributes, and semantic attributes. Type attributes refer to the intrinsic and unchangeable characteristics of the object. State attributes describe the current state of the object — the values for these attributes can change over time. Realization attributes describe the range of possibilities and preferences for an object's instantiation and appearance. Finally, semantic attributes list the class properties and relationships that inform the MARS about the object's meaning and importance, and attempt a classification according to its purpose within the MARS UI. In the following subsections, we will take a look at each of these categories in turn.

4.2.2.1 Type Attributes

Type attributes describe the type and category of a MARS object and specify if it is an atomic or composite object. Type attributes are further subdivided into the *category* and *composition* attributes. The category attribute refers to one of the categories in the hierarchy of MARS objects (cf. Figure 4.2), and thereby decides if the object at hand is a *physical object*, a virtual *realized object*, a *media material*, a *UI template*, or a container object, in which case it combines other objects for the purpose of referring to them as a group, either as a sequential *vector*, or unordered *collection*.

The *composition* attribute informs us about the intrinsic constitution of a MARS object: if it is atomic, a composite structure, or an unordered set of other objects. Note that this concept is related to the abovementioned concept of a container object in the following way: All *collection* objects (cf. Figure 4.2) are classified as *unordered sets* in their composition attribute. *Vectors* are classified as *composite structures*. Independent of the container object concept, AR objects can be either atomic, or they can be compound structures that consist of a group of several other objects. For example, a physical

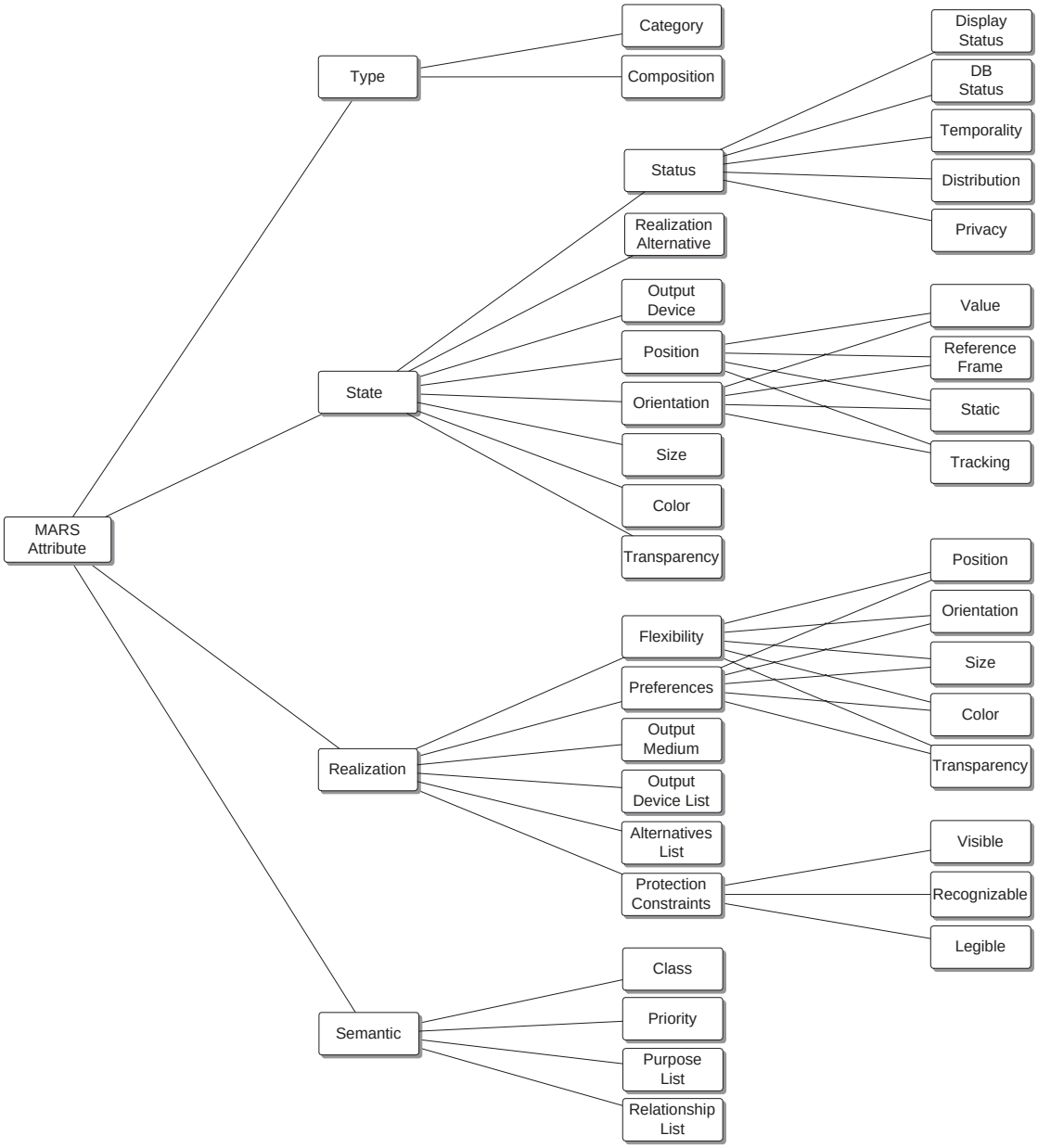


Figure 4.3: MARS object attributes.

building can be represented as a series of floors, which in turn subdivides into a series of rooms and corridors, each of which is a collection of walls, doors, windows, etc. If any subparts of objects are individually represented, and the compound object refers to these subparts, the compound object is classified as a *composite structure* in its composition attribute. If, however, the building is represented as a single entity with no further subdivision, it is classified as an *atom*. The same classification principle applies equally to virtual objects.

4.2.2.2 State Attributes

State attributes describe the alterable characteristics of objects as they change over time. These include the object's status, realization alternative, output device, position, orientation, size, color, and transparency. Note that size, color, and transparency apply only to objects with a visual representation. They certainly do not form a complete set of appearance attributes, but these three can easily be continuously varied. Other visual attributes, such as texture, drawing style, and shading style, can be altered via the *realization alternative* attributes. A list of visual alternatives, all renderable on the same set of output devices as the object in its default appearance, can be stored in the *realization alternatives list* attribute, which is part of the realization attributes (see Section 4.2.2.3). The state attribute *realization alternative* keeps track of which one of the set of realization alternatives is currently active.

Changing an Object's Appearance There are three ways of changing the appearance of any one UI component: First, to update its appearance attributes (size, color, or transparency) directly. Second, to provide one or several representation alternatives, and switch between them. And third, to discard the object and create a new one from a different UI template. Note that the first two alternatives, by necessity, stay within the same output medium. The third alternative allows alternative representations across different output media (e.g., converting a written text message into a spoken audio clip).

Changing the output device while staying within the same output medium (e.g., shifting a 2D graphics element from the head-worn display to a hand-held display), can be easily facilitated by updating the *output device* attribute.

Position and Orientation A set of attributes reflects the object's position and orientation as displayed on the current output device (whenever that situation applies). Orientation and position values are expressed with respect to the current reference frames. Possible reference frames include screen, body, and world, and are discussed in more detail in Section 4.2.3.2.

Two flags indicate the object's dynamic behavior regarding position and orientation: the *static* flag is set if the object stays constantly static (in position or orientation) with respect to the current reference frame, and the *tracking* flag indicates if the object's position and orientation updates are observed and dynamically reported. The tracking flag is most relevant for physical objects that are monitored by position and/or orientation trackers. Note that it is possible for an object to be static with regard to the current orientation reference frame, and not static with regard to position. Consider, for example, a screen stabilized menu on a head-tracked display that is anchored to a 3D point in the world, so as to maintain its position on the screen to exactly overlay the world anchor position, and against the edge of the screen nearest to that point whenever the anchor point is not in view. This menu's reference frame for both position and orientation is the 2D screen, and it is static with regard to orientation (it is always aligned with the screen), but dynamic with regard to position, since it moves with regard to the screen coordinate system (in response to head motion).

Status Flags Status attributes reveal information about the object's current state: the *display status* attribute signifies whether the object is currently displayed. The *DB status* attribute reveals whether an entry for the object is currently stored in the data repository. The *temporality* flag indicates whether the object is currently involved in any temporal behavior, such as an animation, or the playback of an audio or video stream. Note that animations based solely on position and orientation transformations are not included in this case, since those are simply expressed by continuously updating the position and orientation attributes described above. Two more flags keep track of rudimentary distribution and privacy behavior, as discussed in the following paragraph.

Distribution and Privacy Behavior MARS interfaces are often distributed over several computers. This situation arises, for example, whenever the MARS utilizes a shared resource, such as a wall-sized display, that is controlled by its own access server. Also, collaborative interfaces, such as the ones described in Section 5.2 replicate certain interface elements among different clients. Selective partial replication of MARS objects is of particular use when dealing with the issue of privacy management in collaborative UIs (Butz et al., 1998; Szalavari et al., 1998; Butz et al., 1999).

For the purpose of this taxonomy, the question arises if distribution and privacy behavior should be modeled on the level of MARS objects and their properties at all. A more global perspective can more efficiently handle all details of complex distribution behavior. To give an example, interface objects should not have to carry with them a list of clients that they are distributed to. Such information is of little use to the system watching over the composition of the local UI, and can more effectively be handled

centrally with distribution and access lists. We decided that, while distribution and privacy should not be *completely* modeled as MARS object attributes, there is still value in the UI having a notion about which objects are part of a distribution scheme at all, and which objects are considered private in any way, be it only to the local client or a group of clients. This information is captured in the two flags *distribution*, and *privacy*.

4.2.2.3 Realization Attributes

Realization attributes describe the range of display or realization choices for objects, and their display preferences.

As discussed above in the section on *state attributes*, there are three ways of changing an object's appearance. The first two ways, namely directly changing appearance attributes and switching between realization alternatives, are controlled by three sets of realization attributes: *Flexibility attributes*, *preferences*, and the realization *alternatives list*. Flexibility attributes and preferences control the allowed ranges and preferred values for the following five attributes, which we already discussed as part of the *state attributes* set above: position, orientation, size, color, and transparency.

The *alternatives list* is an ordered list of discrete representation alternatives for the given object, all designed for the same output medium. As an example for representation alternatives, consider the case of a mobile AR UI that allows the collection of virtual objects from an augmented environment (cf. Section 5.1.2). The MARS user picks up virtual 3D icons that are scattered around the physical world, representing information about those locations. These 3D objects, once picked up, are kept in a screen-stabilized shelf for easy overview and access. Since the observer's angle onto the screen-stabilized 3D icons does not vary, the icons can be represented by a texture map rather than complex 3D geometry while they are kept in a fixed position on the screen. Such a texture mapped polygon can be conveniently expressed as a realization alternative. Note that the concept of realization alternatives is used dynamically at run-time. Decisions when to create new realization alternatives, and when to switch back and forth between them, are made by the run-time reasoning engine controlling the execution of the MARS application (cf. Chapter 6).

The third, and most radical case of changing an object's appearance, involves discarding the MARS object altogether, and recreating it using a different UI template. This is how an object can switch from one output medium to another (e.g. speech synthesis for an originally written text message, or a 2D image from a web page being rendered as a 3D billboard in the augmented world seen through AR glasses). Unlike the *output device*, the output medium is fixed over the lifetime of a MARS object, and hence this attribute is not part of the set of alterable *state attributes*. The realization attribute *output medium* identifies the medium that the object is expressed in (cf. Section 4.2.3.1).

The *output device list* attribute, on the other hand, keeps a list of possible output devices, ordered by desirability of usage. Most interface elements are specifically designed for one particular output device (e.g. world annotation labels for AR glasses), but some UI objects can be displayed on different devices. A web page that pops up in response to some user selection, for example, may show up on a web browser that occupies a half of the screen in the AR glasses, or on a browser on an accompanying hand-held display. Note that the devices listed in this attribute may not necessarily be available at all times. The *system object* (cf. Section 4.2.1.4) keeps track of the resources available to the MARS at each point in time.

Flexibility Attributes Flexibility attributes describe what the display options are for the object, and what ranges of values the different appearance attributes accept. For position and orientation, the flexibility is most commonly expressed as symbolic levels of flexibility (ranging from "absolutely static" to "completely unconstrained"), but can also include specific quantitative requirements (e.g., "angle with viewing plane $\leq 25^{\circ}$"). All flexibility recommendations are made with respect to one of the possible reference frames (screen, body, or world). For size and transparency, concrete ranges of tolerable values are listed. For color, the situation is a little more complicated. Some objects, such as textual labels, have only one main color. For these, color variations can happen in a fashion similar to transparency variations, with the obvious difference that colors are represented by vectors instead of scalar values and therefore need special data structures for expressing color ranges. In contrast to such uniformly colored objects, however, many MARS objects (such as 3D models of physical things) by default consist of multicolored parts. The effect of assigning a new color to such a graphical object is that all parts of that object are drawn in the given color, thereby highlighting the object. In this case, the whole set of original colors assigned to the various subparts of the object has to be stored together with the correct part-color relationships to enable reverting from the highlighted to the original state. In the case of textured objects, color highlighting is more easily accomplished, since textures can be modulated by colors.

Preferences Preferences refer to the same set of attributes as the flexibility properties above. Here, for each attribute, the preferred value is listed, which is to be used in a situation where no outer constraints and influences demand otherwise. Position and orientation values are expressed with respect to a specific reference frame. Size, color, and transparency values reflect the default appearance of the object. Note the above discussion of color variations in the case of multi-colored objects.

Protection Constraints Protection constraints are similar to the visibility constraints of Feiner and Seligmann (1992) and Seligmann (1993) . With these flags, the system tries to ensure that certain perceptual activities can be successfully applied to the objects in question. The *visible* flag requires the object to be visible at all times. Occlusion of such an object has to either be avoided, or counteracted (e.g., by ghosting, or cutaway views). The *recognizable* condition involves maintaining sufficient detail in the representation of the object that it can be distinguished from all other objects of similar appearance, and be correctly identified. This concept is fuzzier than plain visibility, and it is also much harder to evaluate. The third protection constraint concerns the legibility of text information. Labels and text in popup windows and dialog boxes are to be displayed in such a fashion as to stay legible for the current user. Legibility of text information is dependent on various factors, including the font type, size, color, reading angle, partial occlusion, and the user's eyesight, which can be considered in the system's user model.

4.2.2.4 Semantic Attributes

Semantic attributes provide information about what kind of entity the MARS object represents, about the relative importance of the object as compared to other UI elements, the object's *purpose* in the UI, its relationships to other objects, and, as part of the relationship list, possible trigger *events* that let it change its state.

The *class* attribute is aimed towards classifying the semantics of the object beyond what is revealed by the *category* attribute from the set of type attributes. For example, it may be important for the MARS to know that a specific physical object is a building and not a statue, a truck, or a tree. The same goes for virtual objects. The class attribute aims to capture their type in more detail than just "interface element" or "3D model". Concepts such as "label" and "alert message" are already more specific than a generic "UI element". Virtual objects can be tagged by the concepts they represent (e.g., the model of a desk being classified as "furniture"). Such classifications may be used by a rule-based system (cf. Chapter 6), some of whose rules operate on sets of objects, grouped by their semantic class attribute. Unidentified objects, or objects lacking special behavior rules may be simply classified as "generic". A more complete model of the world could be achieved by forming at least some partial ontology of the world (Russell and Norvig, 2003).

The *priority* attribute aims to provide hints as to the relative importance of some selected MARS objects. In absence of any criteria for priority, all objects are considered to have the same level of consequence. However, certain objects can stand out. For example, an object that serves as the target destination of a planned route might be more important than landmarks on the way. An urgent alert message should take priority over other screen elements. In a military application, the highlighted position of a sniper might

be more important than the names of buildings in the environment. Importance values are application- and task-dependent, and may be updated by special purpose rules.

The *purpose* list enumerates what the object is used for in the UI. Purposes include, for example, serving as information, information provider, anchor, link, landmark, or functionality provider. Section 4.2.3.5 lists the currently implemented set of purposes and gives concrete examples.

The *relationship* list captures the connections between different objects, such as one object being part of another, menu items or buttons triggering certain behaviors in other objects, layout constraints, or semantic links, such as one object representing another (e.g., a virtual model of a building representing the real thing). Section 4.2.3.3 lists a set of relationships and gives concrete examples.

4.2.3 MARS Concepts

While discussing the properties of MARS objects, we have been introducing several concepts that are needed to adequately describe the state, realization attributes, and semantics of these objects. In this section, we will look at these concepts in more detail, and will give application examples. In particular, we will discuss the following MARS concepts: Output media, reference frames for orientation and position measurements, relationships between MARS objects, goals, purposes, and events.

Figure 4.4 presents an overview of MARS concepts, for each of them listing some examples or referring the reader to a more detailed figure. We already presented MARS objects (in Section 4.2.1) and MARS object attributes (in Section 4.2.2). MARS interaction tasks and techniques, and input and output devices will be discussed in Section 4.2.4.

4.2.3.1 Output Media

The output media that we consider for our MARS UIs are the *real world*, *3D graphics*, *2D graphics*, *sound*, and *touch*. Theoretically, augmentations of smell are also possible, as has been demonstrated by Morton Heilig's Sensorama as early as 1960 (Heilig, 1992), and augmentations of taste are at least conceivable, but for now we focus on media technologies that are realistically implementable in our prototypes.

Research in multimedia systems and multimodal interfaces has produced quite a few different taxonomies of multimodal output representations (Bernsen, 1994; Furht, 1994; Blattner and Glinert, 1996; Heller and Martin, 1995; Heller et al., 2001). The common denominator is a set of core media, in particular text, graphics, sound, and video/animation.

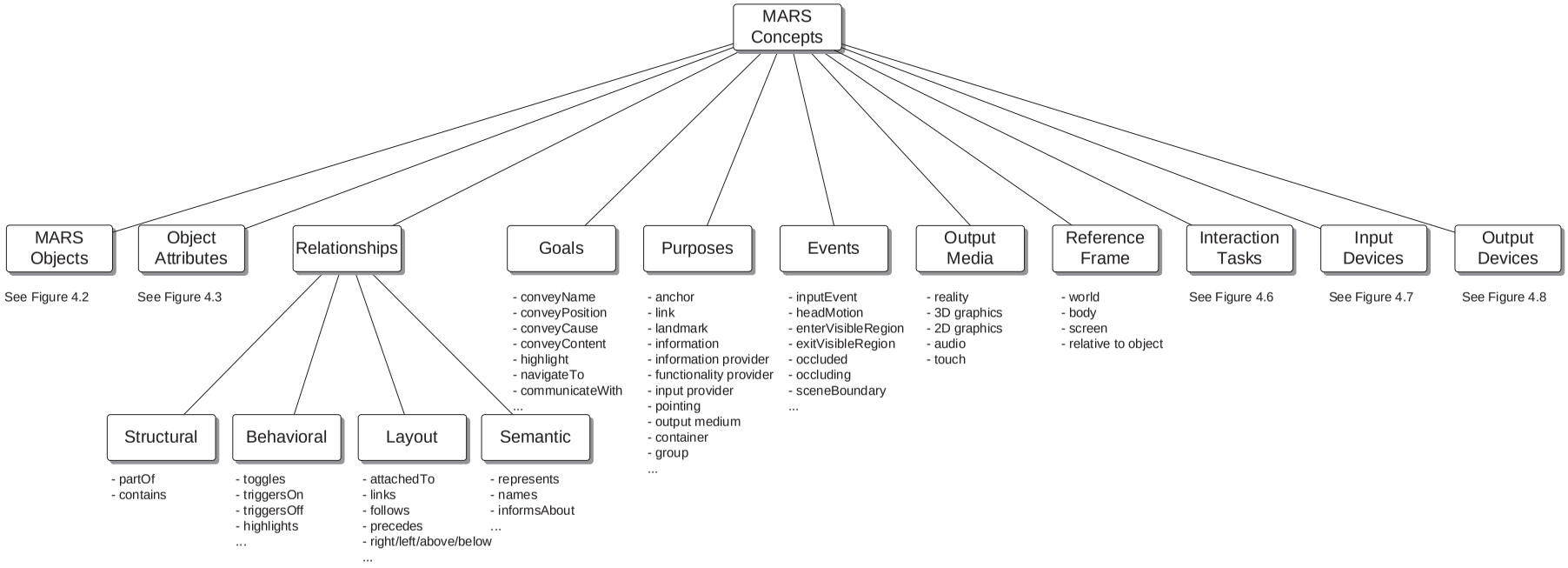


Figure 4.4: MARS concepts.

In our taxonomy we do not include text as an output medium in its own right. Instead, we represent it as a media material (cf. Section 4.2.1.4). While text is an important raw material for presentations, in the context of MARSs it always gets realized via an output medium such as 2D graphics, 3D graphics, or sound (speech). Therefore, as an example, we categorize text that is conveyed via a 2D display as 2D graphics in our taxonomy, no matter if it appears as part of a web page, in a text editor, or in a command-line interface window,

We include the real world, because the computer can control specifically equipped physical objects via robotic interfaces. We include touch, because mobile computing interfaces increasingly employ this modality (Ross and Blasch, 2000; Poupyrev et al., 2002). For example, a navigational interface might communicate direction by tactile feedback that emulates "shoulder-tapping" (Tan and Pentland, 2001).

We differentiate between 2D graphics and 3D graphics because in AR there is a profound difference between a virtual 3D object that is populating a physical environment, and a 2D image displayed, for example, on a palmtop computer. Most 2D graphics elements can be represented easily in a 3D environment — texture mapping on arbitrarily oriented polygons is one possibility — but such a transition is really a media conversion, unless we define such a "screen polygon" to be an abstract 2D graphics device in the 3D environment and provide all the API mechanisms that allow this abstract device to act like any other 2D graphics device. In that case, no media conversion needs to take place, but still, 2D and 3D graphics are represented as different media.

We do not separately consider video and animation, because the temporality of some media in our opinion is an orthogonal issue, and is reflected as such in our taxonomy of MARS object attributes (Section 4.2.2).

4.2.3.2 Reference Frames

An interface for visual mobile AR can combine screen-stabilized, body-stabilized, and world-stabilized elements (Feiner et al., 1993a; Billinghurst et al., 1998b). Figure 4.5 shows a UI from our mobile AR work (Höllerer et al., 1999a), photographed through optical see-through head-worn displays. The virtual flags and labels are world-stabilized objects, denoting points of interest in the environment. They are displayed in the correct perspective for the user's viewpoint, so the user can walk up to and around these objects just like physical objects. The labels turn and maintain their size irrespective of distance to always stay readable. The blue and green menu bars on the top are screen-stabilized, meaning they stay on the screen no matter where the user is looking, as does the cone-shaped pointer at the bottom of the screen, which is always pointing towards the currently selected world object.

Body-stabilized information is stored relative to the user's body, making it acces-

Figure 4.5: Different frames of reference in a MARS UI.

sible at a turn of the head, independent of the user's location. Note, that in order to store virtual objects relative to the body with respect to yaw (e.g., consistently to the user's left), the body's orientation needs to be tracked in addition to head orientation. One can extend the notion of body stabilized objects to using general head-gestures for virtual object control. Section 5.3.2.1 provides an example.

Finally, any MARS object can define its own reference frame. This concept is, for example, used in hierarchical composite structures, such as the representation of a building, whose window and door locations may be expressed in building coordinates instead of world coordinates.

4.2.3.3 Relationships

Relationships between MARS objects are listed as part of the MARS object attributes. We currently distinguish four different kinds of relationships: Structural, behavioral, layout, and semantic.

> *Structural relationships* link composite objects with their parts and vice versa. *IsPartOf* and *contains* are the prime examples.
>
> *Behavioral relationships* reflect the activation of behavioral changes in one object as caused by another. As an example, the selection of a menu item can cause the display (*triggersOn*) or change in state (e.g., *highlights*) of a virtual object.
>
> *Layout relationships* indicate the placement/arrangement of objects relative to each other. Two of the most important examples are the *attached* and *link* relationships: A label is attached to the object it is annotating; a leader line

links two objects. Other layout relationships include *precedes*, *follows*, and other geometrical or temporal constraints.

Semantic relationships, finally, signify semantic links between objects. For example, a virtual model of a building *represents* the real-life original. A label *names* an object. A pop-up window *informsAbout* the object it annotates.

4.2.3.4 Goals

We distinguish between the system's presentation goals and the user's task goals. Presentation goals include all kinds of communication objectives, such as *conveyName*, *conveyPosition*, *conveyContent*, or *conveyCause* (which makes explicit how a specific state change came about). All of these can be broken down into more specific goals (e.g., to *highlight* an object).

The user's task goals come into play when the MARS is to be used in a specific task rather than as a physical world browser. Examples include *navigateTo* a specific place, and *communicateWith* a specific person.

4.2.3.5 Purposes

Purposes indicate what an object is used for in a MARS UI. They are similar to relationships, but are unary, instead of binary, attributes. Purposes include pointing (graphical feedback for input devices), providing information, providing functionality, and indicating relationships. The following is a list of concrete examples, currently implemented in Ruby:

Anchor: anything that other objects are linked to, such as for example a point of interest.

Link: an element that reveals the connection between two objects, such as a leader line.

Information: any content material that is supposed to inform or otherwise entertain the user, such as for example a label or an audio snippet. Many objects whose purpose is not more specifically classified, fall in this general category.

Information provider: an interface element that stores and arranges information, such as for example a billboard or a pop-up note.

Functionality provider: an element that is in a behavioral relationship with other objects (i.e., ones that trigger some action in the interface). Examples include a menu item or button.

4.2.3.6 Events

MARS events include various types of selection, other input device trigger actions, move or turn actions, and certain changes in relationships between objects that are monitored by some entity. To make this more concrete, here is a list of examples implemented in Ruby:

Input event: This is a user interaction coming from an input device. For example: a button click or recognized hand gesture. Also, possibly, a more complicated action, such as visual selection of an object as described in Section 5.1.1.

Head motion: Apart from being involved in more complicated selection and head gesture schemes, the pure presence or absence of head motion (above and below a certain threshold value) can also be a trigger event.

Enter visible region: This is an object-specific event. The event is triggered whenever the respective object enters the view frustum.

Exit visible region: Same as above but for leaving the view frustum.

Occluded: This is another object specific event. It gets triggered when the respective object gets occluded (from the user's view) by another object that the MARS is aware of.

Occluding: This is yet another object specific event. It gets triggered when the respective object occludes (from the user's view) another object that is part of a list of objects to be monitored for that purpose.

Scene boundary: This is a global event that gets triggered when the mobile user crosses the boundary of the scene that the MARS currently has environment information on. This triggers contacting a server and downloading information about the scene across the boundary.

4.2.4 MARS Interaction

MARS interaction provides the mobile user with the means to handle the information presented in the AR environment. Among other things, AR objects can be selected, moved, annotated, altered, or created in the first place. We distinguish between *interaction tasks* (Figure 4.6), which state the purpose that an interaction is supposed to fulfill, and *interaction techniques*, which are concrete implementations of how to perform an interaction task using specific input and output devices (Figures 4.7 and 4.8).

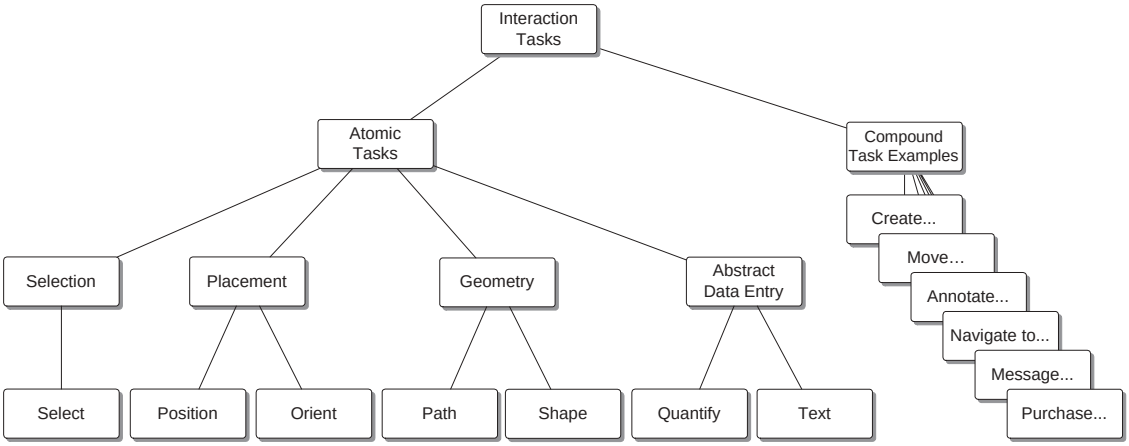


Figure 4.6: MARS interaction tasks.

4.2.4.1 Interaction Tasks

A person makes use of different input and output devices, and the interaction techniques designed around them, to reach his or her goals (cf. Section 4.2.3.4). In fact, many of a user's goals can be stated as high-level compound interaction tasks (e.g., sending a message to a certain person or purchasing a certain item). These complex interaction tasks can be repeatedly broken down into increasingly simpler tasks, taking into account the affordances of the interaction techniques to be employed. In the end, we reach the level of atomic interaction tasks, which can be executed by specific interaction techniques of the user's choice. Foley et al. (1984) suggested the elementary graphical interaction tasks *select*, *position*, *orient*, *path*, *quantify*, and *text*. We add to these the concept of *shaping*, meaning the forming of geometrical shapes using free-form gestures. Shapes could potentially be produced using compound interaction tasks that employ simple manipulation of control points, which locally influence a geometric shape. However, in order to provide for the possibility of using direct manipulation gestures for the same purpose, we decided to represent it as a atomic task in its own right. It is related to path drawing in that both are geometric creation tasks that involve time. Shaping, however, produces general 2D or 3D geometries, whereas paths are one-dimensional routes through 2D or 3D space.

Foley et al. (1984) discussed *shape* as one of four "controlling tasks", together with *sketch*, *stretch*, and *manipulate*. In our framework, *sketch* can be realized with variations of the *path* and *shape* primitives, parameterized with brush and line style information. *Stretch* is a compound task that involves selection and positioning or path drawing. *Manipulate*, likewise, can be expressed using atomic interaction tasks; in fact we see it as a generalizing subsumption of other interactions.

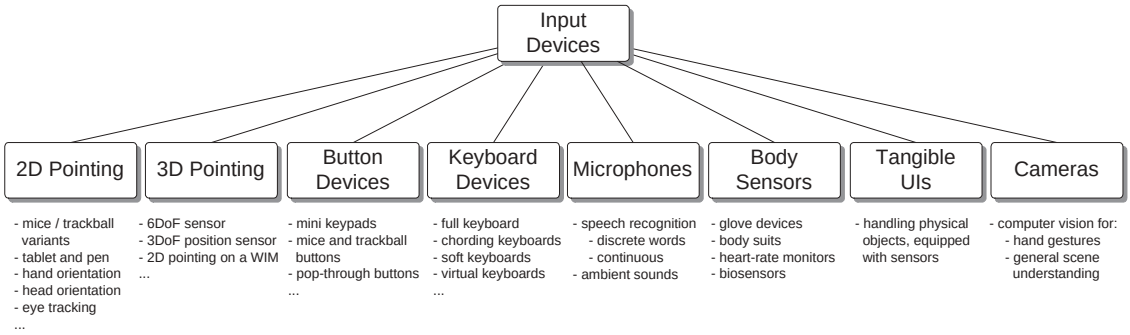


Figure 4.7: MARS input devices.

As can be seen in Figure 4.6, we separate the atomic interaction tasks into four groups, divided by their function. Selection forms a group by itself. Placement, or pose determination includes positioning and orienting. Geometry manipulations are represented by pathing and shaping, both time-dependent operations. Finally, quantify and text entry are examples of abstract data input.

Often, in VR or AR interfaces, we employ special purpose input devices or use tracking mechanisms (such as vision-based gesture recognition) to bring about a certain interaction with the computing environment. Finding the most suitable mapping between the available devices and the intended interaction, is the core challenge in devising appropriate interaction techniques. While it is possible to control certain interaction variables directly using interface hardware, a virtual interface layer is always almost needed to conceptualize the interactions into an overall framework that is intuitive yet powerful.

In the following subsections we briefly discuss the interaction possibilities of different input and output devices.

4.2.4.2 Input Devices

Figure 4.7 gives an overview of several input devices that can be used with MARSs. For more background information, discussion, and research results, please refer to Section 2.4.5.

Pointing devices are very common for selection and manipulation tasks. In WIMP interfaces, a mouse is the most common pointing device, operating in 2D screen space. There are many variants of 2D mice and trackballs that are applicable in a mobile setting, with a small form factor and possibly a wireless connection to the host computer. In absence of a base surface to operate a mouse on, in a general mobile setting a trackpad, trackball, or a hand orientation tracker can be used to steer a 2D cursor on a screen. Tablet and pen are another indirect pointing combination. Also, head-orientation can be used to select certain elements in a world-stabilized interface. Eye-tracking is a technology that

might one day add true gaze control to the arsenal of pointing mechanisms, but for the time being is confounded by usability problems (Jacob, 1991), and is particularly hard to implement in a mobile setting.

Often, the user's task is to select and manipulate an object in 3D. This is considerably more complicated than pointing in a 2D plane as mentioned above. 6DOF sensors can be used to pick objects and turn them around intuitively, but, as described in detail in Section 2.4.3, 6DOF tracking for a mobile computing system is hard. Piekarski and Thomas (2001) propose the use of markers attached to gloves to track a user's hands within the field of view of a head-worn camera. Tracking a ring-mounted ultrasonic sensor in 3DOF position relative to the head, Foxlin and Harrington (2000) explore the UI possibilities of a wearable position-only 3D pointer. Often, 3D manipulation can be achieved using 2D interaction (Hinckley et al., 1994a; Stoakley et al., 1995; Poupyrev et al., 1996; Bowman and Hodges, 1997; Pierce et al., 1997).

Most mobile mouse replacements provide a number of buttons to the user. Button devices are used to register discrete input events with the computer. The number of buttons available on an input device can range from one to the order of tens, as is the case with keyboard variants. Buttons may just be used to register binary events (pressed vs. not pressed), but the potential of providing an extra intermediate state, as for example exploited in the pre-focussing of many auto-focus cameras has recently been explored (Zeleznik et al., 2002).

Keyboards represent a special case of button devices and are therefore listed separately. Their main purpose is to allow a user to input text in as fast a manner as possible. Enabling fast text entry comfortably without exceeding certain form factor restrictions that mobile use imposes is an unresolved question. Among the proposed solutions are software keyboards on computer displays, chording keyboards (Handykey, 2001), flexible and foldable keyboards (Grandtec, 2001), and even virtual keyboards that are projected onto arbitrary surfaces (May, 2003).

An alternative input mechanism for text is speech recognition. The input device of choice for this is a microphone or a microphone array. Speech recognition is feasible in discrete word and continuous speech mode, and recognition rates are rising to acceptable percentages for use in controlled environments. Microphones can also be used to record other sounds that may be of use to the UI.

Different kind of body sensors, such as gloves, tracking suits, heart-rate monitors and other biosensors form the next category in our list. A distinction can be made between sensors that record deliberate gestures the user issues and sensors that collect background information that may prove useful in non-command interfaces (cf. Section 4.1).

Tangible UIs (cf. Section 2.2) make use of everyday objects whose handling can

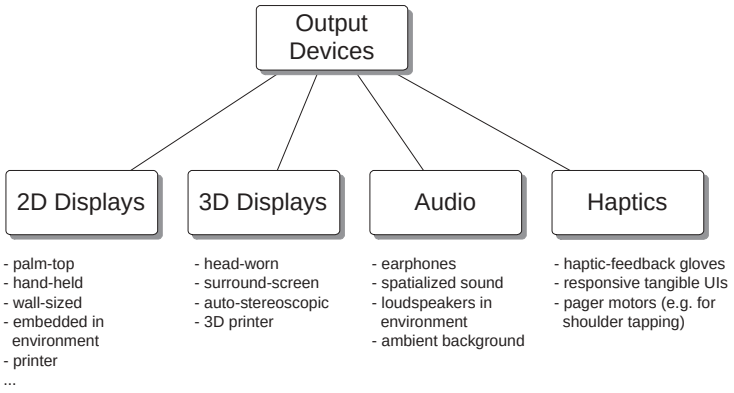


Figure 4.8: MARS output devices.

be observed by the computer. The way a user handles such interface objects can control the object's own perceived behavior, but may also trigger other related computer actions. The reason for making specific objects computationally reactive in such a fashion is that they may provide the most intuitive interface for some special purpose tasks.

A long-term research goal is to make computers more aware of their environment. One major area of research to this end is concerned with computer vision techniques. Cameras are general purpose input devices that can be used for gesture recognition, face detection, and as the most ambitious task, general scene recognition.

4.2.4.3 Output Devices

We considered four categories of output devices, as depicted in Figure 4.8. Section 4.2.3.1 introduced the output media we consider in this taxonomy. Computer-generated information can be perceived by a user through any of his or her five senses, but only three of these senses have commonly available output devices associated with them: viewing, hearing, and feeling. We distinguish 2D and 3D output devices, just as we did in terms of output media.

There is a wide variety of 2D displays. Their form factors range from wall-sized to palm-top, with very different technologies behind them (cf. Section 2.4.2). Off-line displays such as printers are deliberately included in our list.

There are several options for 3D displays, including head-worn, autostereoscopic, and displays supported by polarized or shutter glasses. Surround-screen displays of the latter kind are a good example of immersive 3D displays, but they are the opposite of mobile. 3D printers are machines that can create small plastic objects from CAD models.

Audio output devices also come in different formats and technologies, ranging from earphones, including such that try to minimize blocking out the outside world, to speaker arrays that can mimic spatialized sound, loudspeakers in the environment, and

ambient sound installations.

Haptic output devices, finally, aim to deliver a sense of touch to the user. Force-feedback gloves, for example, allow a user to feel the reality of simulated objects. All kinds of motors can be embedded in physical objects to make them reactive. Vibrators (such as used in pagers and cell phones), worn on the user's body (e.g. in their shoulder pads) can inform a user of certain events.

This concludes our exploration of MARS interaction concepts. We have in turn looked at MARS objects (Section 4.2.1), their attributes (Section 4.2.2), and general MARS concepts (Section 4.2.3). Altogether, this presents a taxonomy of the building blocks and concepts involved in designing mobile AR UIs.

Our goal in presenting this taxonomy of MARS concepts was to establish a theoretic framework that can explain the components of MARS UI techniques. The next chapter will present such UI techniques, as implemented in our MARS framework. Finally, the taxonomy can be a tool in enabling the UI techniques to be adapted to various new situations, and facilitating future design decisions. An architecture to explore such adaptive MARS UIs is presented in Chapter 6.

Chapter 5

Implemented MARS UIs

Over the course of our explorations of MARS UIs, we have designed various UI components and techniques that allow MARS users to interact with the augmented environment. These MARS UI components were developed as part of specific applications that we implemented to showcase the potential of the mobile AR UI paradigm. A good example is the series of *situated documentary* applications (Höllerer et al., 1999a), we will describe in Section 5.1. Many of the UIs presented in this chapter have been developed through teamwork. Table 5.1 lists the main applications or *UI environments* that served as our testbeds for MARS UI development and testing, together with their main developers, respective project timeframes, and underlying software infrastructures and hardware platforms (cf. Chapter 3).

In this chapter we take a detailed look at these UI environments and the implemented user interfaces, discussing research contributions and lessons learned. Section 5.1 describes our work on spatially distributed hypermedia systems. It describes in detail the UIs we designed as part of our *situated documentaries* testbed. Different situated documentaries present historic events, infrastructure, and related information in the spatial context of Columbia's campus, integrated by a geo-referenced hypermedia UI (Section 5.1.1). Our documentary on the early years of the Manhattan Project (Section 5.1.2) experiments with more carefully orchestrated interactive storytelling. Section 5.2 describes UI techniques that we developed to facilitate collaboration among different MARS users and across different UI environments, indoors and outdoors. In particular, Section 5.2.1 presents interfaces for indoor supervision of outdoor exploration and navigation tasks, Section 5.2.2 describes joint work with Andreas Butz, in which we explored AR UIs for augmented meetings that make use of a heterogeneous set of computing and display resources, and Section 5.2.3 discusses UI components and techniques for information browsing and peer-to-peer communication among MARS users.

We then present several UIs that are concerned with supporting and enhancing

UI Environment	Main Developers	Year	Software Infrastructure	Hardware Platform(s)
Touring Machine	B. MacIntyre, (T. Höllerer)	1997	Coterie	Touring Machine
Situated Documentaries I	T. Höllerer	1998	Coterie/SitDoc	Touring Machine
Emmie	A. Butz, T. Höllerer	1998	Coterie	AR Lab 1998
Situated Documentaries II	T. Höllerer	1999	Coterie/SitDoc	MARS 1999
Indoor/Outdoor Collaboration	T. Höllerer, D. Hallaway, T. Terauchi, R. Rashid	1999	Java/Java3D, Coterie/SitDoc	MARS 1999, MARS 2000
Indoor Navigation	T. Höllerer, D. Hallaway	2000	JABAR	Indoor MARS 2000
Situated Documentaries III	T. Höllerer	2001	JABAR	MARS 2000, MARS 2001
AR View Management	B. Bell, T. Höllerer	2000-2001	CGUI AR	AR Lab 2001, MARS 2001
AR WiM	B. Bell, T. Höllerer	2001-2002	CGUI AR	AR Lab 2001/2002, MARS 2001/2002
Restaurant Guide	T. Höllerer, B. Bell, J. Tang, S. Güven, T. Zhou	2002	CGUI AR	MARS 2002
Rule-based AR	T. Höllerer	2001-2003	CGUI AR/JESS	MARS 2002

Table 5.1: Implemented MARS UI environments, software infrastructures and hardware platforms.

a user's spatial awareness and navigational abilities (Section 5.3). These were the outcome of joint work with Blaine Bell. The UIs make use of world-stabilized navigational cues and annotations, as well as body- and screen-stabilized overview visualizations of the user's environment (Sections 5.3.1 and 5.3.2). We summarize our MARS UI design work in Section 5.4. We review the most serious problems we encountered, and discuss solutions. In particular, we emphasize the need for adaptive UIs in Section 5.4.3, and present three important steps to manage the visual composition of a MARS UI, in effect forming a UI-management pipeline (Section 5.4.3.1). Section 5.4.3.2 discusses one of these steps, *view management*, in more detail, explaining how it can control the layout of annotations on the user's view plane, avoiding unwanted overlap and ambiguous or wrong associations of annotations and world objects. The taxonomy introduced in Chapter 4 establishes the theoretic framework to explain the components of all the interfaces described in this chapter, and enables the techniques to be adapted to various new situations, thereby facilitating new design decisions. Section 5.4.2 gives examples of implementation alternatives, devised using the concepts in our UI taxonomies.

5.1 Spatially Distributed Hypermedia

Columbia University's campus served as the test environment for most our explorations of MARS UIs. Position tracking on campus was first done through differential GPS using a differential correction service (cf. Appendix A.1), and later through RTK differential GPS using our own campus base station (cf. Appendix A.2). Compared to other areas in Manhattan, Columbia's campus with its relatively wide open greens and walkways provides good satellite visibility, helping our GPS tracking performance. Over the years we built increasingly accurate models of the campus infrastructure, which, in combination with increasingly accurate position and orientation tracking, enabled us to annotate more detailed features on campus and to create a realistic world-stabilized UI layered on top of a user's view of the campus.

The Columbia Touring Machine (Feiner et al., 1997) was the first outdoor MARS that featured visual AR overlays. It allowed a user to access information about the buildings and departments on Columbia's campus, presented to them as overlaid labels on top of the real environment and on (partially dynamically created) web pages on a handheld tablet computer. As the user looked around the campus, his see-through head-worn display overlayed textual labels on campus buildings. Because the application labeled buildings, and not specific building features, the relative inaccuracy of the trackers at that time was not a significant problem for this application.

In our work on *situated documentaries* we extended the Touring Machine infrastructure in several ways and developed new UI mechanisms. Supported by substantial

improvements in the hardware and software infrastructures for our MARS prototype (cf. Chapter 3), we developed applications that presented location-aware multimedia presentations to outdoor users. A *situated documentary* embeds a narrated multimedia documentary within the same physical environment as the events and sites that the documentary describes.

The concept of situated documentaries originated in a collaboration with Prof. John Pavlik from Columbia's Center for New Media in the Graduate School of Journalism. The results of the Touring Machine project gave rise to the idea of a *Mobile Journalist's Workstation*, a hypothetical "one-person-broadcast-van" kind of device, which would provide all the necessary recording, communication, and collaboration tools for a journalist in the field to cover a story and send it back to a news center for immediate broadcast. While brainstorming the technology's potential for the news *producer*, we also considered the applicability of MARS's for the news *consumer*, which became the focus of our situated documentaries series of application.

Situated documentaries rely in part on the idea of creating hypertext links between physical and virtual objects or locations (Feiner et al., 1993a). They extend the concept of a simple building-centric campus tour from the original Touring Machine application to include multimedia information about historic events and infrastructure, which could be linked to arbitrary locations on campus, designated by world-stabilized 3D icons. One of the most fundamental tasks for a journalist covering a story is to establish the physical space the story takes place in. We accomplish this by situating the news consumer literally at the story's location, and layering a multimedia documentary over that space, thus creating a spatially registered hypermedia presentation.

Working with journalism students from a class on new media technologies held in Spring 1998, Spring 1999, and Spring 2001, we implemented three different situated documentary presentations, each one consisting of multiple story threads. All journalistic investigations on the three situated documentaries were done by the journalism students. The first situated documentary reports the events of the student strike or revolt (depending on one's point of view) of 1968, in the exact places on Columbia's campus where notable incidents had taken place. The second situated documentary adds two more stories: One about the extensive tunnel system underneath Columbia's campus, and one about the Bloomingdale Asylum for the Insane, which had covered the grounds of today's campus in Morningside Heights before Columbia College was moved there in 1897. The third situated documentary tells the story of Columbia's involvement in the Manhattan project that would lead to the creation of the first atomic bomb. It also illustrates the possible power of nuclear destruction.

In the following section, we will present the work from the first two situated documentaries in a combined fashion, since the second documentary extended the first

one, forming a combined interlinked web of stories. We will discuss the third situated documentary separately in Section 5.1.2, because its UI went beyond the concepts of the first two prototypes to explore more directed narratives using interactive iconic elements.

5.1.1 Situated Documentaries I & II

The first situated documentary application used hardware and software environments that were extensions of the Touring Machine infrastructure. The first iteration UI took over several components and concepts from the Touring Machine. In the following paragraph, we briefly summarize the components of the original Touring Machine UI that we also used in the situated documentaries applications in one form or another. After that we will describe how the various situated documentaries UIs went beyond these concepts.

Touring Machine UI The Touring Machine UI, conceptualized and designed by Steven Feiner and Blair MacIntyre, assisted by the author of this thesis, introduced the following MARS UI components that in continued use have proved effective means of dealing with augmented information for mobile users who are exploring an unfamiliar environment and want to access data about physical objects in their surroundings.

- A screen-stabilized menu hierarchy. The menu contained separate bars and coloring for globally accessible commands and functionality associated with the currently selected world object. It was controlled by a trackpad mounted on the back of the hand-held computer. No cursor was used, since it is hard to fine-tune cursor control in a mobile environment. Also, in order to be readily visible on the low-resolution head-worn display, the cursor would have had to be quite big and would have wasted screen estate and gotten in the way of labels and other UI elements. Instead, the menu items were highlighted (by raising their intensity) according to relative finger motion on the trackpad. The trackpad was mounted on the back of the tablet computer. It was operated with the middle or index finger of the user's non-dominant hand, which at the same time held the tablet computer. The dominant hand was used for stylus input and for additional support during trackpad operation.

  We originally considered having the stylus control the head-worn display's menu when it was within a designated physical area of the handheld computer's display. We decided against this, however, because it turned out to be difficult to remain in that area when the user was not looking at the handheld display.

- *Fly-down menus*. To call the user's attention to new material on the hand-held computer when menu items that trigger the display of new web pages are selected,

a copy of the menu item was animated down to and off the bottom of the head-worn display.

- World stabilized *context menus*. These menus arranged information for a selected building (e.g. the list of departments located therein) in a semi-circle around the building's label. Selection of entries could occur via the trackpad or from a list on the hand-held display.

- A *VisualSelect* method for selecting virtual objects represented by labels. We employed the following interaction technique: labels, which normally appear in shades of grey, change their intensity gradually as they approach the center of the user's field of view. If a label is closer than any other to the center of the head-worn display and within a fixed target area, its color changes to yellow to indicate preselection. If it stays in that state for a period of about half a second, the object associated with the label becomes selected and the label color changes to green. This selection strategy is essentially a simple approximation of gaze selection using head tracking information only.

- A screen-stabilized compass pointer. This pointer, positioned at the bottom of the head-worn display, continuously pointed towards the currently selected objects. In combination with selecting objects (buildings) from a list on the hand-held display, this implemented simple navigational guidance functionality.

These UI concepts formed the basis that the first situated documentary application built upon. We iteratively fine-tuned several parameters, such as color and font choices, and we created a whole set of new UI components to extend these basic ideas. In general, the situated documentaries interfaces exhibit the following novel characteristics:

- Rather than linking individual labels or web pages to locations, they support context-dependent, narrated multimedia presentations that combine audio, still images, video, still and animated 3D graphics, omnidirectional camera imagery, and Java applets.

- They make extensive use of overlaid 3D graphics for both the user interface (e.g., world stabilized flags as 3D widgets for user guidance) and the presentation content (e.g., *in situ* reconstructions of buildings that no longer exist, views of visually obstructed infrastructure, and animated computer simulations).

- They embed the informational elements in a new *physical hypermedia* user interface that guides users through a presentation via hyperlinks, while giving them the freedom to follow their own trails through the material.

Figure 5.1: Situated Documentaries: Virtual flags denoting points of interest, photographed from the top of a campus building.

The following scenario describes how these UI concepts are put to use by a person who stands in the middle of Columbia's campus, wearing our experimental backpack computer system and a see-through head-worn display, and holding a tablet computer (cf. Figure 2.4). As the user moves about, his or her position and head orientation are tracked, and through the head-worn display the campus environment is overlaid with virtual material, as shown in Figures 5.1, 5.2, and 5.5 – 5.7.

The system presents a hyperlinked web of three main stories about events and historic information on Columbia's campus: a documentary on the Columbia student revolt of 1968, a tour of Columbia's extensive underground tunnel system, and a description of the Bloomingdale asylum.

The user can interact with the surrounding environment in different ways. On the hand-held computer, which is networked to the backpack computer that drives the head-worn display, the user can view and interact with information, and input data with a stylus. All information on the hand-held display is presented using a standard web browser. Items seen on the head-worn display can be selected with the same approximation to gaze-oriented selection employed in the Touring Machine UI. A menu on the head-worn display can be manipulated using a two-button trackpad mounted on the back of the hand-held computer for easy "reach-around" selection.

The head-worn user interface consists of a screen-stabilized part and a world-stabilized part. The menu bars on top of the screen and the cone-shaped pointer at the bottom (shown most clearly in Figure 5.7a) are screen-stabilized and therefore always visible. World-stabilized material is visually registered with specific locations on campus. World-stabilized 3D elements are displayed in the correct perspective for the user's viewpoint, so the user can walk up to these elements just as they can to physical objects

5.1.1.1 Navigating the Web of Presentations

Our situated documentary begins with a narrated introduction, explaining that the user will be able to learn about events related to the campus, and referring the user to the hand-held display for an overview. Before turning to the hand-held computer, the user looks around and sees virtual flags with textual labels denoting points of interest, positioned around the campus (see Figures 5.1 and 5.7). The virtual flags are world-stabilized user-interface elements that are iconic representations of the topmost *group nodes* in a hierarchical presentation. The hand-held display provides an overview of the material embedded in the surrounding environment. Looking at the surrounding flags, the user can see how the different stories are distributed over the campus area. The labeled flags come in three different colors: red for the student revolt, blue for the tunnel system, and green for the Bloomingdale Asylum.

The user can select a flag in several different ways. One method, which works when the user is in the system's *VisualSelect* mode (described above in Section 5.1.1), is to look in the flag's direction, orienting one's head so the desired flag's projection is closer than any other to the center of the head-worn display and within a fixed target area. When these criteria are met, the flag's label changes color to yellow. If the criteria hold for a half second, then the flag is selected and its label changes color to green. Flags are selectable from any distance. Although the flags scale with distance, their textual labels do not, so there is always a visible anchor that is selectable. A second selection method is based on positional proximity. A menu item allows the user to ask the system to select the flag to which they are currently closest (or to select another flag from a list), and the cone-shaped pointer on the head-worn display will point towards that flag, guiding the user to it. Finally, a flag can be selected automatically by following a link in the presentation.

When a flag is selected, it starts to wave gently, and all flags of a different color are dimmed (reduced in intensity). Therefore, when a user looks around while a flag is selected, the other flags in its category stand out. The cone-shaped pointer always points toward the selected flag, so that the user can be guided back to it should they look away.

Selecting a flag causes the second menu bar (the green *context* menu below the blue top-level menu) to display that flag's label plus additional entries that are available for its group node (e.g., links to other group nodes). All these entries can be selected using the trackpad.

The group nodes (and their corresponding flags) have a default numbering corresponding to an order set forth in the presentation description. A button click on the trackpad directs the user to the next node in this order; however, at all times the user can choose to select a different flag using any of the methods mentioned above.

In our case, the user selects the entry for the student revolt from the overview

Figure 5.2: Situated documentary about the Columbia student revolt of 1968: Documentary photographs and newspaper clips are animated into the user's view, synchronized with a narration of events at the selected site.

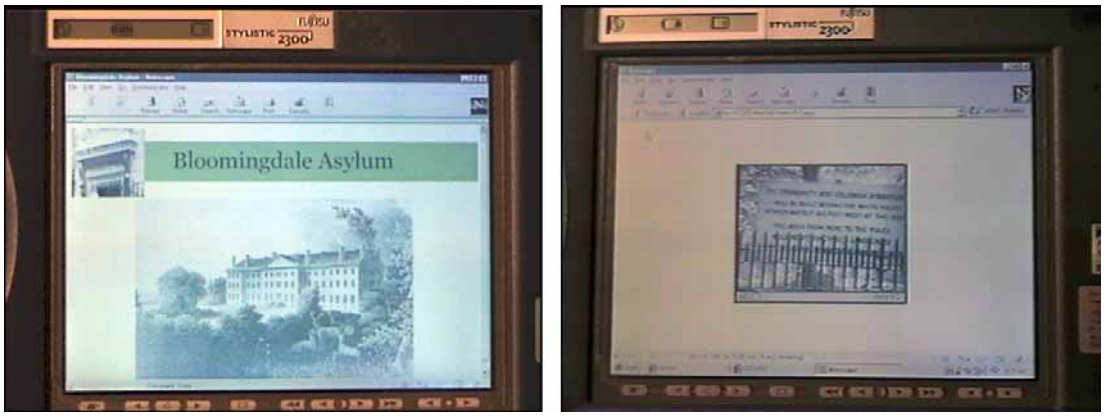
menu on the hand-held computer. The cone-shaped arrow on the head-worn display points to a red flag, which starts waving, in front of Low Library, which is about 150 yards away. This flag is the starting point for information on the student revolt.

Once a flag is selected, the user can display an overlaid *in-place* menu (see Figures 5.6 and 5.7), which lists the parts of the presentation associated with the flag's group node. (Section 5.1.1.3 discusses the in-place menus further.) The in-place menu for Low Library's revolt flag provides access to background information on how the student revolt started, grouped into five segments.

Selecting an entry in this menu using the trackpad starts that entry's part of the multimedia presentation, each of which ranges in length from seconds to minutes in our current material. Here, the user selects the entry labeled *First Clash*. This results in a narrated description of how the students and the police clashed for the first time on the steps of Low Library, where the user is now looking. The presentation includes coordinated still images that are overlaid on the scene (Figure 5.2) and videos that are played on the hand-held computer (Figure 5.3b).

The head-worn display's menu bar allows the user to display an overview of the student revolt on the hand-held computer, or to follow links to other places directly by selecting them with the trackpad to learn more about the revolt and what happened at other campus buildings.

At this point, the user has found a description of how the students used Columbia's tunnel system to occupy buildings guarded aboveground by the police. The user decides to follow a link to learn more about the tunnels by exploring the blue flags. Since the real tunnels are difficult (and illegal) to enter, the user can vicariously explore portions of them through a set of 360° omnidirectional camera photographic images (Figure 5.4)



(a) (b)

Figure 5.3: Images and video material displayed on the hand-held computer. (a) Imagery of the Bloomingdale Asylum. (b) Video material of the 1968 student revolt.

that temporarily teleport the user underground, supplemented by maps and blueprints.

The presentation mentions that the oldest parts of the tunnel system preceded Columbia's move to the area and were originally built for the Bloomingdale Asylum. Intrigued, our user turns to the green flags to find out where the main asylum buildings were situated, and is shown a 3D model of the buildings overlaid in place on the campus, in conjunction with historical images (see Figure 5.5). The documentary mentions that one building built for the asylum is still standing and is now known as Buell Hall, and points the user toward it.

5.1.1.2 Multimedia Presentations

The multimedia material in each presentation node is a coordinated media stream that typically, but not necessarily, makes use of both the hand-held display and the head-worn display, and which includes an audio track. The different media that can be freely combined to create a multimedia presentation are:

- *Audio material on the head-worn display.* Audio is played over the head-worn display's earphones, and includes both narration and non-speech audio (e.g., recordings of the 1968 revolt).

- *Images on the head-worn display.* Images (e.g., Figure 5.2) are displayed as world- or screen-stabilized 3D textured polygons that can make use of simple animated effects. For example, we often "flip up" screen-stabilized images from a horizontal position until they fill the screen.

(a) (b)

Figure 5.4: Exploring Columbia's tunnel system: (a) Schematic view of how a user experiences an omnidirectional camera image. (b) The omnidirectional camera image seen from a user's perspective.

- *Web pages that include static images, video material, and applets on the hand-held display.* Figure 5.3 shows examples of images and video, called up as related material on the hand-held browser as part of synchronized multimedia presentations.

- *3D models.* Figure 5.5 shows an example. Models are shown full-size and world-stabilized in their actual location.

- *360° omnidirectional camera surround views.* These allow us to immerse the user in an environment that is not physically available. We use a commercial omnidirectional camera (Nayar, 1997): a digital camera pointing at a parabolic mirror that captures a 360° hemispherical surround view in a single image. Each of these anamorphic images is texture-mapped onto a hemisphere displayed around the user, as depicted schematically in Figure 5.4a, so that the user can look around (Figure 5.4b). The see-through head-worn display's opacity is controlled by a dial, allowing us to make the display opaque when viewing these images. We are still waiting for displays whose opacity can be controlled by software.

Figure 5.5 shows how multimedia presentations can be made interactive. The history of the Bloomingdale Asylum, the former occupant of Columbia's present campus, is presented through 3D models of its buildings overlaid on the head-worn display. On the hand-held display, the user can select different milestone years in the asylum's history on an interactive timeline (realized by a Java applet) and on selection the corresponding buildings and narrations will be presented in the user's view of the area. This way the user can virtually travel in time.

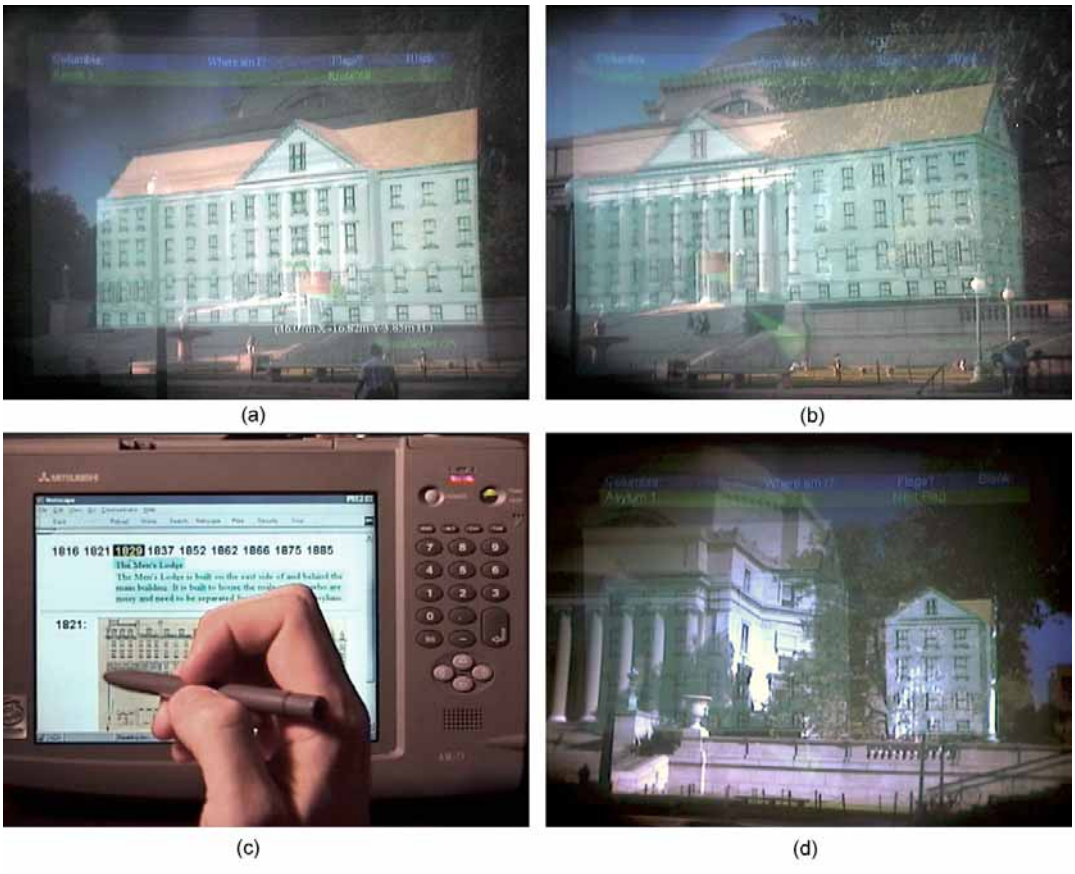


Figure 5.5: (a)–(b) A 3D model of the main Bloomingdale asylum building overlaid on Columbia's campus by the see-through head-worn display. (c) An interactive timeline displayed on the hand-held computer when user selects the year the Men's Lodge was built. (d) The AR scene responds by displaying the Men's Lodge and fading out the main asylum building.

Figure 5.6: Original menu design for context menus, listing multimedia snippets about the 1968 student revolt. World-stabilized circular menu around Low Library (photographed through 1997 Touring Machine head-worn display).

5.1.1.3 Exploratory UI Design

We also use omnidirectional images as backdrops for indoor demonstrations of our system and for exploratory development of new user interface elements and variants. Figure 5.7a demonstrates this approach. Figure 5.6 shows our original version of an in-place menu, shot outdoors through a low-resolution see-through head-worn display; Figure 5.7 shows our updated version of the same menu: We changed the menu to be screen-stabilized instead of world-stabilized, so that head movements would not interfere with the selection process. An anchor line connects the menu with the place in the real world that the menu originated from, so that the user can easily find their way back to that spot at all times. Figure 5.7a was captured as a screen dump of the system running indoors, using an omnidirectional image of the campus as a backdrop, Figure 5.7b was shot through an SVGA-resolution head-worn display outdoors.

In the latter design, the menu is a screen-stabilized element, rather than the world-stabilized circular menu of part (a). A leader line (Feiner et al., 1993b) links the menu to its associated flag, allowing it to be followed back if the user turns away from the flag.

5.1.2 Situated Documentary III: Interactive Storytelling

Our third example of the situated documentaries has a slightly different look and feel than the previous two, since Situated Documentary III were built on top of a new platform, JABAR (JAva-Based AR) (see Section 3.2.3). This new development platform allowed us to explore several new UI mechanisms that were easier to implement, now that we were using Java3D and a set of newly implemented UI support libraries. Examples of new interaction techniques include picking up and dropping off 3D objects, mouse-over

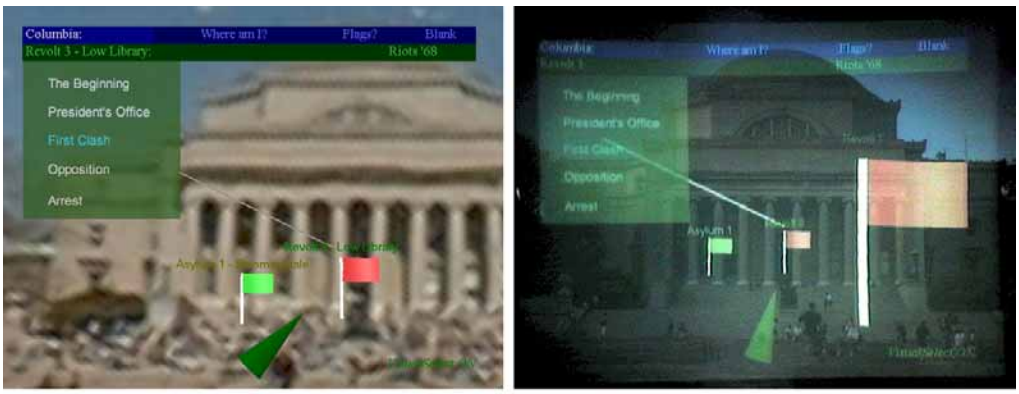


Figure 5.7: Alternative menu design for context menus, listing multimedia snippets about the 1968 student revolt. (a) Screen-stabilized list with anchor to its flag (screen dump of the system running in indoor test mode, with an omnidirectional image as a backdrop). (b) Same menu with outdoor tracking, photographed through 1999 MARS head-worn display.

actions for physical and virtual objects, and billboarded images and texture-mapped text (always oriented towards the user).

Conceptually, we generalized the notion of flags as the top-level representations of nodes in a spatially distributed web of interconnected points of interest. While re-implementing the previous situated documentaries on the new software platform, we experimented with different alternative representations, including a set of interconnected buoys, and a more iconoclastic approach, in which there were no actual world-stabilized representations of presentation nodes anymore, but instead screen stabilized symbolic representations of the story thread(s) in whose spheres of influence the mobile user was at each point in time. Thinking about how a user might be guided to important chronological events, while still taking an active part in experiencing a story, we followed the notion of graphical adventure games in which players have to collect inventory items that help them solve puzzles at later stages in the game.

An example of this approach can be seen in Figures 5.8 and 5.9, that show scenes from a situated documentary about the beginnings of the Manhattan Project. Research performed at Columbia University in the late 1930s and early 1940s had significant impact on this project, which later culminated in the construction of the atom bomb. Both figures, 5.8 and 5.9, were captured in indoor simulation mode with omnidirectional photography providing (comparatively low-resolution) background for the different locations relevant to the story. In Figure 5.8, the user has just dropped a small model of the Columbia cyclotron, an early type of particle accelerator, into the scene in front of him/her, thereby triggering the display of an up-to-scale world-stabilized model of the

Figure 5.8: Situated documentary on the Manhattan Project work at Columbia University. Information on Cyclotron used in experiments in the basement of Pupin Hall.

cyclotron and a narration played back on the MARS's head phones. The small cyclotron icon had been collected in a previous part of the story and now showed up in the "well" interface element at the bottom of the screen to signify relevant content in this location, should the user wish to explore it further. Once dropped into the scene, the user can explore the cyclotron model at will by walking around the world-stabilized full-scale model that popped up in the place where the icon was dropped. At the end of this story thread, the user is enabled to pick up a model of the first constructed atomic bomb.

Figure 5.9 shows one of the possible continuations of the story. Here the user was teleported to the roof of one of the tallest buildings on Columbia's campus. This part is always shown in VR mode with an omnidirectional backdrop, since the user can hardly be expected to climb up the roof as part of the documentary. For this particular location we combined two hemispherical backdrops to form a complete 360° background sphere surrounding the user, since it is important for the user to look both upward and downward.

When dropping the previously collected bomb model into the scene at this location, a full-scale model of the first atomic bomb, Fat Man, shows up in front of the user (Figure 5.9a). Moving a wirelessly controlled mouse cursor over the various objects in the scene (Columbia's campus buildings, the Empire State Building, and the bomb model) triggers explanations and background information. The bomb model activates a part of the story that explains about the impact of an atomic bomb hitting a metropolitan center. Clicking on the bomb model causes the bomb to accelerate towards the Empire State Building, and once it hits its target it sets off a simple simulation of the explosion, indicating the growth of the mushroom cloud and spreading of the shock wave in real time (Figure 5.9, (b) to (d)). This work was done in Spring 2001 to illustrate the terror

Figure 5.9: Situated documentary on the Manhattan Project work at Columbia University. Simple simulation of mushroom cloud that atomic bomb “Fat Man” would have caused when dropped in midtown Manhattan (Empire State Building area).

of nuclear attacks, half a year before the unspeakable terrorist attacks of September 11th brought down the World Trade Center.

5.1.3 Summary of UI Concepts

We can group the main innovations of the situated documentaries UIs into two main categories: New UI components and new interaction techniques. We explored the use of the following new UI components:

- Virtual flags as world-stabilized 3D elements that the user can walk up to just as they can to physical objects. Flags are iconic representations of the topmost group nodes in a hierarchical presentation.

- New contextual in-place menus (replacing the semicircular label arrangement of the Touring Machine). The menu is now a screen-stabilized object but anchored to its origin in the world by a flexible leader line (see Figure 5.7).

- Multimedia presentations as synchronized temporal MARS elements. For example, a multimedia presentation may consist of audio narrative, still images that are "rolled in" onto the head-worn display, and video material shown on the hand-held computer. It may also include world-stabilized 3D objects.

- More complex 3D animations in the AR world, such as for example the nuclear bomb explosion sequence mentioned above. As mentioned above, increased use of animated world-stabilized graphics supports the illusion of the augmented space as an integrated mixed reality.

- Billboarded images and texture-mapped text. In the Touring Machine UI, all text was realized using 2D bitmapped fonts, resulting in labels that would not change size based on the distance to the viewer. Texture-mapped text and billboarded images (2D images that automatically reorient themselves towards the user) provide new possibilities for UI design.

- Full omnispherical backdrops. We use these to implement virtual vantage points, by means of which we can more effectively convey scenes that are not otherwise accessible to the user (compare bullet on "teleportation" below).

The following new interaction techniques were implemented and evaluated informally as successful:

- Additional selection mechanisms for points of interest (represented by 3D flags): a) based on positional proximity to these icons. b) from overview lists and maps on the handheld computer. c) via links from other story elements.
- A hypermedia structure between the visual elements of the MARS UI. Links between group nodes and between presentation nodes and group nodes can be followed via the menu system.
- The concept of being *teleported* to a virtual location that is shown via omnidirectional photographs. This forms a link between augmented and virtual reality, similar to concepts later explored by Billinghurst et al. (2001).
- Collecting, dropping, and discarding of 3D objects via 2D–3D drag and drop. Together with a 2D dashboard interface that allows the user to exchange data with the 3D world, and inspect big objects in full size by dropping them before them and walking around them.
- User guidance through a web of stories via a collected item metaphor. This uses the previous interaction techniques to implement "adventure game style" semi-guided navigation.
- Mouse-over events and "tooltip-style" annotations. Without making selections, the user can inspect physical and virtual objects by moving the mouse pointer over them.

5.2 Collaboration

AR UIs invite collaboration. Several users can discuss and point to virtual objects displayed in a shared physical space (Billinghurst et al., 1998c; Billinghurst et al., 1998d; Butz et al., 1999; Reitmayr and Schmalstieg, 2001a). At the same time, every participant can see their own private version of the shared data, for example to see annotations optimized for their specific viewing angle (Bell et al., 2001). Mobility opens up new avenues of collaboration and a demand for computer-support for this kind of field-based teamwork (Luff and Heath, 1998).

5.2.1 Indoor/Outdoor Collaboration

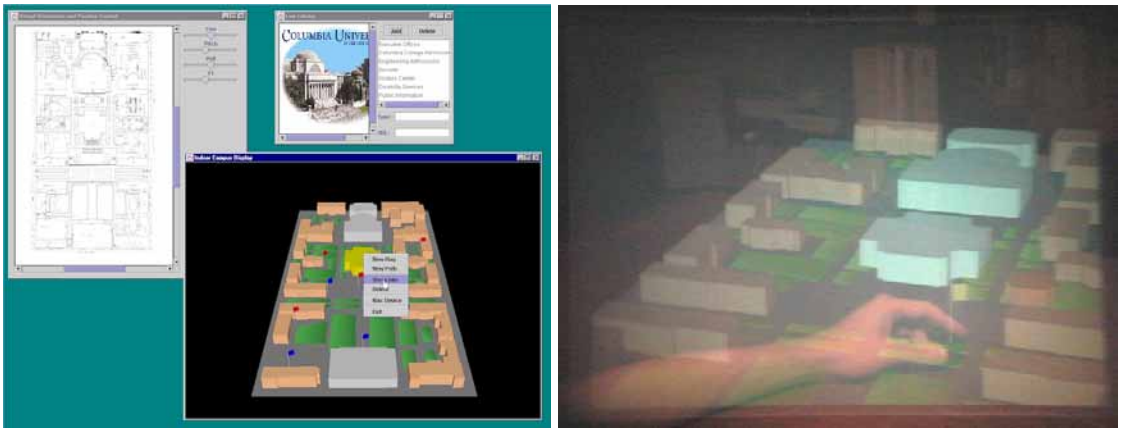
A wearable UI alone is not enough to fully capture the potential of a world-wide layer of spatialized information. For various tasks, a stationary computer system will be more adequate, especially for those applications whose UIs work best with physically large

Figure 5.10: The hand-held computer with a map interface.

displays. Among these applications are tools, especially collaborative ones, for authoring the information layer, for obtaining a broad-scale overview of relevant information, and for playing back logs of user interactions with the augmented world.

In this section we describe some of our design choices for several UIs facilitating indoor/outdoor collaboration. Two stationary indoor UIs and one handheld-based mobile UI complement the basic situated documentaries UI described above. For the hand-held display, Gus Rashid, under supervision by the author of this dissertation, implemented a map-based UI (Figure 5.10), that can be used either in conjunction with the backpack computer and glasses, or standalone. For indoors, the author guided and supervised the design of a desktop or projection-display UI (Figure 5.11a), implemented by Tachio Terauchi. This UI, based on a 3D environment model of the campus, lets users create virtual objects and highlight and annotate real objects for outdoor users to see, and maintain histories of outdoor users' activities; in turn, outdoor users point out interesting objects and events for indoor users to view. An immersive version of the indoor UI (Figure 5.11b), also implemented by Tachio Terauchi, relies on see-through head-worn displays, in conjunction with 6DOF head and hand trackers, and 3DOF object trackers, to overlay and manipulate virtual information on and above a physical desk.

We designed the map based mobile UI to be run on a pen-controlled hand-held computer, that can be used either in conjunction with the outdoor MARS, or standalone. The "map" is really a three-dimensional model of the environment, derived from the same data that we use in our indoor visualizations of the campus. Figure 5.10 shows a top-down view onto this model, emulating a common 2D map interface. Users can review their current position on the map; they can select arbitrary buildings, which, if used in conjunction with the AR interface, will become selected in their head-worn display as well, so they can be directed towards it. Since we designed our hand-held and AR



(a) (b)

Figure 5.11: The Indoor Command Center Interface. a) Desktop UI. b) Immersive augmented reality UI.

UIs to use the same distributed 3D graphics base, we can easily visualize data exchange between the two displays, for example drag and drop mechanisms for 3D objects, as explored by Butz et al. (1999) (cf. Section 5.2.2).

Our immersive indoor UIs have the users wear head-worn displays and tracks their head position and orientation with a 6DOF tracker. We run it in two different modes: as a completely virtual environment and in augmented reality mode overlaid on top of a physical desk. In both cases, our main input devices are wireless trackballs whose 3D position is tracked with the equally wireless 3DOF position sensors of our tracker.

An example of the former setup is depicted in Figure 5.12(a). The helicopter in the picture is slaved to one of the tracked trackballs, and is used to create a path on the ground of the campus model, which is displayed in overview mode, i.e. from a bird's eye perspective. The other setup is operating with the head-worn displays in optical see-through mode, thus working as an augmented reality system. It projects the 3D campus environment onto a physical desk and lets users control it by means of the same position tracked wireless trackballs mentioned above. In another interaction mode users can move small physical props around on the physical desk to place virtual information. Figure 5.11 (b) shows a user moving around a virtual flag, which is linked to a position sensor that the user holds in his hand. Like most AR pictures in this thesis, the picture was taken by a camera embedded in a dummy head wearing the head-worn display. The effect that we get with this kind of UI is similar to the experience in front of a virtual workbench (Krueger and Froehlich, 1994), with the advantages that we can provide for an arbitrary number of stereo views onto the same environment and that we can make local modifications to each participant's view of the scene, e.g. for accommodating for privacy issues (Butz et al., 1999).

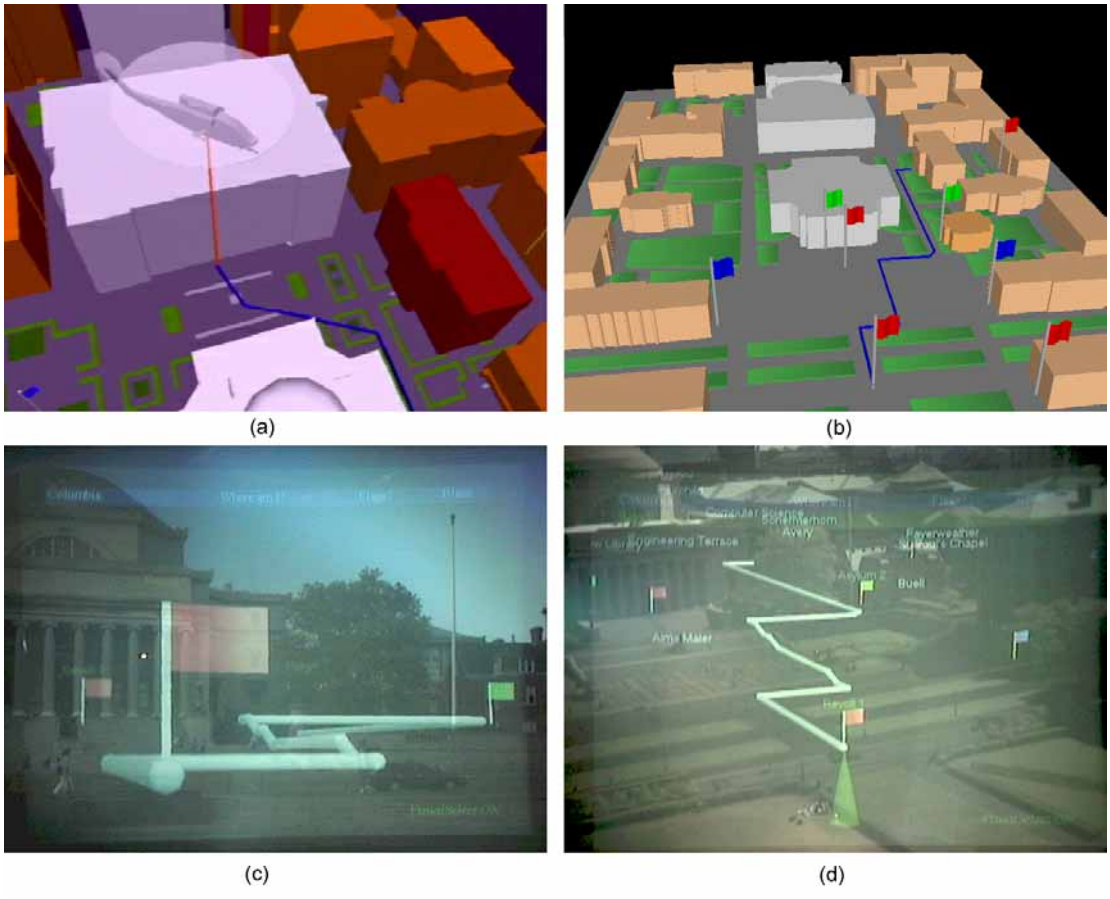


Figure 5.12: Trails: (a) Creating a trail in the indoor immersive UI. (b) Trail in the desktop UI. (c) Trail on campus, seen from ground level. (d) Trail on campus, seen from above.

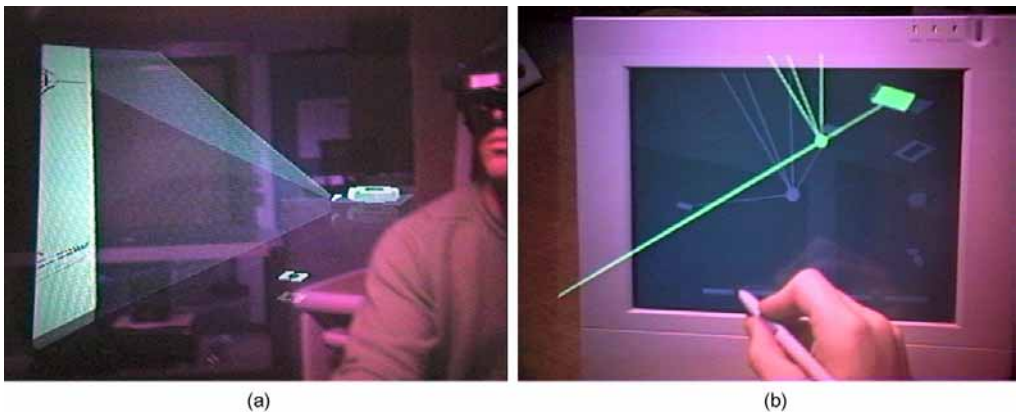


Figure 5.13: EMMIE users controlling different 3D AR interfaces. (a) A virtual "slide projector" displays images after different slide icons were dropped onto it. (b) A simple interactive search mechanism on the tracked display, which mirrors the virtual objects in front of it, creates 3D leader lines to objects satisfying the query.

Example of indoor–outdoor interaction Our different UIs for the augmented reality environment offer various opportunities for indoor/outdoor communication and collaboration. An outdoor user, walking around, is represented by an avatar in the indoor and map based UIs. New virtual objects can be introduced by either UI and, when moved around, their position is updated in all participating UIs. This can be used, for example, to highlight points of interest for all collaborating parties to see.

Figure 5.12 shows another example of indoor/outdoor interaction: an indoor user giving guidance to a roaming user by drawing a trail on the virtual model (part (a) and (b)). Part (c) shows how an outdoor user experiences the same trail appearing before him or her. The picture in (d) shows the same path photographed from the top of a campus building.

5.2.2 Hybrid UIs for Augmented Meetings

While the situated documentaries environment was our main driving force for UI innovations outdoors, we explored new AR interfaces indoors using the EMMIE (Environment Management for Multi-user Information Environments) application concept. The EMMIE system (Butz et al., 1999) provided a prototype experimental user interface to a collaborative augmented environment. It was chiefly designed and implemented by Andreas Butz on top of Blair MacIntyre's Coterie/Repo-3D platform (MacIntyre and Feiner, 1998), and co-designed and later extended by the author of this thesis.

The idea behind EMMIE is that of a 3D *environment manager* (in analogy to 2D window managers) that is distributed across displays, machines, and operating systems.

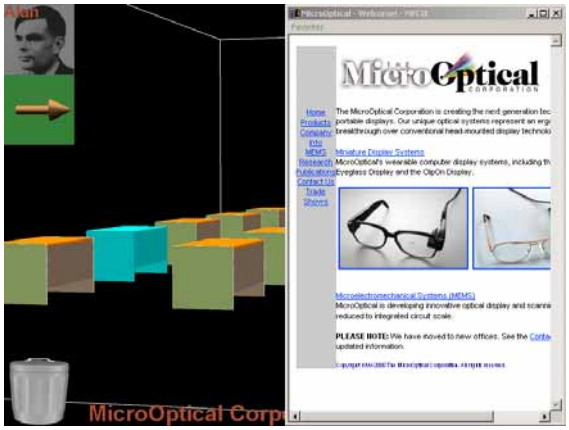


Figure 5.14: Screenshot (non-overlay) of JABAR interface shown at ISWC 2000. The graphics were overlaid on top of a trade show environment. The view shown was for giving a "through-walls" overview of the trade show from outside the hall. We display virtual desks for each booth. On the right-hand side of the screen is an integrated web browser showing information about the currently selected exhibitor.

Note that, while the environment management component does offer interactive services such as drag and drop support among different displays as well as within the AR world (Figure 5.13a), tools for privacy management, or a 3D leader-line-based search function for virtual objects (Figure 5.13b), it does not yet tackle automated dynamic UI management, which we will discuss in Section 5.4.3 and Chapter 6.

5.2.3 Integrated Web Browser and Communication

Based on the JABAR platform (Section 3.2.3), and influenced by the Situated Documentaries III interfaces discussed above (Section 5.1.2), we designed an indoor interface for information browsing and peer-to-peer communication. Figure 5.14 shows the interface as it was presented at the IEEE ISWC 2000 conference trade show. This is a screenshot of the overlay graphics, rather than a photograph through the head-worn display. At the particular time the screenshot was taken, half of the screen is occupied by a web browser, showing information about the currently selected item in the augmented world, which consists of a double row of booths, represented by virtual desks. This particular shot was taken from the outside of the tradeshow ballroom, looking in, with most of the booths hidden from view by the ballroom walls. The AR view enables an overview of the layout of the exhibition booths and their respective exhibitors. Mousing over the virtual desks reveals the names of the exhibitor (displayed at the bottom of the screen), and, if that mode is selected, triggers the exhibitor's web site to be displayed on a web browser that can be adjusted to cover a certain part of the screen. The web browser can be freely

controlled by any actions in the AR part of the interface, courtesy of a control interface implemented by Blaine Bell.

Another feature of this interface is personalization and communication between different online users. When "logging on" to the system for the first time, the user can have his photograph taken with a small digital camera that is part of the wearable system, or provided in the setup area. The user's name is also entered. Now the system can facilitate communication messages – in this case text messages with the corresponding image displayed next to them. These are exchanged via our Java-based distribution infrastructure (Section 3.2.2).

5.2.4 Summary of UI Concepts

In summary, we created and tested the following new UI components and interaction techniques in our collaborative MARS UIs:

- A simple 3D environmental model of our campus, used as a 3D overview map in indoor and handheld interfaces (and also for outdoor registration of annotations with physical objects). This model allowed indoor users to virtually navigate the same space as outdoor users. Exploring the interaction techniques made possible by an overview map view gave us ideas for how to best integrate overview visualizations (world in miniature views) into our mobile AR UIs. (cf. Section 5.3).

- Indoor/outdoor creation and movement/placement of objects, shared between different clients. Our overview model bacame the common reference frame for indoor/outdoor collaboration. Creation and movement of flags and paths in the indoor UIs was reflected outdoors, and outdoor activity was visualized indoors (avatar representing mobile user and paths visualizing the user's movement).

- Selection of buildings and creation of objects from the handheld map, causing effects (highlights, new objects) in the AR view. We facilitated the use of the handheld computer as the "control center," otherwise reserved for the indoor interfaces. The idea was to give the mobile user all the overview manipulation tools that would transfer well to the smaller form factor. In effect, the handheld computer became a physical WiM.

We implemented the following UI support structures as part of collaborative interfaces, but they are really general-purpose lower-level UI components:

- Live video textures in AR view. As an extension to the multimedia support previously implemented in the situated documentaries framework, we provided additional support for displaying videos in the 3D AR space, realized as dynamic 3D

texture maps. As an example from the EMMIE UI, a “movie player” application item could project a “film roll” data item onto a virtual screen.

- Live web browser in AR view, able to be controlled via AR interaction. Previously, we showed web pages on a separate handheld computer only. With increased head-worn display resolution, it turned out to be feasible to pop up a web browser in part of the AR view on demand where it can provide information (based on interactions with the AR world) as a direct annotation rather than as indirect background information.

5.3 Navigation

We have already emphasized the use of MARS techniques for human navigation in Sections 2.3.1.3 and 2.3.1.4. Digital information can more flexibly and interactively be adapted to a person’s navigational needs than, for example, the offline-generated area overviews provided by paper maps. Vehicle-based navigation has already adopted information technology in the form of GPS-guided in-vehicle navigation systems. Mobile AR offers the potential for providing navigational aids to humans walking in the field, directly superimposed on their view of the world.

5.3.1 Indoor Navigation UI

The experimental adaptive mobile AR UI that we describe in this section is intended to assist a user in navigating through an unfamiliar environment indoors. It uses the MARS 2000 hardware platform (Section A.3), and was implemented on top of JABAR, our Java-based AR infrastructure (Section 3.2.3).

We track the orientation of our mobile user with an InterSense IS300 Pro hybrid inertial/magnetic tracker. We can track both the user’s head and body orientation by using head-mounted and belt-mounted sensors. As we reported before, we have to switch off the magnetic component of the tracker and rely on purely inertial orientation information to avoid being affected by stray magnetic fields from nearby labs when walking around indoors (cf. Section 3.3.1).

As mentioned in Section 3.3, our system relies on different technologies for tracking a user’s position in two different circumstances: within part of a research laboratory served by a high-precision ceiling tracker, and in indoor hallways and rooms outside of the ceiling tracker range. Our ceiling tracker is an InterSense IS 600 Mark II with wireless ultrasonic SoniDisk beacons. It covers an area of about ten by ten feet. Outside of this area, we use the hybrid dead-reckoning scheme described in Section 3.3. The system

Figure 5.15: Augmented reality user interface in accurate tracking mode (imaged through see-through head-worn display). Labels and features (a wireframe lab model) are registered with the physical environment.

can detect whether the beacon is in range of the ceiling tracker. Tracking accuracies and update rates vary widely among these position tracking approaches. The IS600 Mark II ceiling tracker can track the position of one SoniDisk to a resolution of about 1 cm at 20–50 Hz. Experimental evidence for our dead reckoning approach reveals a typical positional accuracy of 1–3 meters. Since the position updates occur in direct response to pedometer activity, the update rate is directly coupled with the user's step frequency (about 1-3 Hz). For comparison: The outdoor RTK differential GPS system has a maximum tracking resolution of 1–2 cm at an update rate of up to 5 Hz. The GPS accuracy may degrade to 10 cm, or even meter-level when fewer than six satellites are visible. If we lose communication to our GPS base station, we fall back to regular GPS accuracy of 10–20 m.

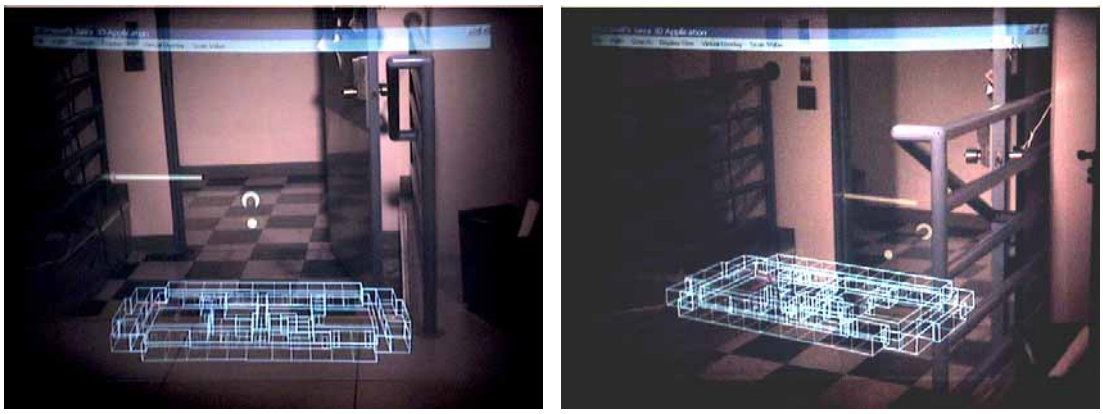
Our augmented reality user interface for navigational guidance adapts to the levels of positional tracking accuracy associated with the different tracking modes. Figure 5.15 shows a view through the see-through head-mounted display when the user is accurately position tracked by the ceiling tracker. The system overlays features of the surrounding room, in this case a wireframe model consisting of our lab's walls and ceiling, doors, static objects of interest (e.g., a rear projection display), and rooms in the immediate neighborhood. Labels are realized as Java 3D (Deering and Sowizral, 1997) Text2D objects: billboarded polygons with transparent textures representing the label text. Labels are anchored at their corresponding 3D world positions, so that closer objects appear to have bigger labels. The color scheme highlights important objects (e.g., results of a navigational query, and passageways from the current room to the main corridors).

When we roam with our mobile system—away from the ceiling tracker, but not yet outdoors where GPS can take over—we currently depend upon our hybrid, dead-reckoning system for positional data (cf. Section 3.3). As a result, we have relatively more accurate orientation tracking than position tracking. To leverage the relatively superior orientation accuracy in this situation, we have chosen to situate much of the overlaid material when roaming within the context of a World in Miniature (WiM) (Stoakley et al., 1995): a scaled-down 3D model of our environment.

Our WiM, implemented by Drexel Hallaway, has a stable position relative to the user's body, but is oriented relative to the surrounding physical world. That is, it hovers in front of the user, moving with him or her as he or she walks and turns about, while at the same time maintaining the same 3D orientation as the surrounding environment of which it is a model. In related work on navigational interfaces, Darken and colleagues (Darken and Cevik, 1999) explore different ways of presenting 2D and 3D map information to a user navigating in a virtual environment. They conclude that while there is no overall best scheme for map orientation, a self-orienting "forward-up" map is preferable to a static "north-up" map for targeted searches. The WiM is a 3D extension of the "forward up" 2D option in Darken's work. Because our WiM's position is body-stabilized, the user can choose whether or not to look at it—it is not a constant consumer of screen-stabilized head-worn display space, and does not require the attention of a tracked hand or arm to position it. If desired, the WiM can exceed the bounds of the HMD's restricted field of view, allowing the user to review it by looking around, since the head and body orientation are independently tracked. The WiM incorporates a model of the environment and an avatar representation of the user's position and orientation in that environment. It also provides the context in which paths are displayed in response to user queries about routes to locations of interest.

When the user moves out of range of the ceiling tracker, position tracking is shifted to the dead-reckoning tracker. To notify the user that this is happening, we first replace the registered world overlay with the WiM model, but at full-scale and properly registered. Then the WiM is interpolated in scale and position to its destination configuration (cf. Pausch et al. 1995).

Figure 5.16 shows the user interface just after this transition. Because the head–body alignment is relatively constant between these two pictures, the position of the projected WiM relative to the display is similar in both pictures, but the differing position and orientation of the body relative to the world reveal that the WiM is world aligned in orientation. These images also include real-world route arrows that point the way along a path to a location that the user has requested (in this case, the nearest stairway). Path calculation and display was implemented by Drexel Hallaway. As the user traverses this suggested path, the arrows advance, always showing the two next segments. The WiM



(a) (b)

Figure 5.16: Augmented reality user interface in DRM-tracked mode (imaged through see-through head-worn display). (a) A body-stabilized world-aligned WiM with world-space arrows. (b) The same WiM with the user at a different position and orientation.

also displays the entire path, which is difficult to see in these figures because of problems retaining fine-grain detail in images captured through the see-through head-worn display. We address the problem of the small size paths in our WiM in later interfaces introducing head-gesture control to get a closer view of the WiM (see Section 5.3.2.1).

5.3.2 Situational Awareness

As we explored above through the handheld map UI of Section 5.2.1 and the WiM of the previous section, it can be extremely helpful for a mobile user to get an overview visualization of their current location and neighborhood. Through annotations and/or live query tools that would allow the user to find out about anything that there is to know about a certain neighborhood, the user could gain an invaluable understanding of that location. He or she would gain locational, or, if the information included live updates on a changing environment, *situational* awareness. The question remains how to best call up such an overview visualization, how to control the zoom level and viewing angle in as straightforward a manner as possible, and how to best establish a link with the world in front of the user's eyes.

5.3.2.1 Intuitive Control of an AR World in Miniature

Blaine Bell implemented a simple head-pose-based technique to control a miniature model of the user's environment, embedded within his or her view. The author of this dissertation helped formalize and iteratively improve the design of the UI and explore its parameter space. The position, scale, and orientation of the model is controlled by head

(a) (b) (c)

Figure 5.17: Head-pitch control of WiM tool.

orientation alone. The resulting tool is designed to make it easy for the user in a mobile, and mostly hands-free, application to determine how much attention they wish to devote to the overview model. Also, annotations can be shared between the miniature model and the full-scale physical environment surrounding the user.

The user of our system is immersed within a full-scale physical environment, most of which is usually viewed directly through a stereo, optical see-through, head-worn display. Overlaid graphics can be enabled to annotate the surrounding world; for example, to label objects or to provide detailed information about them. The tool, which can also be selectively enabled, displays the user's environment as a similarly annotated, perspective-projected WiM, containing schematic virtual representations of the physical objects. The WiM is located significantly ahead of the user to make stereo fusion easier (currently 4m, determined pragmatically for our application). A red dot represents the user's current position, and objects in the WiM are drawn differently depending upon whether they are (partly) visible or fully invisible in the real world from the user's actual viewpoint, using an analytic visible surface algorithm.

We are interested in mobile applications that might require the user's hands to be free for real world tasks. Furthermore, accurate position tracking might not always be available. Therefore, we decided that the tool should be controlled by head orientation alone. The tool's yaw is fixed to that of the surrounding world, and thus changes directly with the user's head yaw (Darken and Sibert, 1993). In contrast, the position, scale, and pitch of the tool, as well as the decision to annotate its contents, are controlled by head pitch (Bell et al., 2002b).

Figure 5.17 shows our laboratory, viewed through our system, with the tool enabled. In (a), the user looks straight ahead and the unannotated tool stays close to the bottom of the viewport, pitched towards the view plane by a default angle (22.5°) about the user's position in the tool. In (b), the user looks halfway downward and the tool is scaled to a larger size, angled closer to parallel to the view plane, and moves up higher in the viewport. In (c), the user looks nearly straight down, and in response receives a

zoomed-in, annotated view, that is nearly parallel to the view plane, with the user's position in the tool located at the center of the viewport. The roll component of the tool's orientation is kept at zero, so that with regard to roll, the tool's ground plane is always parallel to the bottom of the viewport, but not necessarily parallel to the ground plane in the physical world. Note that the only effect of the user's position on the tool is to control the "you are here" point about which the tool orients (in yaw and pitch). This point is positioned at the center of the viewport's width and at a head-pitch-determined location relative to the viewport's height.

In summary, when the user is turning the head down towards the ground, the WiM is shown in more and more detail and from a more and more top-down perspective. This is reminiscent of a body-stabilized element that is stored approximately in front of the user's waist, but the mapping of the user's head orientation to the WiM's position and orientation can actually be freely chosen. In this specific case, the WiM is always kept visible on the screen, aligned in yaw with the surrounding environment it represents.

Figure 5.17 also illustrates how annotations are shared between the physical world view and the WiM. In part (a) a pop-up window provides information about the CPU cycle and memory usage of a lab computer. The annotation is attached to the monitor of the computer called "Rembrandt" via an arrow, without occluding any part of the monitor itself. It does so by using the annotation placement algorithms from Bell et al. (2001), knowing about the geometry of most physical objects in the lab. In part (b) of the figure, Rembrandt's monitor is not in the user's field of view anymore, hence the annotation switched over to the monitor's model in the WiM tool. As soon as the real monitor enters the user's view frustum again, the annotation will switch back to the real world. While the initial implementation of this annotation-sharing mechanism was implemented by maintaining the mapping between real-world objects and corresponding WiM objects with a lookup table that is generated when the WiM is first created, a much simpler and straightforward implementation of the hand-over process can be obtained with our rule-based architecture Ruby, as described in Section 6.2.3.

5.3.2.2 Restaurant Guide

In further work, tested in outdoor use, we implemented an online mobile AR guide to the restaurants in Morningside Heights, the area in Manhattan where Columbia University's campus is located. We covered forty restaurants in an area of sixteen blocks north-south along the two main avenues in the neighborhood, Broadway and Amsterdam Avenue. This area is sufficiently close to Columbia's campus to let us receive the differential corrections of our RTK GPS system (cf. Section 2.4.3). The area is bigger than the coverage of our campus-wide wireless network, hence we stored all information in a local SQL database on the mobile computer.

Figure 5.18: Simple building models overlaid on view of Broadway during registration tests for the restaurant tour guide.

In order to be able to place the annotations for all restaurants correctly with our view-management techniques (Section 5.4.3.2), we built simple geometrical models of all buildings with restaurants in them, and a few of the neighbor buildings in order to correctly take into account occlusion. The building models were extruded from aerial photographs of the Morningside neighborhood by Sinem Güven and Tiantian Zhou, with the height of buildings estimated from the number of floors. Some of the facades were texture mapped for display in an overview world in miniature of parts of the whole environment (cf. Figure 5.19d). Drexel Hallaway and Hrvoje Benko helped with the outdoor testing of the system. Figure 5.18 shows a test visualization we captured to check the registration of our building models with the building rows along Broadway. We increased the modeled building heights in response to this test.

Figure 5.19 shows a series of images of Tom's Restaurant shot through the head-worn display of our MARS 2002 prototype (cf. Appendix A.4). In part (a) the user is simply looking around and has the names of restaurants overlaid on the actual places in his or her current field of view. The labels are selectable via a wireless trackball, leading to the creation of a pop-up window with background information on the chosen place (Figure 5.19b). The pop-up window provides a brief description, a picture of the restaurant's interior, and links to the restaurant's menu, web page, and customer reviews. The pop-up window is automatically positioned so it does not obscure the restaurant's projection. The user can take a closer look at any of the choices provided by the pop-up. In part (c) of the figure he or she decided to take a closer look at the interior. Obviously, such functionality makes more sense when you can call up information on all the restaurants in the neighborhood, rather than just the once you are standing in front of anyway. Figure 5.19d indicates one way of doing so: The user has just brought up a world-in-miniature of the style described in Section 5.3.2.1. The WiM is centered around the user's current

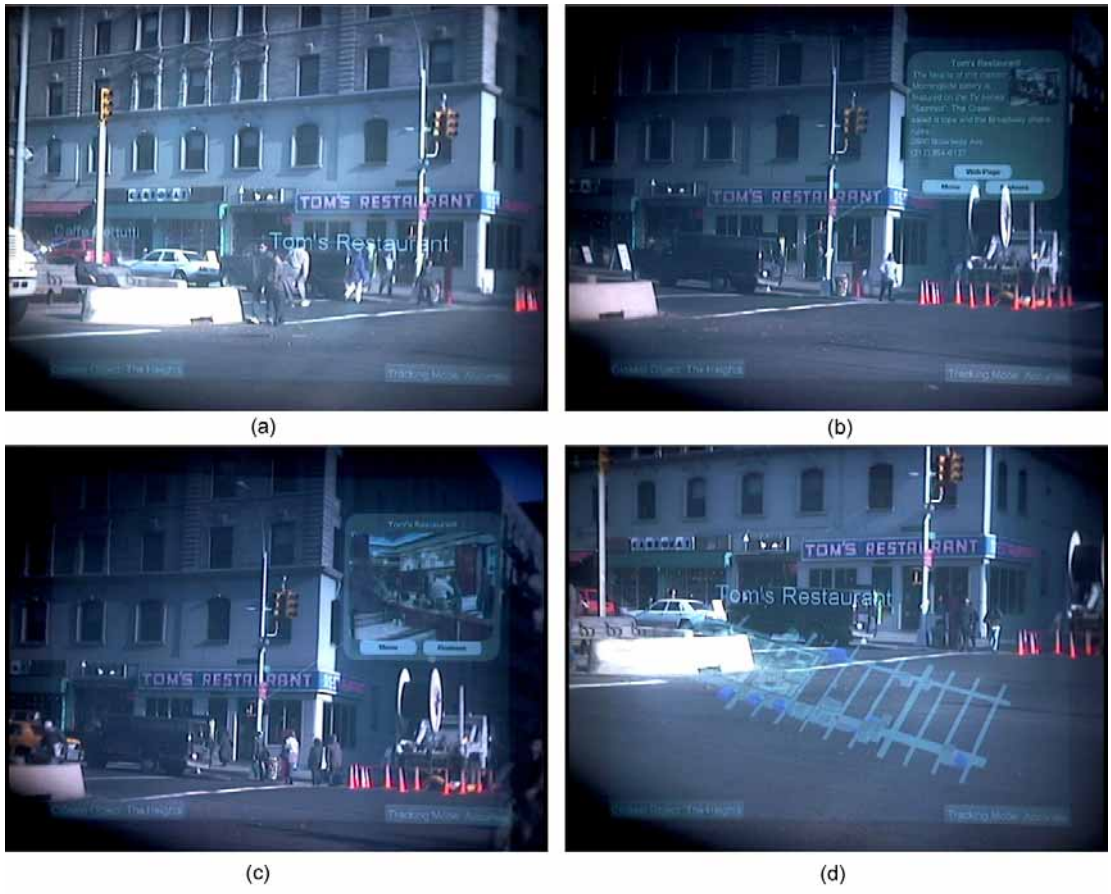


Figure 5.19: AR Restaurant Guide: (a) Labeling restaurants currently in view. (b) Context information on Tom's Restaurant. Pop-Up window avoids overlapping with physical view of restaurant. (c) Interaction with pop-up window: interior view of Tom's Restaurant. (d) WiM of current neighborhood centered around you-are-here marker.

location, marked by a red dot. In order to see all the restaurants within the WiM area highlighted and labeled, the user now just has to tilt their head further downwards and they will see an overview of selectable annotations.

5.3.3 Summary of UI Concepts

In summary, our work on navigational UIs introduced the following set of UI components and interaction techniques:

- World stabilized arrows and paths providing navigational guidance. These world-stabilized objects are temporary task-dependent annotations or highlights of conceptual objects representing navigational routes.

- AR display adapting to tracking accuracy (accurate ceiling tracker vs. approximate dead-reckoning tracking). The example we implemented here switches back and forth between two UI representations, based on the tracking technology currently in reach. In Section 6.2.3 we will discuss a solution that automates such an adaptation to a certain extent by employing a rule-based MARS.

- Screen-anchored objects, whose size, screen-position, and viewing angle can be controlled by user-defined mapping functions using head-pitch. This establishes a powerful interaction technique that keeps the mobile user's hands free for other important tasks. Related to this is the concept of a "body-stabilized" object, which requires the use of a second orientation tracker that tracks the body orientation as compared to the head motion.

- The concept of different representations of the same object (e.g. a physical table, and its virtual representation in the WiM). Annotations can be shared among these different representations of the same object. This principle will be formalized in our rule-based MARS framework, presented in Chapter 6.

- Selectable labels that result in pop-up information about the corresponding object. This information may include buttons and hyperlinks that can be selected by the user. Pop-up information should be kept relatively static with regard to the user's head motion since it is hard to read text that is jumping around. For interaction, screen-stabilization is even more important. These issues are further explored in our work on view management (Section 5.4.3.2).

Figure 5.20: AR view from an unusual vantage point (roof of a campus building), exemplifying some of the problems of general mobile AR UIs, such as limited field of view, visual clutter, information overload, unwanted obstruction, and misplaced labels.

5.4 MARS UI Design, Problems and Solutions

We have explored MARS UIs in a series of application examples. We designed mobile AR interfaces for such diverse application areas as a campus tour guide, historic architecture presentations, hidden infrastructure visualizations, support for building construction, indoor and outdoor navigational guidance systems, augmented group meetings, interactive documentaries, and a neighborhood restaurant guide. In this section, we will highlight the most severe MARS UI problems we observed while testing these UIs, and discuss approaches to solve them.

MARS UIs require a high degree of dynamic behavior. One of the main characteristics that sets them apart from other UIs is that they need to be able to adapt to arbitrary viewpoints of a physical environment, and annotate these in a world-stabilized fashion (cf. Section 4.1). A good approach to highlight some of the difficulties of such an approach is testing a UI from a vantage point that was not a primary consideration at UI design time. Figure 5.20 shows such an example. Here, we took the MARS to the roof of a campus building to get an overview of the augmented environment. The resulting picture draws attention to the following potential problems of general MARS UIs:

- Inaccurate annotation layout. If annotation placement is based on simplified geometric assumptions, misplaced labels will make the interpretation of our AR images hard or impossible. The system that produced Figure 5.20 used a very simple

algorithm for placing building labels, based solely on the centroids of buildings. While the problems with that simple strategy were already noticeable when the annotations were viewed from ground level, it was the overview perspective that brutally revealed the inadequacy of the approach. Since occlusion is not taken into account, the labels “Avery”, “Schermerhorn”, and “Computer Science” for example all seem to refer to the same building (Avery).

- Unwanted obstruction of physical or virtual elements. In addition to the wrong associations between labels and buildings, we have the problem of overlapping labels. Annotations that overlap each other in unintended ways are likely to be useless and, worse, may even confuse the viewer. In Figure 5.20, the labels for “Computer Science” and “Schermerhorn” are placed dangerously close to each other.

- Lack of contrast. It is hard to predict all lighting conditions and background luminosities in an AR scene. In absence of sensors that tell the MARS about the current lighting of the scene in front of the viewer, we have to rely on heuristics. Labels of bright color are usually most easily distinguishable from the physical background, when an optical see-through headworn display is used. When, however, other bright virtual elements are displayed in the AR scene, such as the virtual building overlay in the top left part of Figure 5.20, annotations that happen to fall (partially) on top of such elements become unrecognizable unless more sophisticated contrasting schemes are used.

- Limited field of view. The image clearly reveals the problem of limited field of view imposed by current headworn display technologies. Only a very small window into the augmented environment is visible. The user needs to turn his or her head by a considerable amount in order to completely take in some overlay visualizations (such as the virtual building).

- Visual clutter. All of the above problems may lead in one way or another to the problem of visual clutter, the perception of the UI as a disorganized mess of unconnected elements. This is what we mainly have to guard against in MARS UI design. Since the viewing conditions are hard to predict, some UI mechanism needs to react to potential problems when they occur. The UI needs to become adaptive.

- Information overload. If too much information is displayed in a small window into the augmented environment, the viewer can easily be overwhelmed with information. This problem is exacerbated by visual clutter as introduced in the previous

bullet. Information filtering in AR, a selection process that aims to determine the subset of augmenting material most relevant to the user, is an important step to limit information overload.

In addition to these problem areas, there are several issues that become apparent when using a MARS UI as an interactive dynamic interface, rather than just looking at static example pictures. In order to convey the correct annotation relationships between computer generated material and physical objects at all times (which, as we just mentioned, is hard enough to do at all in the general case), the annotations may need to be shifted around to correctly take into account changing viewpoints and partial occlusions. Moving interface elements, however, can be quite distracting (Shneiderman, 1998). We have to be careful to not draw undue attention to not-so-important annotations that are shifting around. As we will report in the following section, our test users reported that this dynamic aspect of the UI is not so much a problem when carefully scanning an AR environment from a static location, but that it becomes distracting while moving around. Another problem that is connected with the dynamic nature of MARSs is that infrastructure support might vary considerably over the large areas that the MARS can be used in. For example, position tracking accuracy might change drastically when the user enters "urban canyons" and loses sight of GPS satellites. Being able to adapt to such situations is a great challenge for MARS UIs.

5.4.1 MARS UI Design: Lessons Learned

Our MARS UI explorations took place as an iterative process involving domain analysis, creation of suitable application scenarios, UI prototypes, and several rounds of expert and non-expert evaluations. We loosely followed the usability engineering process formalized by Nielsen (1994). Our new UIs built on the experiences and lessons learned from previous prototypes.

The three versions of situated documentary applications in particular are a good example of how we iteratively improved our MARS interfaces. We addressed a series of UI issues that were discovered through in-the-field testing. We already mentioned the change of contextual in-place menus from fully world-stabilized to world-anchored screen-stabilized (cf. Section 5.1.1.3. For an analysis of the concepts behind that change, see Section 5.4.2). We learned that menus should preferrably be screen stabilized, since involuntary head-motion and turns of the head due to distractions make the use of fully world-stabilized menus very difficult. The anchor concept keeps the advantages of world-stabilization, while eliminating the disadvantages. A leader line emphasizes the relationship between context menu and world object and can guide a user back to the origin of a context menu in case he or she looks away.

We also learned from user feedback that world-stabilized UIs are considered to be of little use to the user while he or she is actually walking. Users preferred to explore the world-stabilized overlays looking around while standing still. Some users reported that they found moving UI elements to be distracting when walking. Some of these disruptive motion artifacts can probably be attributed to jitter caused by the orientation tracker and would go away if world-stabilized annotations stayed completely static with regard to their world reference at all times. But the user comments go deeper than that. It is imperative that a walking person not be distracted from obstacles or other dangers. In later interfaces, we reacted to these findings by detecting the user's state with regard to body and head motion, and keeping the UI simple and screen-stabilized while the user is walking or looking around quickly (cf. Section 6.2.2). The limited field of view through the head-worn display was another user complaint that was deemed especially irritating during navigation, but this issue can only be addressed by better display technologies.

One of the most successful UI additions of the situated documentaries as compared to the Touring Machine UI was the introduction of an increasing number of world-stabilized 3D elements. We informally verified our expectation that elements such as our 3D flag icons would genuinely help the 3D spatial perception of the campus environment, as compared with the labels-only UI the Touring Machine had first explored. Flags are displayed as 3D elements with their size and perspective correctly adapted to the user's distance and viewing angle. They make it easier for the user to judge the relative location of these points of interest. Based on user feedback we also made plans to take into account the occlusion relationships between real objects and virtual ones (such as our flags). In all three situated documentaries applications, the flags are displayed even when they are positioned behind physical objects in the user's field of view. This allows a user to select and query a point of interest even if it is currently occluded from view. However, it decreases the user's 3D perception of the space and particularly the location of the flag in question. This observation led to UIs we later implemented, in which objects are displayed differently (e.g., dimmed) when they are occluded but should still be displayed (cf. Section 6.2.1; compare also (Feiner et al., 1993b)).

At the outset of the first situated documentary, the only world-stabilized 3D elements we used were the flags denoting points of interest. In the second iteration we added models of historic architecture (different buildings of the Bloomingdale Asylum complex), and in the third application we even let the users explore 3D models, such as the cyclotron and the "Fatman" nuclear bomb, in a world-stabilized fashion — in places the users themselves determined by dropping screen-stabilized icons into the 3D world. This proved to be very successful, simply because walking around such models and looking at it in its original size offers a more realistic impression than any indirect 3D manipulation can provide. In our third iteration, we also added animations to the

arsenal of world-stabilized graphics, thereby taking another step towards more realistic AR simulations.

We also iteratively improved the physical hypermedia interface. Color as the distinguishing property among flags that represent different story threads works well for a small number of threads, but does not scale. The screen-stabilized iconic story representations of the third situated documentary incarnation represent a more symbolic story identification. A combination of both approaches is possible, texture mapping the story icons onto the respective flags, but at a distance the basic visual appearances still need to stand out in order to let the user identify the story thread. Our attempts to provide the user feedback on the currently selected story thread by highlighting all flags in the current thread or displaying all unrelated flags in a different color had mixed results. In AR, small color changes are often not perceptible. We needed to dim the flags of unrelated story threads by quite a bit in order to make the effect noticeable. Overdoing the effect, on the other hand, can impair the possible distinction between unrelated story thread objects of different types.

The creation of an increasingly detailed geometric model of our campus, which was driven by the need for more fine-grain registration of annotations with physical features, led us to explore UI possibilities based on overview visualizations of larger environments. One example of this were the "command-center" and handheld UIs from Section 5.2.1. We moved aspects of these UIs into the AR realm, by revisiting the notion of WiMs (cf. Sections 5.3.1 and 5.3.2, iteratively refining the control of these UI components with a particular attention to mobile applications. Hands-free operation of MARS UI components is important, because the UI should be able to support users in different kinds of mobile situations, and many of these might require the use of the user's hands for other purposes.

5.4.2 Analysis of UI Alternatives

We have discussed various techniques that allow MARS users to interact with the augmented environment. One goal of our MARS component taxonomy from Chapter 4 was to create a theoretic framework that can explain the components of existing techniques, enable the techniques to be adapted to various new situations, and facilitate future design decisions.

As an example, consider the *fly-down menu* of Feiner et al. (1997) (cf. Section 5.1.1). This technique was designed to emphasize the causal relationship between UI objects on two different display devices: An event (menu item selection) on the head-worn display triggers an action (creation and display of a new web page) on the hand-held display. The motion of the menu item moving downwards and off the screen attracts the

(a) (b) (c)

Figure 5.21: Fly-down menus, as designed by Feiner et al. (1997). Label animation sequence after the ”Departments” menu item is selected and the department list for the Philosophy Building is displayed, arrayed about the building. (a) A fraction of a second after selection. (b) Approximately half a second later. (c) After the animation has finished.

user’s attention and directs it downward to the (default) position of the hand-held (cf. Figure 5.21).

An analysis of this technique within our taxonomy reveals various alternatives to the original implementation. The underlying abstraction for the task at hand is that a selection of a *functionality provider* (the menu item), which is in an *informs about* relationship with an *information container* (the information captured in the newly created web page), triggers the creation of some new UI component that is supposed to inform the user about the information at hand. Associated with this trigger event is the *goal* to convey cause and effect in selecting the functionality provider item. Depending on the *preferences* of the information container to be expressed in a certain *medium*, the *capabilities* of the *devices* the user has available, and possible other *constraints* the currently represented objects on the available displays impose, we might want to consider one of the following alternatives to the *fly-down menu* implementation:

- display the information on the head-worn display and convey the causal relationship with a simple animation, graphical link or just positional proximity to the selected menu item.
- present the information via audio and let temporal proximity clarify the causal relationship.
- present the information on the hand-held display (following the information container’s preferences to display a web page that was designed for use with an opaque display on exactly such a display), and convey the causal relationship by some other means of directing the user’s attention to the hand-held display (e.g. by audio).

For another example, please revisit the images of Figures 5.6 and 5.7. They show the same in-place menu options associated with the red flag in front of the columns, realized in two different ways. In (a), our first implementation, the menu was arranged in a circular world-stabilized fashion around the flag. This caused problems when the user turned his or her head during menu selection. In the design shown in Figure 5.7b, the menu is a screen stabilized element, linked back to its associated flag by a leader line, so that the user can easily turn back to it.

This kind of menu has to make visually clear that it is in an *attached to* relationship with the virtual flag in front of Low Library. Both of the solutions do that in their respective ways. We made the screen-stabilized menu semi-transparent, in order to more strongly emphasize that point. As a result, the view onto other virtual elements is not completely obstructed, which helps tighten the visual link between the menu and the leader line, as the line is more effective pointing to the center of the menu.

The reason that the first solution breaks down when the user happens to look away is that it violates the requirement that a selection interaction (like any other basic interaction) need to give constant feedback from start to finish of the interaction.

5.4.3 Adaptation

We believe that the benefits of MARS will only be achieved if the user interface (UI) is actively managed so as to maximize the relevance and minimize the confusion of the virtual material relative to the real world. We listed some of the biggest challenges of MARS UIs in the beginning of Section 5.4. Here, we present some of the steps we deem necessary to address these challenges, focusing on the design and layout of the mobile user's overlaid virtual environment.

The augmented view of the user's surroundings presents an interface to context-dependent operations, many of which are related to the objects in view—the augmented world is the user interface. We present three user interface design techniques that are intended to make this interface as obvious and clear to the user as possible: information filtering, UI component design, and view management. *Information filtering* helps select the most relevant information to present to the user. *UI component design* determines the format in which this information should be conveyed, based on the available display resources and tracking accuracy. For example, the absence of high accuracy position tracking would favor body- or screen-stabilized components over world-stabilized ones that would need to be exactly registered with the physical objects to which they refer. *View management* attempts to ensure that the virtual objects that are displayed visually are arranged appropriately with regard to their projections on the view plane. For example, the relationships among objects should be as unambiguous as possible, and physical

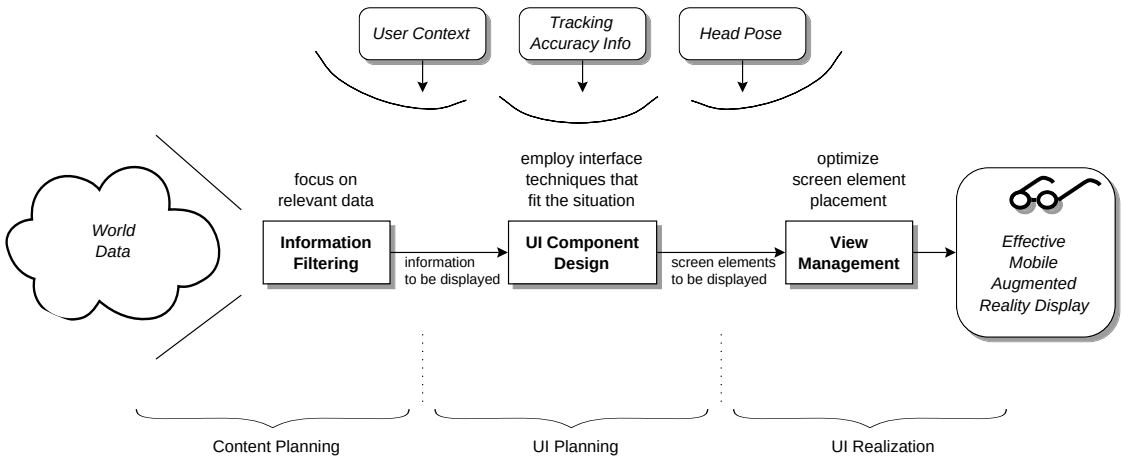


Figure 5.22: Information Filtering, UI component Design, and View Management as parts of a MARS UI-management model.

or virtual objects should not obstruct the user's view of more important physical or virtual objects in the scene.

5.4.3.1 UI Management Pipeline

As described in Section 4.1, MARS applications differ from most virtual environment applications in many ways, including the size of the physical environments that users navigate through, the importance of the physical environment and how virtual information is integrated with it, the quantity and range of virtual information that can be presented to and modified by users, and the potentially large variability in tracking accuracy over time. Based on our experience developing MARS testbeds, we have attempted to address these issues through a set of techniques for designing MARS UIs: information filtering, UI component design, and view management.

The large amount of virtual information that can be displayed, coupled with the presence of a richly complex physical world, creates the potential for clutter. Cluttered displays can overwhelm the user with unneeded information, impacting her ability to perform her tasks effectively.

We address clutter through information filtering. *Information filtering* means culling the information that can potentially be displayed by identifying and prioritizing what is relevant to a user at a given point in time. The priorities can be based on the user's tasks, goals, interests, location, or other user context or environmental factors.

While information filtering determines the subset of the available information that will be displayed, it is still necessary to determine the format in which this information is to be communicated, and how to realize that format in detail. Registration accuracy,

or how accurately the projected image of a virtual object can be positioned, scaled, and oriented relative the real world, is an important factor in choosing the right UI format. Registration accuracy is determined by tracking system accuracy, which, as the mobile user moves about, may vary for a variety of reasons that depend on the tracking technologies used. Therefore, if information is always formatted in a way that assumes highly accurate registration, that information will not be presented effectively when registration accuracy decreases. Tracking accuracy is just one example of user context that may influence the MARS UI composition. The fact whether the user is in motion or not, or if he or she is looking around fast or scanning the environment very deliberately, could have an influence on the UI. Any other information, such as noise level in the environment, or lighting conditions, might have a profound impact on what UI techniques can or cannot be employed. To address this issue, *UI component design* determines the format in which information should be conveyed, based on contextual information, such as the available display resources, tracking accuracy, and any sensory input about the user's current context. This technique determines the concrete elements that comprise the UI and information display.

Filtering and formatting information is not enough—the information must be integrated with the user's view of the physical world. For example, suppose that a selected set of annotations are simply projected onto the user's view of the world such that each is collocated with a physical object with which it is associated. Depending on the user's location in the world (and, thus, the projection that they see), annotations might occlude or be occluded by other annotations or physical objects, or appear ambiguous because of their proximity to multiple potential referents. *View management* attempts to ensure that the displayed information is arranged appropriately with regard to its projection on the view plane; for example, virtual or physical objects should not occlude others that are more important, and relationships among objects should be as unambiguous as possible.

Figure 5.22 shows these three steps in a MARS UI-management pipeline. Note that we do not claim that these steps form a complete UI-management model. Instead, we see them as subsets of the more general design phases of content planning, UI planning, and UI realization. *Content planning* determines the information that is to be conveyed to a user using presentation goals, user models, and online databases of information and taxonomic knowledge. *UI planning* determines the best format in which to give a user access to that information, taking into account the available media, and display and interaction technologies. *UI realization* (or *content realization*) finalizes concrete presentations in each of the media employed. All these techniques must be applied dynamically, since the user's tasks, the tracking accuracy, and the relative location of the user to the surrounding physical environment may change frequently.

Figure 5.23: (a) Outdoor AR UI with simple label placement, leading to clutter and misplaced labels. (b) View management ensures correct placement of labels (here simulated on environment model for a view similar to (a).

5.4.3.2 View Management for AR

In this subsection, we look at the view management problem in a little bit more detail, since quite a few of the MARS UI problems we identified in the beginning of Section 5.4 are related to the correct layout of the overlaid material.

If the annotations are not optimized for the user's given view, the resulting overlays might be hard to parse and even misleading. This happens when augmented material is positioned awkwardly and ambiguously in the user's view. For example, labels and annotations might overlap each other, making them hard to decipher and unclear as to which of several physical objects they annotate. Figure 5.23(a) provides an example of bad annotation placement: The system places several building labels on top of each other, and others in a way such that they annotate the wrong building. This happens because the labeling algorithm used for this figure simply places building labels at the screen positions to which the centers of the real-world buildings get projected. Since the centers of multiple buildings project quite close to each other and the algorithm does not take into account visible surface determination, it is not clear which label refers to what physical building. The labels for "Kent" and "Philosophy" at the right hand side, for example, appear to annotate the wrong buildings. Label color and other visual attributes can be utilized to denote distance from the user and to emphasize the fact that some buildings are hidden by others (Kamada and Kawai, 1987), but as long as there are visible portions of a certain building, label placement alone can correctly identify the annotation relationship.

In joint work with Blaine Bell and Steven Feiner, we explored the notion of *view*

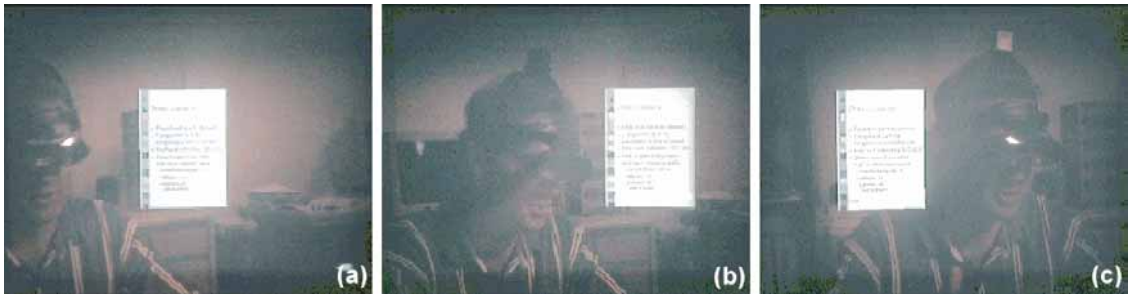


Figure 5.24: View management (imaged through see-through head-worn display). (a) Head-tracked colleague's head is constrained to be visible to head-tracked observer. (b–c) Therefore, virtual agenda automatically moves to avoid obstructing colleague's head as observer and colleague move.

management for AR UIs (Bell et al., 2001). View management tries to ensure that annotations accurately refer to the visible parts of the infrastructure as seen from the current viewpoint. In Figure 5.23(b) we annotate a virtual view of our campus model that was generated to approximately match the one in 5.23(a). Here, the labels correctly take into account what parts of buildings are obstructed by other parts, depending on the user's location and view. Annotations are placed within the biggest rectangle covering the non-occluded portions of the corresponding building's projection to the view plane, if there is enough space for a label given minimum font size. If there is not enough space, but parts of the building are still visible, the label is placed on the outside pointing in to the biggest rectangle covering the visible building parts.

View management makes sure that annotations do not accidentally occlude each other or other important objects of which the user should be guaranteed a clear view. Figure 5.24 illustrates a simple example of a "protected" object that should not be occluded by virtual material. The example application provides support for augmented collaborative meetings (cf. Section 5.2.2). The three images show one meeting participant's view of her colleague, as seen through a see-through head-worn display. Both participants' heads are position- and orientation-tracked and a distributed AR environment provides personalized views of shared 3D graphics models that are discussed during the meeting.

In Figure 5.24 (a), the observer, whose view is shown, has just brought up a screen-stabilized virtual meeting agenda, which is constrained to be visible to the observer and to be positioned as close as possible to the center of the observer's display. Her colleague's head is constrained to be visible to the observer, as long as it remains within her view frustum. Figure 5.24 (b–c) shows how the agenda automatically moves out of the way to avoid obscuring the colleague's head when either the observer or colleague move. In part (c), it has moved to the other side of the observer's head. For a short transition period during this move, one of the visibility constraints had to be relaxed.

We experiment with resolving such temporary conflicts by exploiting flexibilities in the way virtual objects are displayed. Possible solutions include moving the flexible object around the protected object swiftly and smoothly while shrinking it in size, or making the object semi-transparent while it smoothly crosses the protected object. The decision of exactly what to do can be based on the type and properties of the objects involved. While we hardcoded the behavior in this particular early example, our rule-based architecture (see Chapter 6) provides the necessary flexibility to make these decisions dynamically.

A simple two-element example, such as the one in Figure 5.24, is easy to implement, since the system only has to attend to a single protected area. The geometric processing in this case involves only simple comparisons of one upright rectangle representing the agenda's projection on the view plane with upright rectangular extents representing the colleague's head's projection and the viewable area of the head-worn display. 2D UIs, such as Microsoft Word, already position find/replace dialogue boxes in a similar fashion, such that they do not block the text segments to which they refer. View management becomes significantly more difficult, however, if multiple objects, with different types of constraints, are to be considered. If handled naively, satisfying one constraint by moving an object out of the way of another object, is likely to violate other constraints of nearby or associated objects.

To fulfill all requirements posed by the visibility constraints, and to do so in real time, the view management module requires a good representation of the occupied and unoccupied portions of a user's view, which must be updated every rendering frame. We currently make layout decisions for view management in the 2D space of the user's projection plane, based on rectangular approximations of the objects' projections (Bell et al., 2001). This approach leverages the efficient 2D space-management techniques of Bell and Feiner (2000), making it possible for the view-management algorithm to perform at interactive speed.

Our methods to deconflict annotations are especially useful to correctly label small objects that are located close to each other, as for example in an overview visualization of a scaled-down environment, such as the miniature campus model of Figure 5.25. The view management module manages all annotations in this scene in real time. The application is a meeting situation like the one described above. Here, the participants are meeting to discuss the design of our campus model. Building labels are laid out dynamically for each participant so that each label overlaps only its own building as seen from that person's view. Labels change size and style depending upon the amount of space available. In this case, the user selected the model of *Buell Hall* to inspect, causing a copy of the building to be made, and information about it to appear in an attached document that is constrained to stay close to the building copy. Like the agenda in the top left corner, the building copy and document avoid overlapping other objects determined

Figure 5.25: View management in a collaborative system (imaged through see-through head-worn display). Labels are laid out dynamically to annotate the buildings of a campus model as seen by the observer. UI elements avoid overlapping the colleague's head and the campus model.

to be more important (e.g. the campus buildings and the colleague's head).

View management depends on a considerable number of factors that need to be taken into account to determine the best possible UI layout. As we have seen, objects may need to be moved out of the way of other objects deemed more important. Annotations may be attached to points, abstract areas, or object silhouettes in screen, world, or body coordinate systems, which constrains the extents by which they can be moved around the AR environment. Several size and visibility constraints may be placed on certain objects. For example, textual annotations need to maintain a certain size, viewing angle, and distance to other objects to stay readable. In order to control all the properties involved in managing a large set of UI objects, we need to take an automated approach. In the next chapter we introduce a rule-based MARS that can store information about the UI components in the form of facts in a knowledge base.

Chapter 6

Rule-based Architecture for MARS UI Management

In this chapter we present Ruby, our rule-based architecture for adaptive MARS interfaces and UI management.

Looking at the increasingly complex and dynamic MARS UI examples discussed in the previous chapter, it becomes apparent that the mobile augmented reality domain is much more complicated than the one of common desktop UIs: Interfaces have to consider both virtual and physical objects and potentially a multitude of devices (input and output). Screen composition is much more dynamic: The UI needs to change based on the user's position and head pose and should ideally take into account all kinds of situational cues.

As a result of these complications, visuals become easily cluttered and overwhelming. Labels and interface components could overlap each other due to unforeseen dynamic movements and instead of pointing out or emphasizing important physical objects, the system might inadvertently obscure them. Since the augmented world *becomes* the UI, the space that has to be somehow managed by the user is the whole environment that surrounds her.

We believe that direct-manipulation approaches, which are sometimes even barely adequate for 2D desktop window management, when a certain number of open applications is exceeded, will not scale up to handle 3D MARS environment management well. When applied to a large, dynamic, shared environment with more than one display, direct-manipulation approaches could instead overwhelm users by constantly presenting them with situations that require tedious low-level decisions about the position, size, and properties of virtual objects. Our work explores approaches, in which knowledge-based UI-management tools attempt to keep mobile AR UIs usable in a variety of user situations (cf. Section 5.4.3). In order for the MARS to adapt its UI to different situations, it

needs to have a formal representation of the UI components. Our proposed taxonomy of Chapter 4 serves as the starting point for such knowledge-based representations.

The goal is for the MARS UI to be able to reason about these components and, if necessary, rearrange the overall UI layout dynamically. Our design shares goals with other architectures that were developed to ease the construction of complex heterogeneous 3D graphics and interaction environments (Appino et al., 1992; Codella et al., 1993; Benford and Fahlén, 1993; Carlsson and Hagsand, 1993), but unlike most previous work, has a clear focus on adaptive techniques for the UI layout. Much work has been done on the automated design of information presentations, both for static designs and for dynamic multimedia documents (Feiner, 1985; Mackinlay, 1986; Roth and Mattis, 1990; Seligmann and Feiner, 1991; André et al., 1993; Seligmann, 1993; Roth et al., 1994; Zhou and Feiner, 1997; Lok and Feiner, 2001). Setting our focus on interactive interface adaptation, rather than completely automated presentation design, we present a rule-based architecture that allows a wearable computing platform to adapt its AR UI to infrastructure constraints and user context.

The rest of this chapter is structured as follows: Section 6.1 describes our rule-based system architecture Ruby, explaining in detail our formalism for representing MARS components and their properties and relationships, covering Ruby's reliance on Jess, a forward-chaining expert system shell implemented in Java, and the way control flow works in this system of facts, rules, callback objects, and object-oriented event management. Section 6.2 presents first implementation examples exploiting this flexible infrastructure. We conclude this chapter with a discussion of the presented approach in Section 6.3.

6.1 Ruby: a Rule-Based System Architecture

The three UI-management techniques from Section 5.4.3 require a great deal of flexibility on the MARS's part, as well as detailed knowledge about the properties and purposes of MARS UI components. We categorized our experiences with MARS UIs in a taxonomy of UI components (Chapter 4). In this section we show how to formalize the objects and properties from our taxonomy as object-oriented data structures and associated *unordered facts* and their *slots* in a knowledge-base using a forward-chaining expert system formalism.

The main motivation for using a rule-based infrastructure is our belief that such an architecture can more easily handle the complexity of many interacting events than other programming models can. As described above, flexibility is a very important factor in MARS UIs. The user's potential view onto a scene is constrained only by physical limitations and cannot be predicted by the UI designer. Annotations have to be correctly placed

for all possible viewing directions. Also, many different events can happen concurrently. An object that is currently under examination by the user can become occluded because of a change of viewing angle or because of other objects moving in front of it. Likewise, the user might temporarily look away from it. Tracking information can become inaccurate, for example because of loss of line of sight to GPS satellites. The user might start walking, causing involuntary head motion rendering the current interface unusable. The user might leave a particular region that the computer has data on, causing the need for requesting data about a new area via the network. A colleague might page the user with an important message. The system itself might have to issue a warning message, such as for example a low battery warning. These are just a few example events that can happen concurrently. Note that these events are by no means independent of each other. For example, turning momentarily away from an annotated object of focus should cause considerably different changes in a UI in which world-stabilized overlays are possible, than in a predominantly screen-stabilized UI that might have been brought about by sudden position-tracking inaccuracies (with orientation tracking still intact). In the world-stabilized case, the annotation might stay on the screen, with a leader line providing the possibility to get back to it (cf. Section 5.1.1). In the screen-stabilized UI, a text message might alert the user if the annotated object is still straight ahead in his or her field of view, or prompt the user to look left or right to find back to it (cf. Section 6.2.2).

The decision on how a UI should adapt to certain situations cannot be based on events alone. The purpose of a UI and which particular UI elements are employed when a change becomes necessary are also important factors. Going back to the example of losing accurate position tracking, assume that the user has started a navigational guidance task. The target object is highlighted by a direct world overlay, and a virtual path points out the shortest route to get there (cf. Section 5.2.1). When losing accurate position tracking, the system could simply display screen stabilized information about the target being left, right, or straight ahead. In order to adequately convey the path information, however, the computer might decide to present a WiM (cf. Section 5.1.1), and highlight the user's rough location and current viewing angle, the target object, and the landmark objects in it.

Considering these adaptive UI examples, it becomes clear that it would be extraordinarily tedious and error prone to prepare a contingency plan for any given combination of UI elements that might be part of the interface when a certain event occurs. Likewise, the combination of different events is difficult to predict. Coding a complete finite state machine that advises the computer as to exactly what to do in case of what event in presence of what other interface elements is prohibitively difficult. In the following we describe an architecture that we feel is better prepared to handle such dynamic UI design decisions than currently existing MARSs.

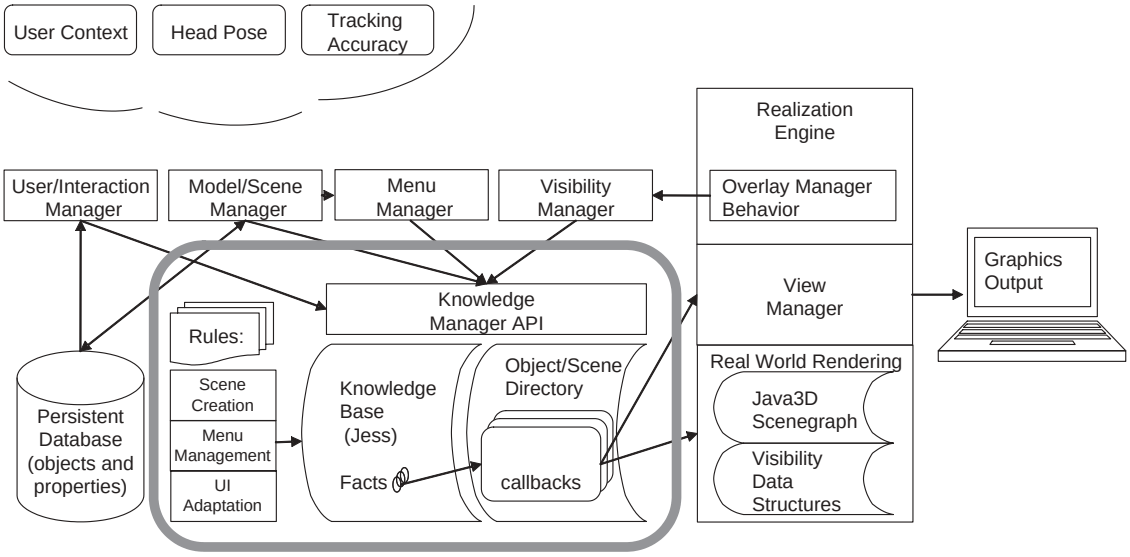


Figure 6.1: Ruby Software Architecture. The grey rectangle highlights the rule-based core of the system including the Jess-based knowledge base, the rule and template libraries, and the interfaces to the object-oriented rest of the system.

6.1.1 Ruby Architecture

The Ruby architecture is based on a radically different control and event model than our previous MARS implementations, which are described in Sections 3.2.1 to 3.2.4.

At the heart of the Ruby architecture lies a set of control modules and data structures forming a knowledge-based core unit. This unit is depicted in Figure 6.1, highlighted by the grey rounded rectangle. A knowledge base, which is implemented using the rule-based formalism Jess (Friedman-Hill, 1998), is controlled by several sets of rules that are authored by the system designer. The formalism allows for these rules to be modified at runtime. The knowledge base keeps track of the system state at any given point of time using a variable set of "facts" (cf. Section 6.1.2), which encode knowledge about the current scene, the objects in and outside the user's field of view, the state of the user, some tracking information, the input and output devices currently accessible, and the different UI components that are used to form the virtual UI layer (annotations, menus, buttons, compound objects such as a WiM, etc.) at that point. Whenever something happens to the system state, for example because of user motion or interaction, new facts get inserted into the knowledge base and/or existing facts get modified or removed. Some of the facts represent MARS objects (cf. Section 4.2.1), such as physical objects and virtual interface elements. These facts have corresponding Java objects in the object/scene directory (cf. Figure 6.1). When the facts are modified, the corresponding Java objects are automatically updated and callbacks are executed, causing an immediate change in

how the UI is represented.

The UI realization engine is depicted to the right of the knowledge base core in Figure 6.1. It is responsible for forming the virtual layer of the AR UI, which is sent to the appropriate displays (symbolized by the graphics output icon at the right of the figure) with an update rate of about 15 to 60 frames per second on the MARS 2002 hardware (cf. Appendix A.4), depending on scene complexity. Data is shared between the knowledge-based system core and the realization engine using a convenient mechanism provided by Jess: `Definstance` facts. These are Java object instances that are at the same time represented as facts in the knowledge base. The very same objects representing UI components in the virtual UI layer as part of the Java/Java3D data structures in the realization engine are represented in the object/scene directory, and are controlled by facts in the knowledge base.

Apart from the Java3D scene graph, the rendering module maintains its own visibility data structures to compute occlusion relationships among objects in the environment. The view manager, depicted right above the rendering engine in Figure 6.1, is in charge of resolving annotations such that they correctly annotate the visible portions of physical objects and do not overlap each other, using the approaches described in Section 5.4.3.2. The overlay manager controls all involved data structures and algorithms on a frame by frame basis. Implemented as a Java3D behavior that gets executed every time a new 3D graphics frame is ready to be sent to the output device, it updates the visibility data structures, feeds occlusion information back to the visibility manager and thus to the knowledge base, and triggers the view-management computations.

The interaction between the user, the sensors, and the knowledge base as the main data structure for system state happens through a well defined knowledge manager API that provides methods for accessing and modifying the knowledge base. Several manager modules process information from the sensors, the user, or from the realization engine. They are briefly presented in the following paragraphs.

The user/interaction manager makes use of a simple user model that is stored in the persistent database and controlled by a user and device configuration file that is read at system startup time. It also processes input from the interaction devices that are part of the current configuration. The model/scene manager sets up and updates the knowledge base to reflect the environment the user is in at any given time. Prompted by tracking information and system requests, it queries the persistent database for information about relevant objects in the user's environment and populates the knowledge base with representations of them. It keeps track of scene boundaries, and requests new information from the connected databases whenever the user enters a sector on which there is currently no information in the knowledge base.

The menu manager is a special purpose module controlling a interaction menu

hierarchy, which a user can bring up at any given time while interacting with the system. The menu system was designed and implemented by Blaine Bell. At the current time, a global menu hierarchy gives access to many user-controllable features, such as what object types in the scene will be labeled, or how the environment should be displayed as a World-in-Miniature, if desired. Also, necessary setup functionality such as tracker calibration is triggered through menu entries. Debug information can also be controlled through menu entries. Menus are represented by circular arangements of menu items, popping up at the bottom center of the screen, and are navigated by a wireless mouse through interaction that was inspired by pie menus (Hopkins, 1987). The menu hierarchy that is accessible at any given point in time can change based on the environment the user is in and on the general application parameters that are either set forth in configuration files at startup time, or determined through previous user selections. The menu manager module ensures that the correct menu items are accessible and updates the knowledge base about the current menu state, based on scene constraints and user preferences. The menu system as implemented in the current Ruby version serves as the main input mechanism for many different and often unrelated concepts. Eventually, menu interaction should be even more flexible, using different kind of menus, possibly expressed in different media, with the knowledge base determining which type to use in a given situation.

We already briefly mentioned the visibility manager above. It updates the knowledge base on a frame by frame basis with information about occlusion and the objects entering and leaving the current field of view. Note that, even though the diagram gives the impression that the only feedback path from the realization engine to the knowledge base is through the visibility manager, feedback is very easily given more directly by updating member variables of the UI objects that are represented in the object/scene directory. Updating those objects modifies the state of the associated definstance facts in the knowledge base immediately (see Jess details in Section 6.1.2).

In summary, changes in the system state as represented by the knowledge base can come about by the knowledge manager inserting, changing, or deleting specific knowledge base facts, or by any part of the program making changes to those Java object instances that are also represented as facts in the knowledge base (definstance facts), or by the rules reacting to previously changed system state by firing and producing more changes. In this way the system behaves exactly like a forward chaining expert system (Buchanan et al., 1969; Buchanan and Shortliffe, 1985) that efficiently keeps the system state up to date according to a dynamic set of behavioral rules. The rules are stored in modules which are populated from rule files at system startup.

The knowledge base itself is populated with facts at system startup time, initially triggered by information from persistent databases the MARS has access to. Several

configuration files control the user model and application parameters that are to be used in the session.

The next subsection will give some more details about how Jess works and how it is integrated into Ruby.

6.1.2 Ruby and Jess

Jess (Friedman-Hill, 1998) is an expert system shell, or, in other words, a rule engine and scripting language, written entirely in Java. Jess supports the development of rule-based expert systems which can be tightly coupled to code written in Java. Jess was originally inspired by the CLIPS expert system shell (Culbert et al., 1987), which was implemented in C. Starting out as a Java-based implementation of CLIPS, Jess has grown into a distinct Java-influenced rule-based environment of its own. The Jess shell language syntax is very similar to CLIPS, which in turn is a highly specialized form of LISP.

Using Jess, knowledge about the MARS system state is encoded as facts stored in the knowledge base. Facts are logical statements, such as "the AR display has a resolution of 800x600 pixels". Jess provides support for three different types of facts: ordered facts, unordered facts, and definstance facts. *Ordered facts* are simply lists in which the first field acts as a sort of category for the fact. The expression

```
(ARDisplayResolution 800 600)
```

would adequately encode the above statement using an ordered fact. *Unordered facts* offer more structure by providing *slots* which are similar to *fields* or *member variables* in object oriented programming languages. The expression

```
(ARDisplay (pixelWidth 800) (pixelHeight 600))
```

conveys the above information using an unordered fact, provided that we previously defined a template for a fact `ARDisplay` with two slots, named `pixelWidth` and `pixelHeight`, both expecting an Integer value.

A convenient feature of Jess that makes it particularly suitable for controlling UI design in Ruby is its close integration with Java. Jess allows any Java object instantiations to be included as *definstance facts* in the knowledge base, as long as simple naming conventions for some of the public methods are adhered to, following the model of Java Beans (Giguère, 1997). The "slots" in these facts correspond to member variables (or properties as they are called in the case of Java Beans) in the corresponding Java objects. When slots in a fact are changed, the member variables of the corresponding Java objects are updated and vice versa. This extends the power of the knowledge base to the Java world. Fact and object are automatically kept in sync. This makes it possible to conveniently represent concepts from our taxonomy of MARS entities as Java objects, and at

the same time as Jess facts that can be governed by rules in the knowledge base. To go back to the above example, a simple Java Bean that represents an AR display could look something like this:

```
public class JS_ARDisplay extends DynamicJessBean
{
  protected int m_pixelWidth = 800,
                m_pixelHeight = 600;

  public int  getPixelWidth() { return m_pixelWidth; }
  public void setPixelWidth(int x) { m_pixelWidth = x; }
  public int  getPixelHeight() { return m_pixelHeight; }
  public void setPixelHeight(int y) { m_pixelHeight = y; }
  ...
  <additional variables and access methods implementing
   display functionality>
  ...
}
```

`DynamicJessBean` is a class that supports the appropriate `PropertyChange-Listeners`. This ensures that the knowledge base will be updated every time a property of the Bean changes (for details see (Friedman-Hill, 1998)). The above Java Bean maps into the following definstance fact when it is inserted into the knowledge base:

```
(ARDisplay (class <External-Address:java.lang.Class>)
(pixelWidth 800) (pixelHeight 600) (OBJECT <External-Address:
 edu.columbia.cs.cgui.jess.JS_ARDisplay>))
```

Every access method starting with “get” maps into a new slot for the associated definstance fact. The slot `class` comes from the method `getClass()` that every object inherits from `java.lang.object`. The slot `OBJECT` is added by Jess; its value is a reference to the Bean itself, which allows public object methods to be called from within Jess rules. A more complicated example of how a class-hierarchy maps into definstance facts supporting inheritance is presented in Figures 6.3 and 6.5 and described in Section 6.1.3.

Ruby uses both definstance and plain unordered facts. The former are used for any concepts that require, or use for convenience reasons, an actual Java object implementation. The latter are used for expressing goals, notifier objects, and simple concepts that do not need the Java backend.

Rules consist of left-hand side patterns and right-hand side actions. Whenever there exist facts in the knowledge base such that all left-hand side patterns of a rule are

matched, the rule fires and the right-hand side actions are performed. For example, assume we have definstance facts `ARDisplay` (as introduced above), and `ImageObject`, which represents a bitmapped image. Then the following rule causes the image to be shown on the ARDisplay only if its width and height do not exceed the display resolution.

```
(defrule display-image
   ;; if there exists a goal fact to convey a specific image:
   ?fact <- (conveyImage (objectID ?id))
   ;; ... and there is an image object with the correct
   ;;     id and a certain width and height:
   (ImageObject (objectID ?id) (width ?x) (height ?y)
                (OBJECT ?iobj))
   ;; ... and the image fits onto the ARDisplay:
   (ARDisplay (pixelWidth  ?w&:(> ?w ?x))
              (pixelHeight ?h&:(> ?h ?y))
              (OBJECT ?dobj))
   ;; THEN ...
  =>
   ;; display the image, calling an ARDisplay method:
   (call ?dobj displayImage ?iobj))
   ;; and retract the goal, because it's accomplished:
   (retract ?fact)
}
```

Note that on successful rule execution the "goal" fact `conveyImage` gets retracted from the knowledge base. To understand exactly how patterns are matched in this and other examples, see (Friedman-Hill, 1998). To maintain a consistent knowledge base, Jess makes use of an improved version of the Rete algorithm (Forgy, 1982) that efficiently implements forward-chaining expert system calculations.

Three other Jess concepts that are used in Ruby are templates, queries, and functions. All unstructured facts in the knowledge base need a definition of the slots that they use and maintain. In case of definstance facts, the corresponding Java object provides this information. All other facts in the knowledge base need to be defined by a `deftemplate` construct. The `defquery` construct allows a programmer to create a special kind of rule with no right-hand-side. While rules act spontaneously, queries are used to search the knowledge base under direct program control. Whereas a rule is activated once for each matching set of facts, a query provides the programmer with an iterator construct to cycle through all the matches. Functions in Jess work exactly like

in any other programming language. They consist of groups of actions and calls to other functions and can be called from any right-hand side of a rule, or from Java code.

The Jess package can be used as a library, rule engine, or system shell. It is quite flexible with regard to the role it plays in developing a new application. The options range from writing programs completely in the Jess scripting language, to an application completely written in Java, which manipulates Jess entirely through its Java API. On this spectrum, Ruby is placed closer towards the Java-centric pole than the Jess pole, but it does allows the loading of Jess language scripts at runtime and also offers optional control and debugging via an interactive Jess shell. As shown in Figure 6.1, the Jess rule engine is used as a central component in a complex Java-based architecture. We are employing Jess as a rule-based control engine for a real-time graphical application. Ruby's realization engine is based on Java3D and implements its own tight event loop. The coupling with Jess works as well as it does, because arbitrary Java objects can be represented as facts in the knowledge base and the rule-based computations involving these objects occur decoupled from the rendering and UI interactions, but happen frequently enough to facilitate interactive response times.

One of the most important system design challenges for rule-based systems is to use the knowledge base only for what it does well: maintaining system state represented by symbolic information, governed by complex rules that can fire in any arbitrary order and combination. Our extensive experiments with this hybrid infrastructure confirm the traditional wisdom that concrete numeric computations are not well suited for rule-based control. As a consequence, we leave detailed computations such as exact placement of labels to the real-time layout algorithms encoded in the realization engine, but let the rule base decide the respective priorities and neighbor relationships that serve as important parameters to the layout.

In the following subsection, we look at how information about the MARS UI is encoded, and how the rules govern the system state.

6.1.3 Ruby Data Formats

In Section 4.2, we have presented a general taxonomy of MARS UI components. Ruby's data formats for describing the MARS UI state are based on that taxonomy. While only a certain subset of the concepts is implemented, the overall structure of Ruby's data representations mirrors the ideas from Chapter 4. In the following sections, we take a closer look at Ruby objects, their internal and external attributes, as well as relationships, events, and goals.

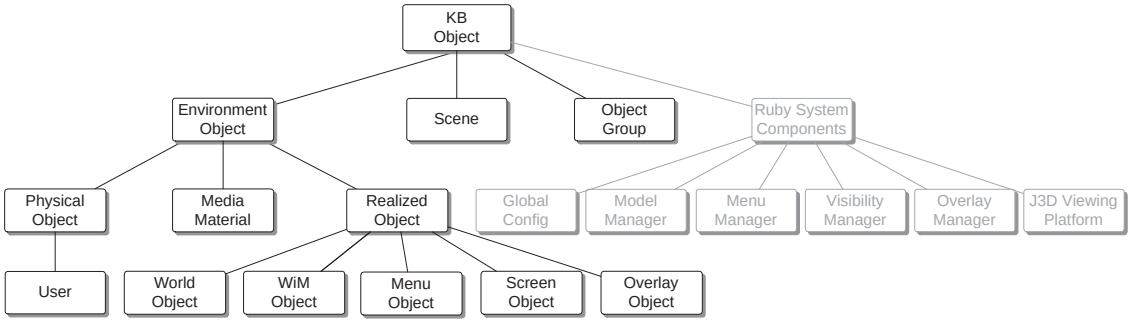


Figure 6.2: Ruby definstance object hierarchy.

6.1.3.1 Ruby Objects

Jess does not by itself enforce object-oriented data representations. Jess's knowledge base facts can represent objects and attributes, goals, and notification flags alike. Rules often make use of small volatile facts that help with the bookkeeping of information and that have no corresponding entity in the Java part of the architecture. However, the concept of a definstance fact, as described in the previous section, establishes an object hierarchy, and the structure it imposes proves very helpful for systematic rule development. In Ruby, we follow the convention that facts representing objects start with a capital letter, whereas the heads of attributes, goals, and notification flags are lower case.

Figure 6.2 gives an overview of the Java-accessible part of Ruby's knowledge base object hierarchy. The figure shows all classes that are implemented as Java objects and definstance facts. The lighter shaded objects in the right half of the figure represent Ruby system components, for which the knowledge base can directly control one or several parameters. The left side shows the hierarchy of Ruby UI components. It is informative to compare the left hand side of Figure 6.2 with Figure 4.2, which depicts the theoretical hierarchy of MARS objects we presented as part of the UI taxonomy of Chapter 4. The differences can be summarized as follows: The container object concept was simplified to a general concept of object groups; the special purpose object "scene" is represented as its own class, independent from "physical object"; and the concept of UI templates is missing from the Ruby object hierarchy. Instead, several specific types of realized objects are represented as subclasses of "realized object".

The missing UI template concept is the biggest difference between the object hierarchy in the taxonomy and the Ruby system. The idea of UI templates is to provide blueprints for types of realized objects. In a generalized fashion, UI templates would describe the overall structure of a virtual UI object without specifying the particular content and realization parameters (cf. Section 4.2.1.2). In other words, they describe

certain subclasses of realized objects – types of virtual UI elements for which the MARS can fill in the content and parameters. Such a template mechanism has not yet been implemented in such generality in the Ruby system. As a first step in that direction, we decided to implement important "UI templates" simply as subclasses of "realized object". We will elaborate on the subclasses we have currently implemented in the next paragraphs. This approach is less general than creating a Ruby concept for UI templates, because these subclasses and their parameters have to be coded in Java first and then added to the knowledge base as definstance facts. With the appropriate UI template concept they could be expressed in the Jess scripting language, which would mean that new UI templates could be provided without recompiling the whole system. This would require a more general formalization of which UI components can be part of what type of templates. We would also have to recreate in some other way the automatic mapping between facts and Java objects for these template objects, since the templates would not be definstance facts anymore. Implementing the general notion of UI templates would be easier if we built our own UI-specific rule-based language on top of Jess, into which we could built abstractions for these concepts. This is left to future work. However, as long as there are only a few UI templates to be considered for rule-based control, the current approach has no serious drawbacks.

As shown in Figure 6.2, we have currently implemented five subclasses of realized objects: "world object", "WiM object", "menu object", "screen object", and "overlay object". Note that these are not the only types of realized objects supported in Ruby. In particular, the overlay object category comprises different types of objects, such as labels and various kinds of pop-up annotations, but the parameters that are controlled by the knowledge base are the same for all these objects, alleviating the need for more specific subclasses of "overlay object" at this point. Also, there are some UI objects that are currently not under any control of the knowledge base, but implemented in Java code alone, such as debug messages and some legacy interface mechanisms from the JaBAR infrastructure, such as 2D to 3D drag and drop (cf. Section 5.1.2).

World objects are virtual 3D objects placed in the 3D world around the user, independent from where the user is looking. In particular, this includes not only virtual models of physical objects that are collocated with their physical counterparts (cf. Figure 6.8d), but also purely virtual objects, such as the flags from the Situated Documentaries UI (cf. Section 5.1). *WiM objects* are any 3D objects placed within the reference frame of a WiM tool (cf. Section 5.3.2.1). *Menu objects* are interface elements implementing the circular menu hierarchies that the menu manager controls, which is mentioned above in Section 6.1.1. *Screen objects* are objects that are placed in absolute 2D screen coordinates, such as the navigational messages and arrows described in Section 6.2.2. Finally, *overlay objects* are annotations of world locations or objects. They are either placed in

the 3D world for stereoscopic viewing, or on the 2D AR image plane (screen), but they always refer to an object or point in the world around the user. If they are screen-based, they shift position to stay linked with the item they annotate, or at least they maintain a link, such as a leader line, to that item. The layout of both screen objects and overlay objects is controlled by the view manager mentioned in the architecture description above (Section 6.1.1) and the techniques and placement algorithms referenced in Section 5.4.3.2.

As Figures 6.2 and 6.3 illustrate, knowledge base objects make use of inheritance. The *User* object, for example, is a specific *physical object*. All physical objects are *environment objects*, and all environment objects are *knowledge base objects*. Figure 6.3 shows how `JS_PhysicalObjects` inherits from `JS_EnvObject`, which in turn is a subclass of `JS_KBObject`. Each of these classes provides ”get” and ”set” access functions for the member variables that are represented as slots in the corresponding definstance facts. Figure 6.4 shows the knowledge base representation of a PhysicalObject. At the top of Figure 6.5 we see a definstance fact for a different physical object, listed in the format that the interactive Jess shell utilizes. All the slots from `JS_PhysicalObject` and its two ancestor classes are represented. Apart from the access functions, each definstance class can provide additional public member functions, which may be called from rules in the knowledge base. An example is `JS_PhysicalObject`'s method `scene()` that returns a pointer to an object describing the current user environment. Section 6.1.4 provides an example of a rule that retrieves the scene object using this method (cf. Figure 6.7).

This subsection described how MARS objects are represented in Ruby. The attributes of these objects can be represented as slots in the facts, which for definstance facts correspond to member variables in the respective Java Beans. We call these attributes *internal*. Other attributes are encoded as standalone facts in the knowledge base, linked to the original object through a unique identifier and/or the object reference. These are called *external* attributes. The use of external attributes can speed up the rule evaluation process. We will discuss them in Section 6.1.3.3.

6.1.3.2 Internal Attributes

Every definstance object has at least two internal attributes. The object reference `OBJECT` provides access to the corresponding Java object and its public methods. The `class` slot, inherited from `java.lang.object`, makes it possible to access Java's reflection API methods. Figure 6.4 lists all internal attributes of a physical object fact, grouped manually by functionality.

The first group of internal attributes consists of identification and type properties. We already mentioned `OBJECT` and `class`. `id` is a unique knowledge base identifier.

```
public class JS_KBObject extends
JessBeanFactBroadcaster implements
FactListener,Serializable {

    public int getId();
    public void setId (int id);

    //======

    public boolean equals(Object obj);
    public int hashCode();
    public void factAsserted(Object fact);
    public void factRetracted(Object fact);

    public JS_KBObject(JS_KBObject kbo);
    public JS_KBObject();
}
```

```
public class JS_EnvObject extends JS_KBObject  {

    public String getName();
    public void setName(String name);

    public int getSceneID();
    public void setSceneID (int id);

    public int getGroupID();
    public void setGroupID(int groupID);

    public boolean getInUse();
    public void setInUse(boolean inUse);

    public boolean getVisible();
    public void setVisible(boolean visible);

    public boolean getInViewFrustum();
    public void setInViewFrustum(boolean ivf);

    public boolean getKeepVisible();
    public void setKeepVisible(boolean kv);

    public boolean getCollision();
    public void setCollision(boolean collision);

    public JS_EnvObject getCollider();
    public void setCollider(JS_EnvObject coll);

    public boolean getOccluded();
    public void setOccluded(boolean occluded);

    public boolean getOccluding();
    public void setOccluding(boolean occluding);

    //======

    public void setVisFact(Fact f);
    public void setViewFrustFact(Fact f);
    public void factAsserted(Object fact);
    public void factRetracted(Object fact);

    public JS_EnvObject(JS_EnvObject envo);
    public JS_EnvObject();
}
```

```
public class JS_PhysicalObject extends JS_EnvObject implements
VisibleSpace.VisibleSpaceListener  {

    public int getDbID();
    public void setDbID(int dbid);

    public int getRoomID();
    public void setRoomID(int room);

    public String getObjType();
    public void setObjType(String objType);

    public String getOriTracking();
    public void setOriTracking(String otr);

    public String getPosTracking();
    public void setPosTracking(String ptr);

    public boolean getPositionStatic();
    public void setPositionStatic(boolean ps);

    public boolean getPointedOut();
    public void setPointedOut(boolean pointedOut);

    public boolean getNeedToCompute();
    public void setNeedToCompute(boolean ntc);

    public boolean getIsCalibPoint();
    public void setIsCalibPoint(boolean icp);

    public boolean getVisibleInWIM();
    public void setVisibleInWIM(boolean viw);

    public boolean getCannotBeVisible();
    public void setCannotBeVisible(boolean cbv);

    public boolean getDoNotProjectInWorld();
    public void setDoNotProjectInWorld(boolean dnpiw);

    public boolean getHasBSPObject();
    public BSPObject BSPObject();

    public boolean getHasInfoPanel();
    public InfoPanel mainInfoPanel();
    public void setMainInfoPanel(InfoPanel infoPanel);

    //======

    public void setScene(JS_Scene scene);
    public JS_Scene scene();
    public JS_OverlayObject createInfoPanelPopup(boolean b);

    public void setPointedOutFact(Fact f);
    public void setNeedToComputeFact(Fact f);
    public void setVisibleInWIM(Fact f);

    public void factAsserted(Object fact);
    public void factRetracted(Object fact);

    public Fact labelObject();
    public void setLabelObject(Fact fact);
    public JS_OverlayObject createLabelObject(boolean b);

    public JS_PhysicalObject(int objectID, String name,
                             int groupID, BSPObject bspo);
    public JS_PhysicalObject(JS_PhysicalObject sco);
    public JS_PhysicalObject();
}
```

Figure 6.3: Public methods of class PhysicalObject and relevant ancestor classes.

```
(MAIN::PhysicalObject
  (class <External-Address:java.lang.Class>)
  (OBJECT <External-Address:edu.columbia.cs.cgui.jess.JS_PhysicalObject>)

  (id 89) (dbID 113) (sceneID 1) (roomID 1) (groupID 1)
  (objType "OBJECT") (name "Wooden Table 1")

  (inUse TRUE) (visible FALSE) (visibleInWIM FALSE) (inViewFrustum FALSE)
  (occluded FALSE) (occluding FALSE)

  (positionStatic TRUE) (oriTracking "NONE") (posTracking "NONE")

  (collider nil) (collision FALSE)

  (keepVisible FALSE) (cannotBeVisible FALSE) (doNotProjectInWorld FALSE)

  (hasBSPObject TRUE) (isCalibPoint FALSE) (hasInfoPanel FALSE)

  (needToCompute TRUE) (pointedOut FALSE)
)
```

Figure 6.4: Internal attributes of PhysicalObject, grouped by functionality.

Physical objects are stored in persistent databases and read in at runtime. Therefore, the next items are a database identifier, `dbID`, and identifiers for the scene and room that the physical object can be found in (`sceneID` and `roomID`). Scene and room are geographical grouping concepts. A scene defines a logical area, for which all data can be kept on the MARS. When the user crosses a scene boundary, new information is read in from the database in order to represent the new scene in the knowledge base. A room is a smaller grouping unit that represents compartments within a scene. For example, Columbia's campus may be represented as a scene, and a specific building as a room. The room concept is hierarchical, so that a laboratory within a building can again be a room. The attribute `groupID` enables semantic grouping of objects across different rooms or even scenes. One group for example brings together all buildings, another all furniture objects, yet another all computers, and so on.

Unlike the `groupID` attribute, which originates with `JS_EnvObject`, which is the environment object of Figure 6.2, the `objectType` attribute applies only to physical objects. It distinguishes special purpose objects such as walls, floors, grass patches, and walkways from other objects, which are listed with a default type, `OBJECT`. Each environment object also has a `name` attribute, which is important for general knowledge base maintainance, in particular understanding the effects of rules on specific objects. It is also used as the default text to display when a label is created. If there is no name stored with an object in the persistent database, a unique name is constructed from the object type and a number.

The rest of the attributes listed in Figure 6.4 consists of different kind of flags. First come the state attributes: is the object currently considered part of the scene (`inUse`)? Is it visible, either in the real world (`visible`) or the world in miniature (`visibleInWIM`), if applicable? Is it currently in the view frustum, the cone shaped region in front of the user's glasses, in which annotations are possible (`inViewFrustum`)? Is it occluded by another object (`occluded`) and/or occluding any other object that is currently part of the scene (`occluding`)?

There are no attributes listing specific position and orientation quantities. This kind of quantitative information is more efficiently maintained outside of the knowledge base. Instead we use discrete flags informing the database about qualitative aspects of the object dynamics, such as a flag that monitors if the object is static or moving. Another two attributes are responsible for monitoring tracking; `oriTracking` and `posTracking` take as values `"NONE"`, `"LOW"`, and `"HIGH"`, and thus provide the knowledge base a simple discretized view of tracking performance. These attributes are used in the demo application presented in Section 6.2.3. Two attributes watch over collisions: a binary flag (`collision`) and an object reference (`collider`) that identifies the object involved in the last collision.

The next group of flags expresses constraints: A flag that requests that the object stay visible (`keepVisible`); a flag that prevents the object from ever being seen (`cannotBeVisible`); and a flag that temporarily exempts an object from being considered in view-management calculations (`doNotProjectInWorld`).

Finally, the association attributes `hasBSPObject` and `hasInfoPanel` inform the knowledge base about corresponding data structures, such as consideration in the BSP tree for visibility computations and the "InfoPanel", a potential annotation. Two notification flags used for internal bookkeeping complete the attribute set of `PhysicalObject`.

Realized objects (cf. Figure 6.2), such as for example an overlay object, use a different set of internal attributes. An example is shown towards the bottom of Figure 6.5. This particular overlay object represents a label (`isLabel TRUE`) with `labelString "Rockwell"`. A flag decides where the overlay is placed on the viewing axis: Either on the front clipping plane, or at the same distance as the object it annotates.

A group of attributes describes realization flexibilities and preferences. The label from Figure 6.5 is currently fully opaque (`transparency 0.0`), but may become transparent up to value 0.6. It is currently displayed at full size, but may change its size by 10 percent in either direction. Its text color is "regular". Similar considerations to those that made us omit concrete tracking pose information from the knowledge base, prompt us to leave concrete color quantities to the realization engine and let the rule engine operate with discrete and purely symbolic values (`"REGULAR", "HIGHLIGHTED",`

"POTENTIAL").

Every overlay object is in an annotation relationship with another environment object. The attribute `anchorID` stores the ID of that object. The annotation relationship is at the same time expressed as an external attribute fact, `isAnchoredAt`.

6.1.3.3 External Attributes, Relationships, Events, and Goals

External attributes are the other way to store object attributes in the knowledge base. They are stored as separate facts, in which the objects that they are associated with are listed by their ID and/or object reference. External attributes are easy to add programmatically. One simply adds a new fact to the knowledge base. To add a new internal attribute, the definstance object needs to be updated and the corresponding Java classes recompiled. A new external attribute on the other hand can be added even at runtime, using the interactive knowledge base shell. Other advantages of external attributes include increased fact matching and updating speed due to the smaller fact size. When an internal attribute gets changed by issuing a `modify` command in the rule base, the whole definstance fact gets retracted from the knowledge base, and a new one with the changed value gets asserted. It is faster to `assert` or `modify` a small external attribute fact than a long definstance fact. Yet another advantage is that external attributes can easily carry additional parameters. An example is the relationship attribute. In Figure 6.5 the following fact states that the virtual world object with ID 195 represents the physical object with ID 55:

```
(MAIN::represents (obj1ID 195) (obj2ID 55) (origin WORLD))
```

The `origin` slot is an additional parameter that specifies that this is a representation relationship between a world object and a physical object, as opposed, for example, a WiM object and a physical object. Note that the value `WORLD` is expressed as a symbolic atom, and not a string. Jess rules work speedily with symbolic atoms. The reason that we use strings for symbolic values in definstance objects (such as the values for `objType`, `oriTracking`, and `posTracking` in the `PhysicalObject` fact, as depicted in Figure 6.5) is that string values convert much more easily to Java than symbolic atoms.

With all these things in favor of external attributes, why do we use internal attributes at all? The answer is that they have two considerable advantages: First, they can be more easily updated from Java. While it is possible to assert and modify external attribute facts from within Java code, it is much easier to simply call the update methods on definstance objects to change their internal attributes. Second, the knowledge base appears more readable to the human eye if the most important attributes are listed right with the objects they concern, instead of being scattered around, with the only link being an

```
f-116   (MAIN::PhysicalObject (class <External-Address:java.lang.Class>) (id 55)
                              (OBJECT <External-Address:edu.columbia.cs.cgui.jess.JS_PhysicalObject>)
                              (collider nil) (collision FALSE) (groupID 5) (inUse FALSE)
                              (inViewFrustum FALSE) (keepVisible FALSE) (name "Rockwell")
                              (occluded FALSE) (occluding FALSE) (visible FALSE)
                              (cannotBeVisible FALSE) (doNotProjectInWorld FALSE)
                              (hasBSPObject TRUE) (hasInfoPanel TRUE) (isCalibPoint FALSE)
                              (needToCompute FALSE) (objType "OBJECT") (dbID 308)
                              (pointedOut FALSE) (positionStatic TRUE) (roomID 1) (sceneID 1)
                              (oriTracking "NONE") (posTracking "NONE") (visibleInWIM FALSE))
...
f-236   (MAIN::isVisible (id 55) (value TRUE)
                         (object <External-Address:edu.columbia.cs.cgui.jess.JS_PhysicalObject>))
f-237   (MAIN::inViewFrustum (id 55) (value TRUE))
...
f-741   (MAIN::represents (obj1ID 195) (obj2ID 55) (origin WORLD))
f-742   (MAIN::WorldObject (class <External-Address:java.lang.Class>) (id 195)
                           (OBJECT <External-Address:edu.columbia.cs.cgui.jess.JS_WorldObject>)
                           (collider nil) (collision FALSE) (groupID 5) (inUse TRUE)
                           (inViewFrustum TRUE) (keepVisible FALSE) (name "Rockwell")
                           (occluded FALSE) (occluding FALSE) (visible TRUE) (boundingBox nil)
                           (displayMode "NONE") (inWIM FALSE))
f-743   (MAIN::displayMode (id 195) (value WIREFRAME))
f-744   (MAIN::isVisible (id 195) (value TRUE)
                         (object <External-Address:edu.columbia.cs.cgui.jess.JS_WorldObject>))
f-745   (MAIN::inViewFrustum (id 195) (value TRUE))
...
f-1434  (MAIN::conveyNamesForObjectGroup (groupID 5))
...
f-1646  (MAIN::conveyName (objectID 55))
...
f-1678  (MAIN::OverlayObject (class <External-Address:java.lang.Class>) (id 258)
                             (OBJECT <External-Address:edu.columbia.cs.cgui.jess.JS_OverlayObject>)
                             (collider nil) (collision FALSE) (groupID 5) (inUse TRUE)
                             (inViewFrustum TRUE) (keepVisible TRUE) (name "Rockwell Label")
                             (occluded FALSE) (occluding FALSE) (visible TRUE) (anchorID 55)
                             (isLabel TRUE) (isOnFrontClippingPlane TRUE) (labelString "Rockwell")
                             (mayBecomeTransparentUpTo 0.6) (mayChangeSizeBy 10.0)
                             (percentSize 100.0) (textColor "REGULAR")(transparency 0.0))
f-1679  (MAIN::isAnchoredAt (obj1ID 258) (obj2ID 55) (anchorType nil))
f-1680  (MAIN::textColor (id 258) (value "HIGHLIGHTED"))
f-1681  (MAIN::isVisible (id 258) (value TRUE)
                         (object <External-Address:edu.columbia.cs.cgui.jess.JS_OverlayObject>))
f-1682  (MAIN::inViewFrustum (id 258) (value TRUE))
```

Figure 6.5: Jess facts describing a physical object, and two virtual objects that represent and annotate it.

ID or object reference. More sophisticated knowledge base debugging and visualization tools could help alleviate this situation.

In Ruby's current implementation we use both internal and external attributes. For most of the external attributes we use, we keep corresponding internal attributes, which for efficiency reasons are not updated in realtime but only when the programmer (or knowledge base engineer) requests it. Keeping these attribute copies is very useful for debugging purposes. Until the user synchronizes the database, however, the internal slots may hold the wrong values. An example of this can be seen in Figure 6.5, where the overlay object (fact f-1678) still shows a text color of `"REGULAR"` in its internal attributes, but it is already `"HIGHLIGHTED"`, as external attribute fact f-1680 shows.

External attributes make most sense if changes to the attribute state occur frequently. The visibility attributes `isVisible` and `inViewFrustum`, for example change quite often due to user head motion. These are the attributes that visibility decisions in rules are based on.

There is a fine line between external attributes and relationships, events, and goals. They are all represented as facts in the knowledge base, linking back to the objects they are related to through IDs or object references. Relationships, such as `represents` and `isAnchoredAt` are facts with links to two (or more) object IDs. Events are represented as changes in the knowledge base. These can occur as assertions of new facts, or as modifications of existing facts. Any arbitrary application event, sensor event, or user event can be hooked up to Ruby simply by letting it change knowledge base state. Changes in tracking accuracy register through modifications of the `oriTracking` and `posTracking` slots of physical objects. Head motion comes into play through slots in the user object, as illustrated in Figure 6.6. Objects entering or exiting the view frustum are noticed by changes in the respective attribute slots. Occlusion is treated in a similar fashion. A `Scene` object keeps track of scene boundary events.

Goals, finally, can be expressed either as internal attributes of the `user` object, or as standalone facts. A standard presentation goal is to convey the names of all environment object. A fact `conveyAllNames` triggers creation of more specific goal facts `conveyNamesForObjectGroup`, parametrized by a group ID, and finally `conveyName`, parametrized with an object ID. If such an object then enters the view frustum, an overlay object is created and displayed as a label. A goal that is stored with the user object is `navigateTo`, which triggers a mode in which rapid head motion causes a change in the UI (cf. Figure 6.6 and Section 6.2.2).

```
(defrule set-labels-transparent-on-head-motion
   (User (headMotion FAST) (goal NAVIGATE_TO_OBJECT))
   fact <- (OverlayObject (isLabel TRUE) (inUse TRUE) (id ?oid)
                          (transparency ?tr&:(< ?tr 1.0))
   ;; limit this rule to visible labels, for performance reasons:
   (isVisible (id ?oid) (value TRUE))
 =>
   (modify ?fact (inUse FALSE) (transparency 1.0))
   (assert (turnedTransparent (id ?oid) (oldValue ?tr))
)

(defrule set-labels-opaque-on-head-motion
   (User (headMotion ?hm&:(neq ?hm FAST)) (goal NAVIGATE_TO_OBJECT))
   fact <- (OverlayObject (inUse FALSE) (isLabel TRUE) (id ?oid))
   ttf <- (turnedTransparent (id ?oid) (oldValue ?tr))
 =>
   (modify ?fact (inUse TRUE) (transparency ?tr))
   (retract ?ttf)
)
```

Figure 6.6: Simple Ruby rules controlling label display during fast head motion.

6.1.4 Ruby Rules

At the center of Ruby's event and UI-management architecture lies the rule engine. Rules react to events, which, as described in the previous section, are expressed as changes in the knowledge base. All rules are coded in the Jess scripting language. Currently, there are 264 rules implemented in Ruby, stored in different modules for scene creation, menu management, and UI adaptation. These rules watch over a set of facts that, for the Columbia campus scene starts out with about 1200 facts and can grow to about double that size dependent on the specific UI choices and user tasks. The rule engine is constantly updating the system state encoded in the knowledge base, reacting to any new facts and modifications. Figures 6.6 and 6.7 show example Ruby rules.

An important programmatical concept in Jess rules is that variable bindings can be made during pattern matching and used later throughout the same rule. Also, patterns can include test predicates, such as boolean expressions (second pattern in Figure 6.7) and equalities and inequalities (as illustrated by the transparency slot in the first overlay object pattern in Figure 6.6).

The rules in Figure 6.6 show how labels can be made invisible during fast head motion, if the user is in navigation mode. Together with several other rules, these implement the UI behavior presented in Section 6.2.2. The first rule, paraphrased in English, reads like this: If the user is in `NavigateTo` mode and is moving his or her head fast, and there is a label that is currently in use, not fully transparent, and currently visible, then make this label fully transparent and save the old transparency value in the notifica-

```
(defrule world-object-add
   ; if there is a PhysicalObject:
   (PhysicalObject (id ?id) (OBJECT ?physObject))
   ; and nothing represents it yet in the world, or whatever it
   ; represents is not a World Object:
   (or (not (represents (obj2ID ?id) (origin WORLD)))
       (and (represents (obj1ID ?wid) (obj2ID ?id) (origin WORLD)))
            (not (WorldObject (id ?wid))
       )
   )
   ;; for now: only one world object representing a physObj
   ;; is possible
 =>
   ;; DEBUG: (printout t "world-object-add id=" id)
   ; retrieve scene object from physical object:
   (bind ?scene (call ?physObject scene))
   ; create new world object from physical object and scene origin:
   (bind ?nwo (new edu.columbia.cs.cgui.jess.JS_WorldObject
               ?physObject (call ?scene getSceneOrigin)))
   ; retrieve id for new object:
   (bind ?nid (call ?nwo getId))
   ; add represents fact:
   (assert (represents (obj1ID ?nid) (obj2ID ?id) (origin WORLD)))
   ; add definstance fact for world object:
   (definstance WorldObject ?nwo)
)
```

Figure 6.7: Ruby rule creating a virtual world object for each physical object.

tion fact `turnedTransparent`. The second rule restores the old situation when there is no fast head motion anymore. Note how entire facts can be stored in variables during rule matching, so that the facts can be modified or retracted in the action part of the rule.

The rule in Figure 6.7 defines how a world object is created for each physical object while populating the knowledge base initially, after the physical objects were instantiated from a persistent database. The idea behind this is that we keep a virtual world object for each physical object the knowledge base knows about. This virtual world object is collocated with the physical object, and it can be used to highlight the physical object and to clarify occlusion relationships (cf. Section 6.2.1). The first pattern matches in turn with each physical object in the knowledge base and retains the ID and object reference in variables. The boolean expression that follows, linking several patterns together, only gets matched if there is no `represents` relationship yet that links to the physical object matched before, or if there is a `represents` fact, but the object that represents the physical object is not a world object (it could be a WiM object or an overlay object). If there is already a world object that represents the physical object, the rule does not get matched. The first action retrieves the scene object the physical object is associated with, since the new virtual world objects should live in the same scene. The next actions create a new world object in Java, retrieve the unique ID that the new object

got assigned, assert a new `represents` relationship between this new object and the physical object, and add a definstance fact for the new object to the knowledge base.

6.1.5 Ruby Control Flow

In this section we give a quick overview of how the knowledge base is instantiated and maintained at runtime, what other data structures keep track of application state, and how data is passed among the different modules of Ruby to make things work.

Ruby is started as a Java application that creates a Jess rule engine, and instantiates the manager modules and Java3D-based realization backend (cf. Figure 6.1). A property file is read in, in which various startup parameters are listed, such as for example what trackers are connected to the MARS at startup. There is no automatic device discovery yet. The tracker gets instantiated by the main java class, and initiates scene creation in the model/scene manager.

All Ruby knowledge base content originates from information stored in a persistent database. This means that the MARS either needs network access, preferably wireless, to a database server, or the database can be stored on the MARS itself. A network connection is required if the MARS is to retrieve live information updates from database servers at runtime. We use Microsoft SQLServer as our database backend. The information is kept in different tables for objects and their relationships. Geometrical models and multimedia snippets are stored in file or web repositories, and only the pointers to them are kept in the database. Information flow in Ruby starts with the scene and object tables from the database.

The rule engine starts up with an empty knowledge base. A few administrative facts are added from a script file. Facts for the default menu structure are created, parametrized by the properties file. Template definitions for all non-definstance knowledge base objects and query definitions are loaded from scripts for future use. Then, all the rules are read in and instantiated in the rule engine. We use several script files for storing the rules, reflecting different functional modules for scene creation, menu management, and UI adaptation.

Triggered by tracker and property file information, the model/scene manager determines the current scene, and reads the scene information from the database. A scene object maintains a hash table with the physical objects and pointers to any geometry data and other related information. One by one, the physical object facts are instantiated in the knowledge base. The actual object locations, which are stored in the database, are not reflected in the knowledge base, but instead, the visibility data structures in the realization engine are initialized with them.

The presence of a definstance fact for a new physical object in the database trig-

gers the rule depicted in Figure 6.7. Thus, for each new physical object added to the knowledge base, a collocated virtual world object gets created. The scene object is consulted to retrieve the pointer to the geometry definition, and, if available, a VRML model representing the object is loaded and added to the Java3D scene graph, without yet getting displayed. Once all the physical objects for the current scene are read in and all corresponding virtual world objects are created, the knowledge base is in a stable state. All further rule-firing activity depends upon user activity or other events that modify the system state in the knowledge base. For example, the user can bring up the main interaction menu (by pressing a button on an interaction device) and make a selection that asserts a goal fact into the knowledge base or calls a pre-defined function that makes suitable modifications to the knowledge base.

Figure 6.5 depicts a few example facts that the knowledge base contains after the user chooses to display virtual world objects as wireframe overlays, show labels for physical objects, and mouse over a particular label. The depicted facts all refer to a computer called “Rockwell”. Fact 116 represents the physical computer object. Facts 236 and 237 are external attributes, reflecting the visibility conditions of the object. Facts 741–745 are the result of the rule from Figure 6.7 and a subsequent rule cascade. They realize the virtual world object that stays collocated with the physical computer. The user selected wireframe display of all virtual world objects. The knowledge base is informed by the visibility manager that computer Rockwell is currently in the user’s view. The wireframe model of a computer monitor is displayed in the spot that to the MARS’s best knowledge coincides with the physical location of the real monitor.

Then the user chose to bring up labels. Consequently, the goal fact 1434 got added to the knowledge base, and after some more rules fired, the specific goal to convey the name of computer Rockwell is added (fact 1646). In response to that, an overlay object representing the “Rockwell” label gets created (fact 1678). When the user mouses over that label, the external textColor attribute gets modified to `"HIGHLIGHTED"`. This is when the snapshot of the knowledge base was taken. A change of the `textColor` attribute is reflected immediately in the Java3D scene graph, changing the color of that particular label to yellow. The mapping between symbolic and specific colors is hardcoded in Java code for now.

This concrete example gives an overview of how the different modules in the Ruby architecture interact with each other, how the knowledge base represents and maintains the current system state in a simplified symbolic way, and how the rule engine ensures that all events are properly taken into account and the UI assembled accordingly.

6.2 Examples

In this section we present three examples implemented with Ruby and the formalized partial MARS UI component taxonomy. The first use we made of the new architecture was not for the purpose of an adaptive UI but for convenient interactive testing of various display options for occlusion (Section 6.2.1). In a second example, we adjust an outdoor mobile MARS UI for navigational guidance, dependent on the extent of a user's head motion: while keeping the head relatively still, world-stabilized overlays populate the user's field of view. During heavy head motion this interface is swapped for an entirely screen-stabilized navigational guidance message (Section 6.2.2). The third example illustrates the case of a user interface adapting to different levels of tracking accuracy. While underneath a high-precision 6DOF ceiling tracker, we display world-stabilized labels and annotations for objects in the environment. When leaving the area covered by the ceiling tracker, a much more approximate tracking method takes over, based on sparsely positioned infrared beacons (Hallaway et al., 2004). In response, the interface is smoothly changed to a choice of several WiM-based displays (Section 6.2.3).

6.2.1 Exploring Display Modes for Occluded Infrastructure

The first application that we put Ruby to use for, exhibits only limited adaptive UI behavior, but it nevertheless highlights the flexibility and potential of the overall approach. We were concerned with testing different combinations of drawing styles for occluded objects, a topic that we since then further explored in a user study, performed at the Navy Research Laboratory (Livingston et al., 2003).

For our example, we added rules to Ruby that are fired whenever a new object partially enters or leaves the view frustum, and whenever a new (partial) occlusion takes place because of changes in the user's head pose. This way we can change the style of an object representation when the object becomes partially or fully occluded. The notifications about objects entering the view frustum basically provide "computational clipping" – we can efficiently deal with changes in the appearance of virtual objects, since we only have to consider those objects currently in the view frustum (which are automatically tagged by the firing rules).

The rule-based architecture also proved very useful for testing different combinations of display styles for occluded objects, since the Jess shell allows for online modifications of system state without having to recompile or even to restart the application. Figure 6.8 shows four different examples of such display style combinations for the virtual object representing the occluded "Steve's Office", which is located behind the laboratory's wall and door. Picture (a) shows the occluded object labeled and outlined in wireframe mode. Visible objects in the user's view are labeled, with the labels avoiding

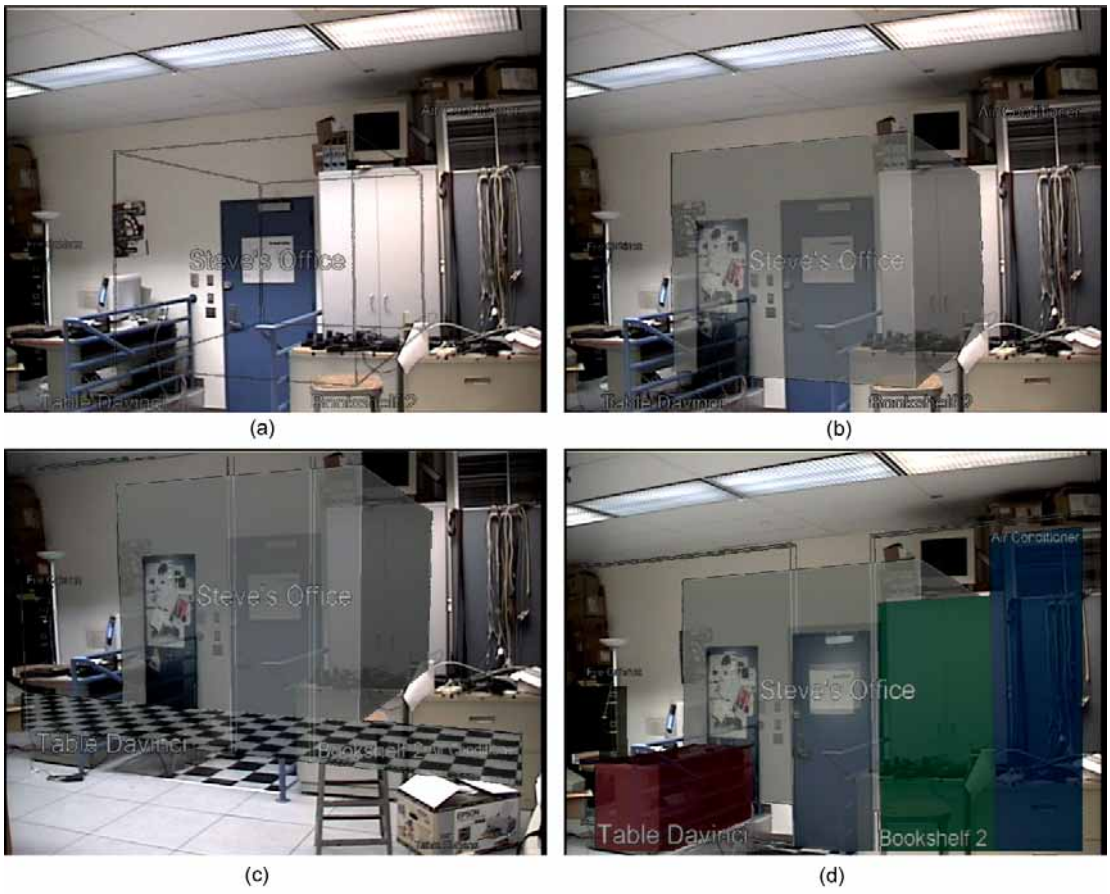


Figure 6.8: Testing different visual styles for depicting occluded objects interactively. (a) Wireframe. (b) Solid. (c) Solid with wireframe outlines of walls and textured floor. (d) Additional virtual overlays for other occluding objects.

overlap with the occluded object. Picture (b) shows the office labeled and solidly shaded; (c) adds the outlines of the wall parts, which are drawn in wireframe since they occlude the office, and the outer corridor floor, which is drawn textured, since it does not occlude anything important. As none of these examples satisfactorily conveys the occlusion relationship, we manually added shaded outlines associated with the visible represented objects in the foreground (table, bookshelf, and air conditioner). To provide an example of the simplicity of such scene modifications, the line we had to type in the Jess shell for rendering all visible virtual world objects in solid shading mode (change from (b) to (d)) was

```
setVirtualRenderings TRUE SOLID
```

Ruby's instantaneous knowledge-base updates of occlusion events and the immediate reaction to that via the forward-chaining rule base ensure that the system can adapt the UI to new visibility conditions in real time. This is an advantage over the application of "evaluators" that were run in a decoupled fashion as part of a planning computation in the rule-based IBIS (Feiner and Seligmann, 1992; Seligmann, 1993).

6.2.2 Adapting to Mobile User Context

Our second example is a simple application of dynamic UI component design, as set forth in our UI Management Pipeline (Section 5.4.3). Depending on the extent of a user's head motion, we change a UI for navigational guidance from a world-stabilized labeled AR display of the target building to a simple screen stabilized message identifying the relative location of the target.

Figure 6.9 illustrates this adaptive UI. The example shows a campus scene at the University of California, Santa Barbara. Our development setup for this test was to film a video of a campus view, turning a tripod-mounted camera in different directions at different speeds, in order to simulate a person who is trying to orient him- or herself by looking around. The camera is orientation-tracked by an IS-300 InertiaCube, and the spot on which the tripod is placed is carefully calibrated against a local campus origin using centimeter-accurate GIS and CAD data of campus infrastructure. Until we set up an RTK differential GPS for the UCSB campus, we do not have the means for outdoor position tracking at that level of accuracy.

We synchronize the tracked video with playback of the logged orientation data by means of an audio signal recorded when starting the tracking log. Thus, we can experiment with overlaying a variety of live-generated AR material on top of the video during playback.

In order to adapt the interface to head motion, we implemented a simple head motion classifier, which keeps track of the angular differences between pairs of subsequent

(a) (b)

Figure 6.9: User interface adapting to user's head motion. (a) Standing still: world-stabilized overlays of building model and target pointer. (b) With considerable head motion the view changes to a completely screen-stabilized message.

tracker readings and runs these through a simple box filter. We tweaked the parameters to differentiate between three kinds of motion: no motion, slow and steady head motion, and fast or jerky head motion. The detected head motion state directly modifies the `headMotion` slot of the `User` fact in the knowledge base, which affects several rules in the rule base (cf. Figure 6.6).

We use this setup for the following simple adaptive user interface: As in the previous example, we have rules set up to detect objects entering and leaving the view frustum. When there is no or slow head motion, we display the target location of our user's orientation task as a direct 3D overlay when the (often occluded) object falls into the user's view frustum (see Figure 6.9(a)). Whenever the target location is outside the view frustum, we show world-stabilized labels for all buildings in the user's field of view (in a different style than for the target object) and a screen-stabilized arrow pointing into the direction the user should turn to locate the target.

When there is fast head motion, we do not display a world-stabilized AR interface at all, but instead show a screen stabilized text message if the target lies in front of the user (in their field of view), as shown in Figure 6.9(b). The rationale for this UI behavior is our informal test result that — unless one has excellent tracking and registration — world-stabilized overlays tend to annoy users during rapid head motion, since they bounce around quite a bit. Also, when questioned about this effect, users stated that when looking around fast they usually do so in order to purposefully change direction, and not to scan the environment. Only during slow turns do they actually make use of world-stabilized information.

Since the time we captured the images in Figure 6.9, we have updated our UIs by

Figure 6.10: User interface adapting to different tracking accuracies. (a) Tracked by accurate 6DOF ceiling tracker: World-stabilized annotations. (b) Tracked by coarse wide-area infrared tracker: Screen-stabilized WiM.

changing the screen-stabilized text message to an arrow pointing forward, as we found in experiments with an HMD that the user's eyes are not easily kept focused at the text on the HMD's view plane, when he or she looks around quickly.

6.2.3 Adapting to Tracking Accuracy

The third example presents a UI that adapts to different tracking accuracies. For setup we used the same "tracked video" approach as in the last example. This time we mounted two different sensors on top of a digital camera that we attached to a "GlideCam" hand-held stabilizer for filtering physical camera motion. The first sensor was an InterSense 6DOF station for use with the IS900 ceiling tracker that covers most of our lab; the second was a homebrew *IrDA* tracking system consisting of eight infrared receiver dongles arranged in a circle for omnidirectional receiption of signals from wall mounted strong infrared senders (Hallaway et al., 2003). This tracker tracks 2D (x and y) position only, assuming a constant height (the height of the user), and in terms of orientation it only tracks yaw. The tracking performance is approximately as follows: With ten beacons covering approximately a 30x30 foot area, our system tracks a user to an observed (non-uniform and layout-dependent) average x and y position accuracy of about one to four feet. It tracks yaw roughly, with inaccuracies up to 20–40 degrees. It also suffers from half a second to a second lag. We logged the time-stamped tracking data from both trackers to disk and made sure we could synchronize it with the video later on by marking the start of the logging with an audio signal.

The goal of the UI is to give the user an overview of a lab environment. Different

workspaces, shelves, cabinets, and some computers are labeled with their names or short descriptions as the user looks around. Figure 6.10(a) shows what the interface looks like for the precise tracker. From this particular viewpoint we see three objects labeled with accurate registration. We can now simulate a handoff to the low accuracy tracker (we can do that at any time since we have both tracking streams available for each timestamp). Figure 6.10(b) illustrates the resulting view: A screen-stabilized WiM that tries to stay aligned with the world around the user, but obviously can only do so to the accuracy of the tracker. Also note that, in contrast to the WiM in Section 5.3.2.1, the lab model here always keeps the same size and stays at the same viewing angle, since the coarse tracker does not provide any pitch information. Note that only the objects that are currently in the user's field of view (according to the rough position and orientation readings), are labeled in the WiM. This is because of the specific goal to convey the names of currently visible objects. As in the previous examples, the notion of objects entering and leaving the view frustum is controlled by the rule base.

The parameters for the WiM (full labels, no labels, only labels for visible objects, head-pitch control and various parameters for mapping head pitch to size, angle, and screen position) are all represented in the knowledge base. The MARS can detect which tracking systems are available at each point in time and updates this information in the knowledge base (simulated here by user input). We represented the notion of tracking accuracy in the knowledge base. For this particular example we simply distinguish between different levels for both position and orientation tracking (`"LOW"`, `"HIGH"`, and `"NONE"`). Note that we can simulate all nine combinations of no, low, and high accurate position and orientation tracking easily from our tracking logs by splitting up the 6DOF tracking samples into their orientation and position components. Such combinations are also practically relevant, since we can easily replace the IS900 tracker by a combination of a 3DOF orientation tracker (e.g. IS300 InertiaCube) and a 3DOF position sensor (e.g. wireless IS600 SoniDisk) (Höllerer et al., 2001b).

When both orientation and position tracking are accurate, we want fully registered world-stabilized labels. When orientation is accurate, but position tracking is only approximate, we apply the WiM interface from Section 5.3.2.1, dubbed hereafter *default-WiM*: head-pitch controlled WiM with full labels appearing when looking downwards beyond a point. The user's approximate position is displayed as a red dot around which the WiM is centered.

If orientation tracking is low, neither of these approaches makes much sense, so we instead show what is illustrated in Figure 6.10(b) and described above: A more constrained WiM with labels for only the visible objects (*constrainedWiM*).

If there is no orientation tracking at all, there can be no detection of the objects currently in the view frustum. The only way to convey the visible objects then, is to

Orientation Tracking ↓ / Position Tracking →	High Accuracy	Low Accuracy	None
High Accuracy	**accurate overlay**	**WiM w. head-control (“defaultWiM”)**	**WiM with all labels, no user marker**
Low Accuracy	**"constrained” WiM**	**"constrained” WiM**	**"constrained” WiM with all labels, no user marker**
None	**static WiM with all labels**	**static WiM with all labels**	**static WiM with all labels, no user marker**

Figure 6.11: UI decision matrix for different availabilities and accuracies of orientation and position trackers.

convey all objects in the environment. Also, without orientation tracking, head-motion control of the WiM is not possible at all. Therefore the UI displays a static WiM from the bird’s eye perspective, north direction oriented upwards.

If there is orientation tracking but no position tracking, we still cannot determine the objects in the user’s view, but we can use head-motion control for UI components. With the accurate orientation tracker, we show the default WiM, but we omit the user marker, since we do not know the user’s position. With low accuracy orientation we show the constrained WiM, but with all labels and no user marker. We assume that even without position tracking, the MARS will still learn about the user leaving the current room or scene, so that the WiM representation can be changed in that case.

Figure 6.11 depicts the 3x3 decision matrix that maps available tracking accuracy to the different UI behaviors just described. The following high-level rules (omitting implementation details) are needed to trigger the transitions among the UI possibilities from the decision matrix. Remember that rules only fire in the Rete algorithm when there is a change in input conditions that causes one of the left-hand-sides to evaluate to true.

```
((User (oriTracking "LOW"))
       → <smooth transition to constrainedWiM>)

((User (oriTracking "HIGH") (posTracking "LOW"|"NONE"))
       → <smooth transition to defaultWiM>)

((User (oriTracking "HIGH") (posTracking "HIGH"))
       → <smooth transition to fullOverlay>)

((User (oriTracking "NONE"))
       → <smooth transition to static WiM, showing all labels>)
```

```
((User (posTracking "NONE"))
       → <show all labels, hide user marker>)
```

In our initial implementation, the `oriTracking` and `posTracking` slots in the `User` fact were modified directly from Java listener objects that are registered to react to changes in availability of the employed trackers. To handle trackers even more flexibly, we can represent each tracker sensor as a separate fact in the knowledge base, with relationship facts denoting the entities they track:

```
(Tracker (id 243) (name "IS900Station3") (inUse TRUE)
         (oriTracking "HIGH") (posTracking "HIGH"))
(Tracker (id 244) (name "IS600Station2") (inUse FALSE)
         (oriTracking "NONE") (posTracking "HIGH"))
(Tracker (id 245) (name "IrDA") (inUse FALSE)
         (oriTracking "LOW") (posTracking "LOW"))
(Tracker (id 246) (name "IS300ProStation1") (inUse FALSE)
         (oriTracking "HIGH") (posTracking "NONE"))
(Tracker (id 247) (name "GPS") (inUse FALSE) (oriTracking "NONE")
         (posTracking "LOW"))

(User (id 20) (headMotion "FAST") (walking FALSE)
      (goal NAVIGATE_TO_OBJECT) (focusObject 673)
      (oriTracking "HIGH") (posTracking "HIGH"))

(tracks (tid 243) (oid 20) (orientation TRUE) (position TRUE))

(TrackingStation (id 250)
      (tracksObj <External-Address:edu...jess.JS_PhysicalObject)
      (oriTracker <External-Address:edu...jess.JS_Tracker)
      (posTracker <External-Address:edu...jess.JS_Tracker))
```

The `Tracker` and `TrackingStation` facts, like the `User` fact, are definstance objects. The Java object for `TrackingStation` controls the frame-by-frame position and orientation updates for the respective tracked object, based on the sensor information from the one or two trackers assigned to it. The `tracks` attribute denotes the fact that a specific tracker is physically set up to track a particular object, but only when the `Tracker` objects are registered within the `TrackingStation`, do the pose updates actually get taken into account.

The following rule replaces an orientation tracker with a more accurate one, if available. Note that this rule works on the user object, since the user is a special case of a physical object.

```
( po <- (PhysicalObject (id ?poid) (oriTracking ~"HIGH") (OBJECT ?op))
  tr <- (Tracker (id ?tid) (oriTracking "HIGH") (inUse FALSE)
```

```
                (OBJECT ?ot))
        (tracks (tid ?tid) (oid ?poid) (orientation))
  ts <- (TrackingStation (tracksObj ?op) (oriTracker ~?ot))
  →
  (modify ?tr (inUse TRUE))
  (modify ?ts (oriTracker ?ot))
  (modify ?po (oriTracking "HIGH"))
)
```

A similar rule watches over the position tracking. In reality, we use a few more rules: `TrackingStation` objects need initially be created for every tracked object. We also want to invoke a transition from `"NONE"` to `"LOW"` when a new tracker becomes available (the above rule only triggers "`"NONE"` to `"HIGH"`" or "`"LOW"` to `"HIGH"`" transitions). Also, we find it preferable to change both position and orientation tracking to a single high accuracy 6DOF tracker if one is available, even when a different tracker already tracks either orientation or position with high accuracy. Furthermore, if multiple trackers with accurate orientation tracking are available, we might want a say in which one gets chosen. Finally, we need to make sure that all rules work in both possible directions: the effects of a rule should be undone when the conditions for rule invocation cease to be true. Jess provides a special construct (`logical`) to automate this process for rules without computational side effects, but in the general case, the rule designer needs to take care of this.

With the above logic in place, we can now easily manage trackers dynamically. The value of the tracking accuracy slots can change based on the environment and user context. Rule-based tracker control applies to outdoor as well as indoor navigation. Outdoors, GPS tracking accuracy can be `"HIGH"` if the tracker is in RTK differential mode, `"LOW"` when losing the differential signal, and `"NONE"` when losing a position lock due to a lack of visible GPS satellites. Magnetometer-based orientation tracking can get distorted by environmental factors, such as subway tracks running underneath the user's current location. Suppose that outdoors the user's head is tracked with GPS and an IS300 Pro orientation tracker. On entering a building, he or she loses GPS tracking. If the user's MARS gets access to any kind of position tracking in the building, be it through worn sensors communicating with building-mounted tracking equipment, or a dead-reckoning method, such as the one described in Section 3.3.1, the above Ruby code causes the new tracking information to be used. The rule base can be extended to make other tracking decisions. If the knowledge base is notified whenever the user enters areas affected by large magnetic distortions, it can replace a magnetometer-based tracker with other technologies. There are already rules in place that react to the user crossing boundaries between different scenes and rooms (cf. Section . 6.1.3.2). Magnetic distortion can be modeled on a room by room basis.

We have described three simple example applications implemented in the Ruby framework. All control logic for these examples is cleanly encoded in the form of rules. Rules can interact in intuitive ways that the programmer does not necessarily have to foresee.

Even though the three examples were initially coded to run independently from each other, they can be easily made to work together. All the interactive explorations from the first example work without a change in the latter two examples. A combination of the rule bases for the mobile user context and tracking accuracy examples is straightforward. We only need to define the respective goals carefully, and understand and model the limitations that are inflicted by changes in the resources. The overall goal in the mobile user context example is to navigate to a specific target object. The system highlights the target object if possible, and gives orientation directions otherwise. In the tracking accuracy example, the overall goal is to provide an overview of the environment, specifically of the objects in the user's view. We can test a combination of the two examples by forming the union of the respective goals: "Direct me to a target object and inform me about the objects in my view!"

We have to understand that there can be no head motion detection when orientation tracking is not available. In that case, the tracking adaptation rules cause a static WiM view of the current scene environment, with all objects labeled, and the rules from the navigation example cause the target object to be pointed out in the WiM specifically. For this to work, we need additional rules that point out a WiM representation of an object if the object cannot be properly located in the world view. These are general rules, that will be of advantage in other applications as well. All the semantic relationships between a WiM object, and the corresponding virtual world and physical objects are already represented in the knowledge base. If we have position tracking, the user's location is pointed out in the WiM as well (this is the default behavior from example three).

When orientation tracking is available, either approximately or accurately, we can detect head motion again. As long as orientation and position information are not both accurate, the tracking adaptation rules will trigger the display of a WiM in different styles according to the decision matrix in Figure 6.11. When there is rapid head motion, the rules from the mobile user context example will switch off all labels. The rules are general enough to apply to WiM object labels. However, we still have the goal to point out the target object. If there is a rule in the knowledge base that allows for label-independent highlighting of objects (such as a color change of the object geometry or drawing a bounding box or screen-placed bounding rectangle), it will get fired and the target pointed out that way. If additionally, position tracking is working, the user's location is marked in the WiM (again, default behavior from example three). If we want more navigation direction functionality in the WiM (e.g., path displays), additional rules

need to be written.

When both orientation and position tracking are precise, the combined example is identical to the mobile user context example alone. No new factors have to be taken into account. In the presence of strong head motion, the UI reacts as described in Section 6.2.2.

6.3 Discussion

With Ruby we implemented a radically new system design as compared to the previous MARS architectures presented in Chapter 3. The main motivation for deciding on this approach was the increased flexibility it affords. Flexibility is of utmost importance in MARS UI design. A UI that works well in a specific situation can suddenly be rendered useless because of some unforeseen event, such as a change of tracking accuracy. When such a disruption occurs, the correct remedial action may depend on many different factors, including the user's goal at that moment, the UI elements on the screen, and other events that may have occurred or are about to occur. MARSs should be able to cope with such situations and provide the best possible UI. A UI-management technique should take into account the entire system state at the moment of disruption and implement a smooth correction of the problem.

We believe that a rule-based infrastructure can readily handle the complexity of many interacting concurrent events and comfortably resolve the dynamic constraints a mobile AR interface imposes. The power and flexibility comes from representing all UI objects symbolically in the knowledge base, thus expressing all relevant system state in one coherent formalism. We can reason about the objects' properties and implicitly react to state changes. Event management can be implemented in a much more goal-oriented fashion, compared to traditional approaches. Rule execution is based on the entire system state when a new event occurs.

Figure 6.12 compares Ruby's event management with the two most common imperative event models. Figure 6.12(a) depicts the way our previous MARS implementations (cf. Chapter 3) typically handled events. In a main event loop that is executed as fast as the dynamic UI updates allow, possible events are repeatedly checked using large nested `switch` and `if-then-else` statements. The sensors, input devices, and interactive UI elements are either polled directly, or, as the figure implies, record their state continuously in an intermediate "event state" data structure, which is then polled from the main event loop. In order to make any intelligent decisions about how to react to a certain event, the event handler code would have to consult global application state. To avoid having to repeat such queries in every single branch of the nested if statement, the application state can be recorded in state variables at the beginning of the event loop,

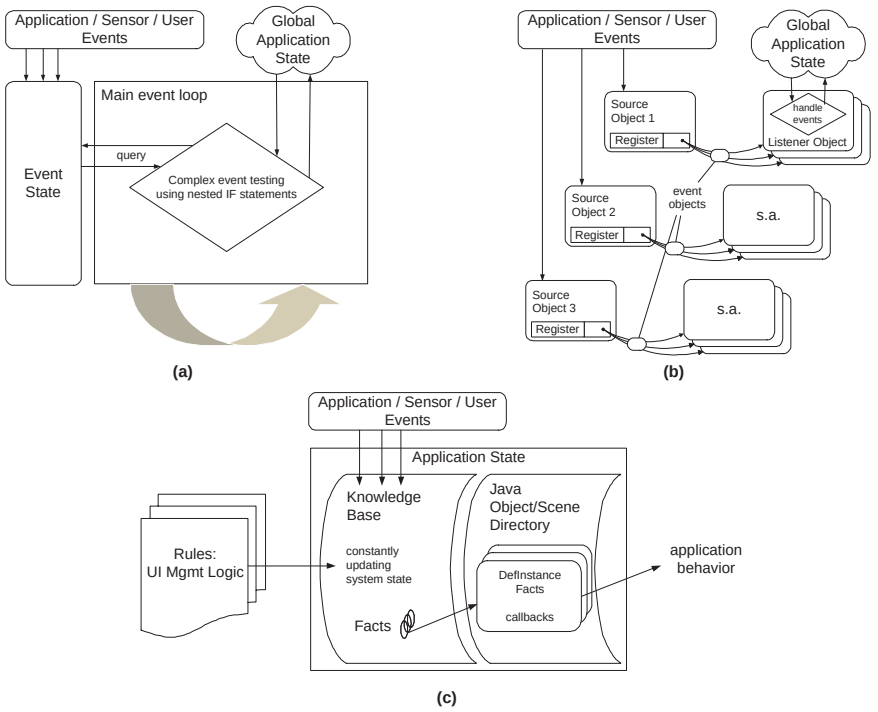


Figure 6.12: Event management according to three different system architectures. (a) Event loop model. (b) Event delegation model. (c) Rule-based event control.

but such a “summarization” only limits the access to other objects. The state variables themselves still need to be examined in each branch of the event testing. When the events and the system state are analyzed, the branches that were actually traversed apply their effects to the global application state.

Figure 6.12(b) depicts an event delegation model that is based on source objects and event listener objects. The Java event model since version 1.1 is a prominent example of this approach (Gosling et al., 2000). All events are triggered/mediated by source objects. In order to get informed about events, so-called listener objects register with the source objects. When the events occur, every listener object that registered with the respective source object gets notified to handle the event. Hence, all event handling happens inside of listener objects. This approach is more object-oriented than the one in part (a). Unfortunately, it is not better suited for handling adaptive interfaces such as those introduced in this chapter. Located in different objects as they are, the event handlers are even more isolated from each other. Global system state would need to be queried from external objects. Combinations of different events are hard to take into account. A more promising approach than responding to the events right there in the listeners would be to accumulate all event information in one common event state data structure and then proceed as in part (a).

Ruby's event control, depicted in Figure 6.12(c), is different from purely imperative event handlers in several respects. First and foremost, there is a formal representation of the entire application state, embodied by the Jess-based knowledge base and associated Java-based object directory. Second, the events directly affect the application state by changing attributes in knowledge base facts or inserting new facts. Third, all rules are considered all the time, reducing coding redundancy as compared with the conditional statements of (a) and (b) in which many checks are repeated in different branches. Fourth, the decision-making is occurring proactively, by the Rete algorithm's continually matching the rules with the knowledge base facts whenever there are any state changes.

The flexibility in the rule-based approach becomes increasingly apparent the bigger the set of UI components and/or events that have to be accommodated. Consider the navigational guidance example from Section 6.2.2. As implemented, it presents a simple switch between two straightforward UIs. The first one uses direct annotation of the target object and other buildings, implemented with overlay objects. The second one uses screen-stabilized UI elements indicating the direction to look/go. The UI is switched based on degree of head motion. We already saw how this example can adapt to different tracking accuracies when combining it with the example in Section 6.2.3. To extend the scenario further, assume that we want to take into account landmark objects. In addition to the target, we now also want to point out landmarks that help the user in their orientation. When the goal is set to navigate to a specific object, a set of relevant landmark objects is determined, based on the location of the user relative to the target. We specify as a sub-goal that we want to point out landmark objects whenever they enter our field of view. All the rules to detect `inViewFrustum` events are in place. One additional rule suffices to create new distinctive overlay labels for the landmarks to be displayed during smooth head motion. But what happens during rapid head motion? If the existing transition rules are sufficiently general, then all overlay objects are turned into screen messages (an extra rule would be necessary to define a specific format for landmark objects). If they were not coded to be general, only the target object is transformed, but all world overlay labels are probably switched off (compare rule displayed in Figure 6.6). However, in that case the knowledge base is aware that the sub-goal to point out the landmark objects is currently not satisfied. As soon as a rule or a set of rules would enable satisfation of that goal, it would get executed. This is what we called "goal-oriented" event management above.

The above scenario also highlights potential limitations of the rule-based paradigm. Rules could be over-generalized, such that new events trigger unexpected results that do not fit the situation. For example, displaying landmark objects in the same style as the target object in the screen-stablized UI is probably not a good idea. Rules could be incomplete, resulting in missing UI elements or faulty interactive UI behavior.

Programming with rules takes getting used to, especially for most programmers coming from an object-oriented background. Rule design can get much more complicated than the three simple examples above. One of the key design questions is, which decisions are easily made via rules, and which ones are easier to handle in object-oriented code. This is not easy to answer categorically – experience with this programming paradigm is crucial for keeping the rule base simple and modular.

Testing and debugging rules is hard. It is quite possible to create incomplete rule sets that cause an endless loop or do not trigger the expected behavior. Debugging tools, such as a fact visualizer / editor (written by Blaine Bell) and step-by-step rule execution alleviate some of the problems, but the programming ease of rule coding is by far inferior to the relative comfort provided by the debugging tools of modern imperative programming developments.

In this chapter we have presented design decisions and a concrete architecture for rule-based UI management, called Ruby. We have shown how the expert system shell Jess can be used to efficiently control AR UI design decisions. Using data formats that we optimized for use within Ruby, we implemented a considerable portion of the general MARS UI taxonomy from Chapter 4. We demonstrated how the different modules of the Ruby architecture work together to enable real adaptivity in MARS UIs. Three example applications give a first taste of the new flexibilities that this kind of system design opens up. We concluded this chapter with a discussion of the presented approach.

Chapter 7

Conclusions and Future Work

In this dissertation, we have explored user interfaces for mobile augmented reality applications. Through exploratory design of indoor and outdoor mobile AR interfaces we have demonstrated how the computer interface can be embedded into the real world and how the real world can become part of the computer interface. Whereas static desktop computers provide occasional windows of interaction to the world of computing, mobile computers can take the interface into the unrestricted space of the real world and enable context-sensitive adaptive applications that augment people's abilities in their daily mobile lives.

We have presented the system design of a series of mobile augmented reality systems (Chapter 3), ranging from the Situated Documentary architecture, which built on the Columbia Touring Machine, to Ruby, our rule-based architecture for adaptive MARS UIs. On these different MARS platforms we developed a large set of exploratory interfaces for applications in tourism, journalism, collaborative work, military training, and navigation and wayfinding (Chapter 5), exploiting new capabilities of hardware and software infrastructures as we implemented them.

One of the most outstanding features of MARSs is their potential to adapt to the user's situation, in particular to their location, viewing direction, their task, current activity, perhaps mood, and to dynamic changes in the available resources. Augmented reality is quite literally about staying on top of such dynamic conditions. *Mobile* AR exacerbates the situation by taking the computer away from a controlled environment, into the unpredictability of the real world.

Creating MARS user interfaces is hard. While there has been respectable progress in all technological aspects concerned with the requirements of MARSs, as reviewed in detail in Chapter 2, tracking and display technology is still not accurate, reliable, and convenient enough to create commercial MARS solutions. Because there is such a huge entrance hurdle to getting involved with MARS UI research, namely the investment of

time and resources needed for creating the necessary hardware and software infrastructures to make things work, less research has so far been performed on the user interface aspects of this new computing paradigm, and more on the technical prerequisites. However, the user interface side of things is hardly any less challenging, as becomes clear through iterative protoype application design.

Once the tracking and display requirements are addressed, it is fairly straightforward to create UIs that overlay simple computer-generated material onto a real world background. The difficulties and challenges lie in making such a user interface work with full generality. For example, the amount of overlaid material displayed for a certain scene, and its layout, depend on many factors including the number and location of the objects to be annotated, the viewing position and angle, the complexity of the annotations, the visual complexity of the screen-stabilized part of the UI, the need for visual feedback on application state, and so on. Views with an overabundance of information overlay tend to be cluttered and confusing. In order to be able to maintain a certain level of clarity, the application must be given substantial flexibility in handling the information display. We describe three stages of a UI-management pipeline to address these and other UI optimization issues: information filtering, UI component design, and view management (Section 5.4.3).

Flexibility alone, however, is not enough. The application framework also needs a lot of information about the components of the MARS UI. These components include interaction elements (e.g., for triggering computations and visualizations), the computer-generated annotations, and also the objects in the physical world that the MARS knows about (sensed objects and objects that are part of the environment model). A MARS interface can only be successfully adaptive if the system keeps track about all these objects, their type, purpose, and other properties. This is why, starting from our exploratory interfaces (Chapter 5) and from related work in the field (Chapter 2), we have developed a taxonomy of MARS UI components (Chapter 4). This taxonomy is geared towards use in an actual implementation, namely the rule-based MARS architecture Ruby (Chapter 6), in which the UI components are tagged with information about their type and purpose, which enables novel kinds of adaptive behavior. We have presented several examples of adaptive interfaces that we implemented using the knowledge-based infrastructure and embedded description of MARS UI components: a mobile UI adapting to the user's state of motion and navigational interfaces that adapt to changes in tracking accuracy, with annotations being shared between multiple representations of the same object.

7.1 Summary of Results

Going over the process stages: MARS architecture design; theoretical foundation and categorization; UI implementation and exploration; and adaptive interface design, this dissertation has presented the following specific research results:

1. Hardware and software system design of a series of MARSs, first extending and later superseding the Columbia *Touring Machine*, which was the first outdoor MARS.

2. A large set of application prototypes exploring MARS UIs, leading to a library of MARS components. We described our design experiences and lessons learned in Section 5.4.

3. A taxonomy of MARS UI components.

4. Introduction of the concept of *MARS UI management*, as set forth in a three-stage pipeline architecture.

5. A rule-based system architecture for user interfaces that adapt to user context.

Our hardware and software infrastructures, described in Chapter 3 formed the basis for a large set of UI prototypes that were built in an iterative design process.

We extended a simple campus tour concept to form the notion of *Situated Documentaries*. In three increasingly complex realizations of this application scenario, we built spatially distributed hypermedia systems, which let users experience an interconnected web of multimedia news stories, presenting events in the exact places that are relevant to the story. This application framework allowed us to implement and test many new user interface concepts, most notably a 3D world-stabilized interface for information access. Using this series of outdoor AR applications as a testbed, we identified the need for research in several areas. The need for better user guidance across different story threads was postulated and implemented in our later Situated Documentaries interfaces (Section 5.1.2). When we viewed the augmented campus scene from extreme vantage points (e.g. Figure 5.20), the need to remove clutter in the UI, and to correctly place annotations to take into account occlusion became apparent, leading to our work on information filtering and view management (Höllerer et al., 2001a).

In related indoor AR work, we began to tackle the problem of environment management with the long-term goal of developing a set of semi-automated behaviors that would ease the problem of controlling the positions and interactions of many virtual and physical objects in a world of multiple users, multiple displays, multiple interaction devices and multiple input and output media in general. Environment management is

an especially challenging task if it is to address the needs of mobile, collaborating users, whose proximity to other users, displays, and interaction devices may change rapidly and unpredictably as users move about. Our frameworks for these UI explorations were our Java-based environment for indoor-outdoor collaboration (Section 5.2.1) and the collaborative EMMIE environment (Section 5.2.2) by Butz and colleagues (Butz et al., 1999).

Using these infrastructures, we started exploring environments that span areas covered by a variety of tracking technologies, exhibiting different position and orientation tracking accuracies (Höllerer et al., 2001b; Hallaway et al., 2004). This led to the challenge of creating user interfaces that stably adapt to different tracking environments, by allowing the UI to choose different components depending on available tracking accuracies. The general concept of choosing different UI components based on user context, we coined "UI component design", which, together with information filtering and view management, completed the UI-management pipeline, which we described in Section 5.4.3.1.

As a well-suited navigational component that is independent of high-accuracy positional tracking, we revisited the concept of Worlds in Miniature (Stoakley et al., 1995) for AR (Sections 5.3.1 and 5.3.2.1). When applying our view management results to the WiM, the notion of shared annotations gave rise to data structures that identify properties of and relationships between AR objects (such as for example the "represents" relationship that states that a certain object depicted in a WiM represents a certain object in the physical world). These kind of relationships could most readily be expressed in our emerging rule-based infrastructure Ruby, which we designed as a more flexible framework for MARS UI Management.

7.2 Future Work

In this dissertation, we have developed a comprehensive series of MARS hardware and software infrastructures and application prototypes. Based on these, we have defined a practical taxonomy of MARS UI components and established methodology for adaptive MARS interfaces that dynamically rearrange themselves in response to changes in user context. We would like to extend the presented research in several areas.

7.2.1 Environment Management

We believe that the benefits of the rule-based approach to managing MARS UIs will become even more apparent, once we apply it to scenarios in which many different factors of influence to user context all come into play simultaneously. The research contributions of user interface management techniques for augmented reality can thus be seen

in the context of tackling the broader problem of ”environment management” (MacIntyre and Feiner, 1996a) for general computational infrastructures. Assume a world of many collaborating users being exposed to information in an environment of many computing devices (wearable computers, palm-top computers, or computers embedded in appliances, walls, or furniture). With multiple displays and multiple interaction devices available, it is important to keep the interface to such an overall computing environment clear and simple. An environment like this will ultimately be successful only if the interface to the available computational services is unified and if substantial computer support exists for decision-making on where, when and how to convey what information.

7.2.2 Scalable Real-Time Knowledge Processing

Future user interfaces will require more knowledge about their environment and their users than the prototype solutions developed for this dissertation. The knowledge bases that they will draw from might exceed the size of Ruby's rule base by several orders of magnitude. We will have to look into ways of modularizing and optimizing such knowledge bases and the corresponding inferencing algorithms in order to ensure scalability in terms of processing speed. Maintaining real-time response rates will be of utmost importance.

7.2.3 Support for Hierarchical UI Descriptions

One drawback of our rule-based approach to UI Management is the unusual programming paradigm of writing rules that execute object-oriented code as side effects when fired, triggered by events that bring about changes in the state of one or more objects in the knowledge base. Writing rules directly in the Jess formalism is not always straightforward, and at many places we had to create more complicated-looking rules than necessary, because the formalism is not streamlined for our purposes. It would be very beneficial for us to design our own rule-based language that is optimized for the purposes of maintaining knowledge about UIs.

One example would be better rule-authoring support for UI transitions. Often the UI changes its entire usage paradigm, for example when switching from world-stabilized annotations to a screen-stabilized annotated overview visualization (using a map or WiM). It is not easy to code rules on the level of specific UI components for these scenarios, because the programmer constantly needs to make sure that all cases are covered. We may have world-stabilized labels, highlights, navigational widgets, and data visualizations that need to be represented differently after the transition. It is not even sufficient to transform these components one by one, because there may be interrelationships between them. For example, navigational widgets may be placed purposefully

close to certain landmark objects, and the choice of the most suitable landmarks may be completely different in the world-stabilized and screen-stabilized UIs.

Based on our experience with coding rules for simple UI transitions, we would like to explore an approach, in which we have the ability to encode such transitions at a higher level of abstraction while still keeping the ability to make decisions based on the presence or absence of concrete UI components. That could be done by maintaining a hierarchical representation of the UI at each point in time instead of the current flat collection of components and relationship facts we have now. At the top of the hierarchical description we would encode the overall purpose and main characteristics of the UI, which would branch out to the more specific components on lower levels. Then it would be possible to code rules based on the high-level UI information, and the rule development environment could automatically generate the heads of specific rules that would need to be implemented in order to facilitate all the details of a transition to another type of UI.

7.2.4 Usability Evaluation

One major area of future work lies in the usability evaluation of our implemented UI techniques and in creating guidelines for future design choices by means of empirical user studies. Usability evaluation of UIs that are naturally and purposefully dynamic, flexible, and able to cope with the unpredictable is a very challenging topic. For meaningful user studies one has to constrain many variables to ensure repeatability and generality. Trying to strictly fix even just the hardware- and environment-related conditions for MARS is simply not achievable in the most accurate sense. Compromises will have to be made, and the set of dependent and independent variables will have to be carefully selected. So far, user studies for AR are usually very basic and typically carried out in controlled indoor environments. One initial exception to that rule is a recent study that the author of this dissertation helped perform (Livingston et al., 2003). The setup for this far-field outdoor AR study placed the subjects underneath an accurate ceiling tracker on the inside of a building looking out through wide open swing doors. One of the alternative setups was a mobile “tracking cage” that could bring the highly accurate and reliable ceiling tracker into the outdoors for repeatable tracking results.

The purpose of this investigation was to test the impact of different drawing styles on a user’s perception of a mobile AR scene with multiple layers of occlusion. The user should be enabled to infer the depth relationships between different physical and virtual objects presented in the AR scene. We designed a number of sets of display attributes for the various layers of occluded objects and conducted a user study to determine which representation best conveys to the user the occlusion relationships among far-field objects

(Livingston et al., 2003). The final goal of such investigations is to create a set of design guidelines that helps UI designers to create more informative and useful MARS UIs for different application scenarios.

7.2.5 Adaptivity in Non-AR UIs

Finally, our work on MARS is part of a broader research agenda on adaptive user interfaces that depart from the normal WIMP paradigm of the conventional desktop computer (cf. Section 4.1). We would like to extend the current work by pursuing several related research topics, including further development of ”environment management” in non-AR contexts, such as information visualization, multimodal interfaces, sensor networks, ubiquitous computing, and generally intelligent computing environments.

7.2.6 Additional Interaction Modalities

In the area of multimodal interfaces, we would like to extend the range of input and output devices for mobile and situated computing, and expand the presented infrastructure for managing user interfaces to take these new modalities into account. Our work on 3D interaction techniques so far has mostly relied on a limited set of input devices: a head tracker and a 2D wireless mouse or palmtop computer. We have also implemented support and started initial experiments with a microphone and camera. We would like to experiment with additional input modes, such as more sophisticated speech recognition, eye tracking, and finger tracking, all of which function in limited research environments but are not easily achieved in a general mobile setting. As part of this experimentation, we would like to expand our rule-based user interface management approach to determine in what particular situations the mobile user could rely on which modalities.

References

3G-LBS (2001). 3G location-based services. http://www.nttdocomo.com/, http://www.siemens-mobile.com/pages/isleofman/english/index.htm.

Abdelguerfi, M. (2001). *3D Synthetic Environment Reconstruction*. Kluwer Academic Publishers, New York, NY.

André, E., Finkler, W., Graf, W., Rist, T., Schauder, A., and Wahlster, W. (1993). WIP: The automatic synthesis of multimodal presentations. In Maybury, M. T., editor, *Intelligent Multimedia Interfaces*, pages 75–93. AAAI Press.

Appino, P. A., Lewis, J. B., Koved, L., Ling, D. T., Rabenhorst, D. A., and Codella, C. F. (1992). An architecture for virtual worlds. *Presence*, 1(1):1–17.

Azuma, R. T. (1997). A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385.

Azuma, R. T. (1999). The challenge of making augmented reality work outdoors. In Ohta, Y. and Tamura, H., editors, *Mixed Reality, Merging Real and Virtual Worlds*, pages 379–390. Ohmsha/Springer, Tokyo/New York.

Azuma, R. T., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B. (2001). Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47.

Azuma, R. T., Hoff, B., Neely III, H., and Sarfaty, R. (1999a). A motion-stabilized outdoor augmented reality system. In *Proc. IEEE Virtual Reality '99*, pages 242–259.

Azuma, R. T., Hoff, B. R., III, H. E. N., Sarfaty, R., Daily, M. J., Bishop, G., Chi, V., Welch, G., Neumann, U., You, S., Nichols, R., and Cannon, J. (1998). Making augmented reality work outdoors requires hybrid tracking. In *Proc. IWAR '98 (First Int. Workshop on Augmented Reality)*, pages 219–224, San Francisco, CA.

Azuma, R. T., Lee, J. W., Jiang, B., Park, J., You, S., and Neumann, U. (1999b). Tracking in unprepared environments for augmented reality systems. *Computers and Graphics*, 23(6):787–793.

Bahl, P. and Padmanabhan, V. (2000). RADAR: an in-building RF-based user location and tracking system. In *Proc. IEEE Infocom 2000*, pages 775–784, Tel Aviv, Israel.

Baillot, Y., Brown, D., and Julier, S. (2001). Authoring of physical models using mobile computers. In *Proc. ISWC '01 (Fifth Int. Symp. on Wearable Computers)*, pages 39–46, Zürich, Switzerland.

Barrilleaux, J. (2000). *3D User Interfaces With Java 3D*. Manning Publications, Greenwich, CT, USA.

Beadle, H., Harper, B., G. Maguire Jr., and Judge, J. (1997). Location aware mobile computing. In *Proc. ICT '97 (IEEE/IEE Int. Conf. on Telecomm.)*, Melbourne, Australia.

Behringer, R. (1999). Registration for outdoor augmented reality applications using computer vision techniques and hybrid sensors. In *Proc. IEEE Virtual Reality '99*, pages 244–251.

Behringer, R., Park, J., and Sundareswaran, V. (2002). Model-based visual tracking for outdoor augmented reality applications. In *Proc. ISMAR '02 (Int. Symposium on Mixed and Augmented Reality)*, pages 277–278, Darmstadt, Germany.

Behringer, R., Tam, C., McGee, J., Sundareswaran, S., and Vassiliou, M. (2000). A wearable augmented reality testbed for navigation and control, built solely with Commercial-Off-The-Shelf (COTS) hardware. In *Proc. ISAR '00 (Int. Symposium on Augmented Reality)*, pages 12–19, Munich, Germany.

Bell, B. and Feiner, S. (2000). Dynamic space management for user interfaces. In *Proc. ACM UIST 2000 (Symp. on User Interface Software and Technology)*, pages 239–248, San Diego, CA.

Bell, B., Feiner, S., and Höllerer, T. (2001). View management for virtual and augmented reality. In *Proc. ACM UIST 2001 (Symp. on User Interface Software and Technology)*, pages 101–110, Orlando, FL. (CHI Letters, vol. 3, no. 2).

Bell, B., Feiner, S., and Höllerer, T. (2002a). Visualization viewpoints: Information at a glance. *IEEE Computer Graphics and Applications*, 22(4):6–9.

Bell, B., Höllerer, T., and Feiner, S. (2002b). An annotated situation-awareness aid for augmented reality. In *Proc. ACM UIST 2002 (Symp. on User Interface Software and Technology)*, pages 213–216, Paris, France.

Benford, S. and Fahlén, L. (1993). A Spatial Model of Interaction in Large Virtual Environments. In *Proceedings of ECSCW '93*, Milan, Italy.

Bernsen, N. O. (1994). Foundations of multimodal representations: A taxonomy of representational modalities. *Interacting with Computers*, 6(4):347–371.

Bertin, J. (1983). *Semiology of graphics*. W. Berg, transl., University of Wisconsin Press, Madison, Wisconsin. (Orig. pub. in French, 1967).

Billinghurst, M., Bowskill, J., Dyer, N., and Morphett, J. (1998a). An evaluation of wearable information spaces. In *Proc. VRAIS '98 (IEEE Virtual Reality Annual International Symposium)*, pages 20–27.

Billinghurst, M., Bowskill, J., Dyer, N., and Morphett, J. (1998b). Spatial information displays on a wearable computer. *IEEE Computer Graphics and Applications*, 18(6):24–31.

Billinghurst, M., Bowskill, J., Jessop, M., and Morphett, J. (1998c). A wearable spatial conferencing space. In *Proc. ISWC '98 (Second Int. Symposium on Wearable Computers)*, pages 76–83.

Billinghurst, M., Kato, H., and Poupyrev, I. (2001). The MagicBook: a transitional AR interface. *Computers and Graphics*, 25(5):745–753.

Billinghurst, M., Weghorst, S., and Furness, T. (1997). Wearable computers for three dimensional CSCW. In *Proc. ISWC '97 (First Int. Symp. on Wearable Computers)*, pages 39–46, Cambridge, MA.

Billinghurst, M., Weghorst, S., and Furness III, T. A. (1998d). Shared space: An augmented reality approach for computer supported collaborative work. *Virtual Reality*, 3(1):25–36.

Blaskó, G. and Feiner, S. (2002). A menu interface for wearable computing. In *Proc. ISWC '02 (Sixth Int. Symp. on Wearable Computers)*, pages 164–165, Seattle, WA.

Blattner, M. M. and Glinert, E. P. (1996). Multimodal integration. *IEEE MultiMedia*, 3(4):14–24.

Bowman, D. A. (1999). *Interaction Techniques for Common Tasks in Immersive Virtual Environments*. PhD thesis, Georgia Institute of Technology, Atlanta, GA.

Bowman, D. A. and Hodges, L. F. (1997). An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *1997 Symposium on Interactive 3D Graphics*, pages 35–38, Providence, RI.

Broll, W., Schäfer, L., Höllerer, T., and Bowman, D. (2001). Interface with angels: The future of VR and AR interfaces. *IEEE Computer Graphics and Applications*, 21(6):14–17.

Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S. (2000). Easyliving: Technologies for intelligent environments. In *Proc. Second Int. Symp. on Handheld and Ubiquitous Computing (HUC 2000)*, pages 12–29, Bristol, UK. Springer Verlag.

Buchanan, B. G. and Shortliffe, E. H., editors (1985). *Rule-based Expert Systems: The MYCIN experience of the Stanford Heuristic Programming Project*. Addison-Wesley (Reading MA).

Buchanan, B. G., Sutherland, G. L., and Feigenbaum, E. A. (1969). Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry. In Meltzer, B., Michie, D., and Swann, M., editors, *Machine Intelligence 4*, pages 209–254. Edinburgh University Press, Edinburgh, Scotland.

Butz, A., Beshers, C., and Feiner, S. (1998). Of vampire mirrors and privacy lamps: Privacy management in multi-user augmented environments [technote]. In *Proc. UIST'98*. ACM SIGGRAPH.

Butz, A., Höllerer, T., Feiner, S., MacIntyre, B., and Beshers, C. (1999). Enveloping users and computers in a collaborative 3D augmented reality. In *Proc. IWAR '99 (Int. Workshop on Augmented Reality)*, pages 35–44, San Francisco, CA.

Buxton, W. (1986). Chunking and phrasing and the design of human-computer dialogues. In Kugler, H. J., editor, *Information Processing '86, Proc. of the IFIP 10th World Computer Congress*. North Holland Publishers, Amsterdam.

Card, S. K., Mackinlay, J. D., and Robertson, G. G. (1991). A morphological analysis of the design space of input devices. *ACM Transactions on Information Systems*, 9(2):99–122. Special Issue on Computer-Human Interaction.

Cardelli, L. (1995). A language with distributed scope. *Computing Systems*, 8(1):27–59.

Carlsson, C. and Hagsand, O. (1993). DIVE — A platform for multi-user virtual environments. *Computers and Graphics*, 17(6):663–669.

Castro, P., Chiu, P., Kremenek, T., and Muntz, R. (2001). A probabilistic room location service for wireless networked environments. In *Proc. ACM UbiComp 2001: Ubiquitous Computing*, volume 2201 of *Lecture Notes in Computer Science*, pages 18–35.

Caudell, T. P. and Mizell, D. W. (1992). Augmented reality: An application of heads-up display technology to manual manufacturing processes. In *Proceedings of 1992 IEEE Hawaii International Conference on Systems Sciences*. IEEE Press.

Cheverst, K., Davies, N., Mitchell, K., and Blair, G. S. (2000). Developing a Context-aware Electronic Tourist Guide: Some Issues and Experiences. In *Proceedings of CHI' 00*, Netherlands.

Chia, K. W., Cheok, A. D., and Prince, S. J. D. (2002). Online 6DOF augmented reality registration from natural features. In *Proc. ISMAR '02 (Int. Symposium on Mixed and Augmented Reality)*, pages 305–313, Darmstadt, Germany.

Clarkson, B., Mase, K., and Pentland, A. (2000). Recognizing user context via wearable sensors. In *Proc. ISWC '00 (Fourth Int. Symp. on Wearable Computers)*, pages 69–75, Atlanta, GA.

Codella, C., Jalili, R., Koved, L., and Lewis, B. (1993). A toolkit for developing multi-user, distributed virtual environments. *Proceedings of VRAIS'93*, pages 401–407.

Cohen, P., Johnston, M., McGee, D., Orviatt, S., Pittman, J., Smith, I., Chen, L., and Clow, J. (1998). QuickSet: Multimodal interaction for distributed applications. In *Proceedings of The Fifth ACM International Multimedia Conference (MULTIMEDIA '97)*, pages 31–40, New York/Reading. ACM Press/Addison-Wesley.

Cometa Networks (2002). Cometa networks wholesale nationwide broadband wireless internet access. http://www.cometanetworks.com.

Conner, D. B., Snibbe, S. S., Herndon, K. P., Robbins, D. C., Zeleznik, R. C., and van Dam, A. (1992). Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics, Special Issue of Computer Graphics, Vol. 26*, pages 183–188.

Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. (1993). Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Computer Graphics (Proc. ACM SIGGRAPH '93)*, Annual Conference Series, pages 135–142.

Culbert, C., Riley, G., and Stavely, R. T. (1987). An expert system development methodology which supports verification and validation. In *Proceedings of the Fourth IEEE Conference on Artificial Intelligence Applications*, Houston , TX , USA.

Dähne, P. and Karigiannis, J. N. (2002). Archeoguide: System architecture of a mobile outdoor augmented reality system. In *Proc. ISMAR '02 (Int. Symposium on Mixed and Augmented Reality)*, pages 208–209, Darmstadt, Germany.

Darken, R. and Cevik, H. (1999). Map usage in virtual environments: Orientation issues. In *Proceedings of IEEE VR '99*, pages 133–140.

Darken, R. P. and Sibert, J. L. (1993). A toolset for navigation in virtual environments. In *Proc. UIST '93 (Sixth Annual ACM Symp. on User Interface Software and Technology)*, pages 157–166, New York, NY, USA. ACM Press.

Deering, M. and Sowizral, H. (1997). *Java3D Specification, Version 1.0*. Sun Microsystems, 2550 Garcia Avenue, Mountain View, CA 94043, USA.

Degen, L., Mander, R., and Salomon, G. (1992). Working with audio: Integrating personal tape recorders and desktop computers. In *Proceedings of the CHI 1992 Conference on Human Factors in Computing Systems (CHI-92)*, pages 413–418. ACM Press.

Ekahau (2002). Ekahau – location in wireless networks. http://www.ekahau.com.

Ericsson, IBM, Intel, Nokia, and Toshiba (1998). Bluetooth mobile wireless initiative. http://www.bluetooth.com.

European Commission, Energy and Transport (2002). The Galilei project, Galileo design consolidation. http://europa.eu.int/comm/dgs/energy_transport/galileo/index_en.htm.

Farringdon, J., Moore, A. J., Tilbury, N., Church, J., and Biemond, P. D. (1999). Wearable sensor badge and sensor jacket for context awareness. In *Proc. ISWC '99 (Third Int. Symp. on Wearable Computers)*, pages 107–113, San Francisco, CA.

Feiner, S. (1985). APEX: an experiment in the automated creation of pictorial explanations. *IEEE Computer Graphics and Applications*, 5(11):29–37.

Feiner, S. (2002). Augmented Reality: A new way of seeing. *Scientific American*, 286(4):48–55.

Feiner, S., MacIntyre, B., Haupt, M., and Solomon, E. (1993a). Windows on the world: 2D windows for 3D augmented reality. In *Proc. UIST '93 (ACM Symp. on User Interface Software and Technology)*, pages 145–155, Atlanta, GA.

Feiner, S., MacIntyre, B., and Höllerer, T. (1999). Wearing it out: First steps toward mobile augmented reality systems. In Ohta, Y. and Tamura, H., editors, *Mixed Reality: Merging Real and Virtual Worlds*, pages 363–377. Ohmsha (Tokyo)–Springer Verlag, Berlin.

Feiner, S., MacIntyre, B., Höllerer, T., and Webster, A. (1997). A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. In *Proc. ISWC '97 (First Int. Symp. on Wearable Computers)*, pages 74–81, Cambridge, MA.

Feiner, S., MacIntyre, B., and Seligmann, D. (July 1993b). Knowledge-based augmented reality. *Communications of the ACM*, 36(7):52–62.

Feiner, S. and Seligmann, D. (1992). Cutaways and ghosting: Satisfying visibility constraints in dynamic 3D illustrations. *The Visual Computer*, 8(5–6):292–302.

Feiner, S. and Shamash, A. (1991). Hybrid user interfaces: Breeding virtually bigger interfaces for physically smaller computers. In *Proc. UIST '91*, pages 9–17. ACM press.

Foley, J. D., Wallace, V. L., and Chan, P. (1984). The human factors of computer graphics interaction techniques. *IEEE Computer Graphics and Applications*, 4(11):13–48.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37.

Foxlin, E. and Harrington, M. (2000). Weartrack: A self-referenced head and hand tracker for wearable computers and portable vr. In *Proc. ISWC '00 (Fourth Int. Symp. on Wearable Computers)*, pages 155–162, Atlanta, GA.

Friedman-Hill, E. J. (1998). Jess, the expert system shell for the java platform. Technical Report 98-8206, Sandia National Laboratories, Livermore, CA.

Fuchs, H., Livingston, M., Raskar, R., Colucci, D., Keller, K., State, A., Crawford, J., Rademacher, P., Drake, S., and Meyer, A. (1998). Augmented Reality Visualization for Laparoscopic Surgery. In *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*.

Furht, B. (1994). Multimedia systems: An overview. *IEEE MultiMedia*, 1(1):47–59.

Furmanski, C., Azuma, R., and Daily, M. (2002). Augmented-reality visualizations guided by cognition: Perceptual heuristics for combining visible and obscured information. In *Proc. ISMAR '02 (Int. Symposium on Mixed and Augmented Reality)*, pages 215–224, Darmstadt, Germany.

Furness, T. (1986). The super cockpit and its human factors challenges. In *Proc. Human Factors Society 30th Annual Meeting*, pages 48–52, Santa Monica, CA.

Genc, Y., Riedel, S., Souvannavong, F., Akinlar, C., and Navab, N. (2002). Marker-less tracking for AR: A learning-based approach. In *Proc. ISMAR '02 (Int. Symposium on Mixed and Augmented Reality)*, pages 295–304, Darmstadt, Germany.

Getting, I. (1993). The global positioning system. *IEEE Spectrum*, 30(12):36–47.

Giguère, E. (1997). Java Beans and the new event model. *Dr. Dobb's Journal of Software Tools*, 22(4):50, 52, 53, 79, 80.

Global Locate (2002). Global Locate/Fujitsu GL-16000 Indoor GPS chip. http://www.globallocate.com/.

GML (2003). *OpenGIS Geography Markup Language (GML) Implementation Specification, Version 3.00*. Open GIS Consortium, Inc. http://www.opengis.org/docs/02-023r4.pdf.

Golding, A. R. and Lesh, N. (1999). Indoor navigation using a diverse set of cheap, wearable sensors. In *Proc. ISWC '99 (Third Int. Symp. on Wearable Computers)*, pages 29–36, San Francisco, CA.

Gosling, J., Joy, B., and Steele, G. (1996). *The Java Language Specification*. Addison-Wesley, Reading, MA, first edition.

Gosling, J., Joy, B., Steele, G., and Bracha, G. (2000). *The Java Language Specification*. Addison-Wesley, Reading, MA, second edition.

Grandtec (2001). Pocketvik, the virtually indestructible keyboard for pda. http://www.grandtec.com/pocketvik.htm.

Green, M. (1987). Directions for user interface management systems research. *ACM Computer Graphics*, 21(2):113–116.

Green, M. (1990). Virtual reality user interface: tools and techniques. In Chua, T.-S. and Kunii, T. L., editors, *CG International '90, Computer graphics around the world. Proceedings.*, pages 51–68, Singapore (Singapore). Springer Verlag.

Hallaway, D., Höllerer, T., and Feiner, S. (2003). Coarse, inexpensive, infrared tracking for wearable computing. In *Proc. ISWC '03 (Seventh Int. Symp. on Wearable Computers)*, pages 69–78, White Plains, NY.

Hallaway, D., Höllerer, T., and Feiner, S. (2004). Bridging the gaps: Hybrid tracking for adaptive mobile augmented reality. To appear in: Applied Artificial Intelligence Journal, Special Issue on AI in Mobile Systems.

Handykey (2001). Handykey twiddler2 chord keyboard. http://www.handykey.com.

Harbison, S. P. (1992). *Modula-3*. Prentice-Hall.

Hasvold, P. (2002). In-the-Field Health Informatics. In *The Open Group conference*, Paris, France. http://www.opengroup.org/public/member/q202/documentation/plenary/hasvold.pdf.

Heilig, M. L. (1960). Stereoscopic-television apparatus for individual use. *United States Patent Number 2,955,156*. Filed May 24, 1957. Also reprinted in Computer Graphics, 28 (2), May 1994, pages 131–134.

Heilig, M. L. (1992). El cine del futuro: The cinema of the future. *Presence*, 1(3):279–294.

Heller, R. S. and Martin, C. D. (1995). A media taxonomy. *IEEE MultiMedia*, 2(4):36–45.

Heller, R. S., Martin, C. D., Haneef, N., and Gievska-Krliu, S. (2001). Using a theoretical multimedia taxonomy framework. *Journal of Educational Resources in Computing*, 1(1es):22.

Hinckley, K., Pausch, R., Goble, J. C., and Kassell, N. F. (1994a). A survey of design issues in spatial input. In *ACM UIST'94 Symp. on User Interface Software & Technology*, pages 213–222.

Hinckley, K., Pausch, R., Goble, J. C., and Kassell, N. F. (1994b). A survey of design issues in spatial input. In *Proc. ACM UIST'94 (Symp. on User Interface Software & Technology)*, pages 213–222.

Höllerer, T., Feiner, S., Hallaway, D., Bell, B., Lanzagorta, M., Brown, D., Julier, S., Baillot, Y., and Rosenblum, L. (2001a). User interface management techniques for collaborative mobile augmented reality. *Computers and Graphics*, 25(5):799–810.

Höllerer, T., Feiner, S., and Pavlik, J. (1999a). Situated documentaries: Embedding multimedia presentations in the real world. In *Proc. ISWC '99 (Third Int. Symp. on Wearable Computers)*, pages 79–86, San Francisco, CA.

Höllerer, T., Feiner, S., Terauchi, T., Rashid, G., and Hallaway, D. (1999b). Exploring MARS: Developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers and Graphics*, 23(6):779–785.

Höllerer, T., Hallaway, D., Tinna, N., and Feiner, S. (2001b). Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In *2nd Int. Workshop on Artificial Intelligence in Mobile Systems (AIMS '01)*, pages 31–37.

Hopkins, D. (1987). Directional selection is easy as pie menus! *login: The Usenix Association Newsletter*, 12(5).

Hua, H., Gao, C., Biocca, F., and Rolland, J. P. (2001). An ultra-light and compact design and implementation of head-mounted projective displays. In *Proceedings of IEEE VR '01*, pages 175–182, Yokohama, Japan.

Hull, R., Neaves, P., and Bedford-Roberts, J. (1997). Towards Situated Computing. In *Proc. ISWC '97 (First Int. Symp. on Wearable Computers)*, pages 146–153, Cambridge, MA.

InterSense (2001). InterSense Inc., IS-900 Wide Area Precision Motion Tracker, IS300 and InertiaCube2 orientation sensors. http://www.isense.com.

Ioannidis, J., Duchamp, D., and Maguire, G. (1991). IP-based protocols for mobile internetworking. In *Proc. SIGCOMM '91*, pages 235–245. ACM.

Ishii, H. and Ullmer, B. (1997). Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proc. Conference on Human Factors in Computing Systems (CHI '97), (Atlanta, March 1997)*, pages 234–241. ACM Press.

Jacob, R. J. (1991). The use of eye movements in human-computer interaction techniques. *ACM Transactions on Information Systems*, 9(3):152–169.

Jacob, R. J. K., Deligiannidis, L., and Morrison, S. (1999). A software model and specification language for non-WIMP user interfaces. *ACM SIGCHI Bulletin*, 6(1):1–46.

Jang, B., Kim, J., Kim, H., and Kim, D. (1999). An outdoor augmented reality system for GIS applications. In Ohta, Y. and Tamura, H., editors, *Mixed Reality, Merging Real and Virtual Worlds*, pages 391–399. Ohmsha/Springer, Tokyo/New York.

Julier, S., Baillot, Y., Lanzagorta, M., Brown, D., and Rosenblum, L. (2000a). BARS: Battlefield Augmented Reality System. In *NATO Symposium on Information Processing Techniques for Military Systems*, Istanbul, Turkey.

Julier, S. and Bishop, G., editors (2002). *IEEE Computer Graphics and Applications, Special Issue on Tracking*, volume 6(22), pages 22–80. IEEE Computer Society.

Julier, S., Lanzagorta, M., Baillot, Y., Rosenblum, L., Feiner, S., Höllerer, T., and Sestito, S. (2000b). Information filtering for mobile augmented reality. In *Proc. ISAR '00 (Int. Symposium on Augmented Reality)*, pages 3–11, Munich, Germany.

Kamada, T. and Kawai, S. (1987). An enhanced treatment of hidden lines. *ACM Transactions on Graphics*, 6(4):308–323.

Kasai, I., Tanijiri, Y., Endo, T., and Ueda, H. (2000). A forgettable near eye display. In *Proc. ISWC '00 (Fourth Int. Symp. on Wearable Computers)*, pages 115–118, Atlanta, GA.

Kasik, D. J. (1982). A user interface management system. *Computer Graphics (SIGGRAPH '82 Proceedings)*, 16(3):99–106.

Klinker, G., Creighton, O., Dutoit, A. H., Kobylinski, R., Vilsmeier, C., and Brügge, B. (2001). Augmented maintenance of powerplants: a prototyping case study of a mobile AR system. In *Proc. ISAR '01 (Int. Symposium on Augmented Reality)*, pages 124–133, New York, NY.

Kobsa, A. (1990). User modeling in dialog systems: Potentials and hazards. *AI and Society*, 4(3):214–240.

Koller, D., Klinker, G., Rose, E., Breen, D., Whitaker, R., and Tuceryan, M. (1997). Real-time Vision-Based camera tracking for augmented reality applications. In Thalmann, D., editor, *ACM Symposium on Virtual Reality Software and Technology*, New York, NY. ACM, ACM Press.

Kristoffersen, S. and Ljungberg, F. (1999). Designing interaction styles for a mobile use context. *Lecture Notes in Computer Science*, 1707:281–288.

Krueger, W. and Froehlich, B. (1994). The responsive workbench. *IEEE Computer Graphics and Applications*, 14(3):12–15.

Kurata, T., Okuma, T., Kourogi, M., Kato, T., and Sakaue, K. (2001). VizWear: Toward human-centered interaction through wearable vision and visualization. *Lecture Notes in Computer Science*, 2195:40–47.

Langley, R. (1997). Glonass: Review and update. *GPS World*, pages 46–51.

Lee, J. W., You, S., and Neumann, U. (2002). Tracking with omni-directional vision for outdoor AR systems. In *Proc. ISMAR '02 (Int. Symposium on Mixed and Augmented Reality)*, pages 47–56, Darmstadt, Germany.

Lee, S.-W. and Mase, K. (2001). A personal indoor navigation system using wearable sensors. In *Proc. ISMR '01 (Second Int. Symp. on Mixed Reality)*, pages 147–148, Yokohama, Japan.

Licklider, J. C. R. (1960). Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1:4–11.

Livingston, M. A., Swan II, J. E., Gabbard, J. L., Höllerer, T. H., Hix, D., Julier, S. J., Baillot, Y., and Brown, D. (2003). Resolving multiple occluded layers in augmented reality. In *Proc. ISMAR '03 (Int. Symposium on Mixed and Augmented Reality)*, pages 56–65, Tokyo, Japan.

Lok, S. and Feiner, S. (2001). A survey of automated layout techniques for information presentations. In *Proc. Smart Graphics 2001 (First Int. Symp. on Smart Graphics)*, pages 61–68, Hawthorne, NY.

Loomis, J., Golledge, R., and Klatzky, R. (1993). Personal guidance system for the visually impaired using GPS, GIS, and VR technologies. In *Proc. Conf. on Virtual Reality and Persons with Disabilities*, Millbrae, CA.

Loomis, J. M., Golledge, R. G., and Klatzky, R. L. (1998). Navigation system for the blind: Auditory display modes and guidance. *Presence: Teleoperators and Virtual Environments*, 7(2):193–203.

Luff, P. and Heath, C. (1998). Mobility in collaboration. In *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work*, From Single-Display Groupware to Mobility, pages 305–314.

MacIntyre, B. (1997). Repo: Obliq with replicated objects. Programmers guide and reference manual. Technical report, Columbia University, Department of Computer Science.

MacIntyre, B. (1998). *Exploratory Programming of Distributed Augmented Environments*. PhD thesis, Columbia University, New York, NY.

MacIntyre, B. and Feiner, S. (1996a). Future Multimedia User Interfaces. *Multimedia Systems*, 4(5):250–268.

MacIntyre, B. and Feiner, S. (1996b). Language-level support for exploratory programming of distributed virtual environments. In *Proc. UIST '96*, pages 83–94, Seattle, WA.

MacIntyre, B. and Feiner, S. (1998). A distributed 3D graphics library. In *Computer Graphics (Proc. ACM SIGGRAPH '98)*, Annual Conference Series, pages 361–370, Orlando, FL.

Mackinlay, J. D. (1986). *Automatic design of graphical presentations*. PhD thesis, Computer Science Department, University of Stanford, Stanford, CA.

Mann, S. (1997). Wearable computing: A first step toward personal imaging. *IEEE Computer*, 30(2):25–32.

Mann, S. (1998). Humanistic intelligence: Wearcomp as a new framework for intelligent signal processing. *Proceedings of the IEEE*, 86(11):2123–2151.

Marcus, A. (1992). A comparison of graphical user interfaces. In Marcus, A., editor, *Graphic Design for Electronic Documents and User Interfaces*, pages 137–188. Addison-Wesley Publishing ACM Press, Reading, MA.

May, M. (2003). Type it anywhere. *Scientific American*, 288(1). Innovations.

Milgram, P. and Colquhoun Jr., H. (1999). A taxonomy of real and virtual world display integration. In Ohta, Y. and Tamura, H., editors, *Mixed Reality, Merging Real and Virtual Worlds*, pages 1–26. Ohmsha/Springer, Tokyo/New York.

Milgram, P. and Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE Transactions on Information Systems*, E77-D(12):1321–1329.

Mine, M. R., Brooks Jr., F. P., and Sequin, C. H. (1997). Moving objects in space: Exploiting proprioception in virtual-environment interaction. pages 19–26.

Mizell, D. (2001). Boeing's wire bundle assembly project. In Barfield, W. and Caudell, T., editors, *Fundamentals of Wearable Computers and Augmented Reality*, pages 447–467. Lawrence Erlbaum Assoc, Mahwah, NJ.

MRML (2002). Multimedia retrieval markup language. http://www.mrml.net/.

Myers, B. A. (1988). A taxonomy of window manager user interfaces. *IEEE Computer Graphics and Applications*, 8(5):65–84.

Mynatt, E., Back, M., Want, R., and Frederick, R. (1997). Audio aura: Light-weight audio augmented reality. In *Proc. UIST '97*, pages 211–212. ACM Press.

Najork, M. A. and Brown, M. H. (1995). Obliq-3D: A high-level, fast-turnaround 3D animation system. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):175–145.

Nayar, S. (1997). Catadioptric omnidirectional camera. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '97)*, pages 482–488.

Nielsen, J. (1993). Noncommand user interfaces. *Communications of the ACM*, 36(4):82–99.

Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufmann.

Nusser, S. M., Miller, L. L., Goodchild, M. F., and Clarke, K. (2001). Future views of field data collection in statistical surveys. In *Proceedings of dg.o 2001 National Conference on Digital Government Research*, Los Angeles, CA. http://www.isi.edu/dgrc/dgo2001/papers/session-3/nusser1.pdf.

Olsen, Jr., D. R. (1992). *User Interface Management Systems: Models and Algorithms*. Morgan Kaufman Publishers Inc.

Olsen, Jr., D. R., Green, M., Lantz, K. A., Schulert, A., and Sibert, J. L. (1987). Whither (or wither) UIMS? In *Proceedings of ACM CHI+GI'87 Conference on Human Factors in Computing Systems and Graphics Interface*, Panel, pages 311–314.

Pascoe, J., Ryan, N., and Morse, D. (2000). Using while moving: HCI issues in fieldwork environments. *ACM Transactions on Computer-Human Interaction*, 7(3):417–437.

Pasman, W. and Jansen, F. W. (2001). Distributed low-latency rendering for mobile AR. In *Proc. ISAR '01 (Int. Symposium on Augmented Reality)*, pages 107–113, New York, NY.

Pausch, R. (1991). Virtual reality on five dollars a day. *Proc. ACM CHI '91 Conference on Human Factors in Computing Systems*, pages 265–270.

Pausch, R., Burnette, T., Brockway, D., and Weiblen, M. (1995). Navigation and locomotion in virtual worlds via flight into hand-held miniatures. *Proc. SIGGRAPH '95*, pages 399–401.

Pavlik, J. V. (2001). *Journalism and New Media*. Columbia University Press, New York, NY.

Petrie, H., Johnson, V., Strothotte, T., Raab, A., Fritz, S., and Michel, R. (1996). MoBIC: Designing a travel aid for blind and elderly people. *Jnl. of Navigation*, 49(1):45–52.

Picard, R. W. (1997). *Affective Computing*. MIT Press, Cambridge, MA.

Piekarski, W., Gunther, B., and Thomas, B. (1999). Integrating virtual and augmented realities in an outdoor application. In *Proc. IWAR '99 (Int. Workshop on Augmented Reality)*, pages 45–54, San Francisco, CA.

Piekarski, W. and Thomas, B. (2001a). Tinmith-evo5 – an architecture for supporting mobile augmented reality environments. In *Proc. ISAR '01 (Int. Symposium on Augmented Reality)*, pages 177–178, New York, NY.

Piekarski, W. and Thomas, B. (2001b). Tinmith-Metro: New outdoor techniques for creating city models with an augmented reality wearable computer. In *Proc. ISWC '01 (Fifth Int. Symp. on Wearable Computers)*, pages 31–38, Zürich, Switzerland.

Pierce, J. S., Conway, M., van Dantzich, M., and Robertson, G. (1999). Tool spaces and glances: Storing. accessing, and retrieving objects in 3D desktop applications. In Spencer, S. N., editor, *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, pages 163–168, New York. ACM Press.

Pierce, J. S., Forsberg, A., Conway, M. J., Hong, S., Zeleznik, R., and Mine, M. R. (1997). Image plane interaction techniques in 3D immersive environments. In *Symposium on Interactive 3D Graphics*. ACM SIGGRAPH.

Pisacane, V., editor (1998). *The Legacy of Transit*, volume 1(19) of *JHU APL Technical Digest*. Johns Hopkins University Applied Physics Laboratory. http://techdigest.jhuapl.edu/td1901/index.htm.

Poupyrev, I., Billinghurst, M., Weghorst, S., and Ichikawa, T. (1996). The go-go interaction technique: Non-linear mapping for direct manipulation in VR. In *Proc. UIST '96 (ACM Symposium on User Interface Software and Technology)*, pages 79–80.

Poupyrev, I., Maruyama, S., and Rekimoto, J. (2002). Ambient touch: designing tactile interfaces for handheld devices. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 51–60. ACM Press.

Pouwelse, J., Langendoen, K., and Sips, H. (1999). A feasible low-power augmented reality terminal. In *Proc. IWAR '99 (Int. Workshop on Augmented Reality)*, pages 55–63, San Francisco, CA.

Raab, F. H., Blood, E. B., Steiner, T. O., and Jones, R. J. (1979). Magnetic position and orientation tracking system. *IEEE Trans. on Aerospace and Electronic Systems*, AES-15(5):709–718.

Raskar, R., Welch, G., Cutts, M., Lake, A., Stesin, L., and Fuchs, H. (1998). The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proc. SIGGRAPH '98*, pages 179–188.

Reitmayr, G. and Schmalstieg, D. (2001a). Mobile collaborative augmented reality. In *Proc. ISAR '01 (Int. Symposium on Augmented Reality)*, pages 114–123, New York, NY.

Reitmayr, G. and Schmalstieg, D. (2001b). A wearable 3D augmented reality workspace. In *Proc. ISWC '01 (Fifth Int. Symp. on Wearable Computers)*, pages 165–166, Zürich, Switzerland.

Reitmayr, G. and Schmalstieg, D. (2003). Location based applications for mobile augmented reality. In Biddle, R. and Thomas, B., editors, *Fourth Australasian User Interface Conference (AUIC2003)*, volume 18 of *Conferences in Research and Practice in Information Technology*, page 65, Adelaide, Australia. ACS.

Rekimoto, J., Ayatsuka, Y., and Hayashi, K. (1998). Augment-able reality: Situated communication through physical and digital spaces. In *Proc. ISWC '98 (Second Int. Symp. on Wearable Computers)*, pages 68–75, Cambridge, MA.

Rekimoto, J. and Nagao, K. (1995). The world through the computer: Computer augmented interaction with real world environments. In *Proc. ACM Symposium on User Interface Software and Technology*, Virtual and Augmented Realities, pages 29–36.

Rhodes, B. J. (1997). The wearable remembrance agent: A system for augmented memory. In *Proc. ISWC '97 (First Int. Symp. on Wearable Computers)*, pages 123–128, Cambridge, Mass., USA.

Rolland, J. P., Davis, L. D., and Baillot, Y. (2001). A survey of tracking technologies for virtual environments. In Barfield, W. and Caudell, T., editors, *Fundamentals of Wearable Computers and Augmented Reality*. Lawrence Erlbaum Assoc, Mahwah, NJ.

Ross, D. A. and Blasch, B. B. (2000). Wearable interfaces for orientation and wayfinding. In *Fourth Annual ACM Conference on Assistive Technologies*, pages 193–200. ACM.

Roth, S. F., Kolojejchick, J., Mattis, J., and Goldstein, J. (1994). Interactive graphic design using automatic presentation knowledge. In *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1 of *Active Support for Interaction*, pages 112–117.

Roth, S. F. and Mattis, J. (1990). Data characterization for intelligent graphics presentation. In *Proceedings of ACM CHI'90 Conference on Human Factors in Computing Systems*, End User Modifiable Environment, pages 193–200.

Roy, D. K. and Schmandt, C. (1996). Newscomm: A hand-held interface for interactive access to structured audio. In *Proc. CHI'96 (ACM Conf. on Human Factors in Computing Systems*, pages 173–180. ACM Press.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Pearson Education Inc., Upper Saddle River, NJ, second edition. Pages 320–374.

Sato, K., Ban, Y., and Chihara, K. (1999). MR aided engineering: Inspection support systems integrating virtual instruments and process control. In Ohta, Y. and Tamura, H., editors, *Mixed Reality, Merging Real and Virtual Worlds*, pages 347–361. Ohmsha/Springer, Tokyo/New York.

Satoh, K., Anabuki, M., Yamamoto, H., and Tamura, H. (2001a). A hybrid registration method for outdoor augmented reality. In *Proc. ISAR '01 (Int. Symposium on Augmented Reality)*, pages 67–76, New York, NY.

Satoh, K., Hara, K., Anabuki, M., Yamamoto, H., and Tamura, H. (2001b). TOWNWEAR: An outdoor wearable MR system with high-precision registration. In *Proc. ISMR '01 (Second Int. Symp. on Mixed Reality)*, pages 210–211, Yokohama, Japan.

Sawhney, N. and Schmandt, C. (1998). Speaking and listening on the run: Design for wearable audio computing. In *Proc. ISWC '98 (Second Int. Symp. on Wearable Computers)*, pages 108–115, Cambridge, MA.

Schmidt, A., Gellersen, H.-W., Beigl, M., and Thade, O. (2000). Developing user interfaces for wearable computers—don't stop to point and click. In *Intelligent Interactive Assistance and Mobile Multimedia Computing (IMC'2000)*, Rostock-Warnemünde, Germany.

Schnädelbach, H., Koleva, B., Flintham, M., Fraser, M., Izadi, S., Chandler, P., Foster, M., Benford, S., Greenhalgh, C., and Rodden, T. (2002). The Augurscope: A mixed reality interface for outdoors. In Terveen, L., Wixon, D., Comstock, E., and Sasse, A., editors, *Proceedings of the CHI 2002 Conference on Human Factors in Computing Systems (CHI-02)*, pages 9–16, New York. ACM Press.

Seligmann, D. (1993). *Interactive Intent-Based Illustration: A Visual Language for 3D Worlds*. PhD thesis, Columbia University, New York, NY.

Seligmann, D. and Feiner, S. (1991). Automated generation of intent-based 3D-illustrations. *ACM SIGGRAPH '91, Computer Graphics*, 25(4):23–32.

Shneiderman, B. (1998). *Designing the User Interface*. Addison-Wesley, third edition.

Simon, G. and Berger, M.-O. (2002). Reconstructing while registering: A novel approach for markerless augmented reality. In *Proc. ISMAR '02 (Int. Symposium on Mixed and Augmented Reality)*, pages 285–294, Darmstadt, Germany.

Sowizral, H., Rushforth, K., and Deering, M. (1997). *The Java 3D API Specification*. Addison-Wesley.

Spohrer, J. (1997). Worldboard — what comes after the WWW? http://www.worldboard.org/pub/spohrer/wbconcept/default.html.

Spohrer, J. (1999). Information in places. *IBM Systems Journal*, 38(4):602–628.

Starner, T., Leibe, B., Singletary, B., and Pair, J. (2000). MIND-WARPING: Towards creating a compelling collaborative augmented reality game. In *Proc. Int. Conf. on Intelligent User Interfaces (IUI '00)*, pages 256–259.

Starner, T., Mann, S., Rhodes, B., Levine, J., Healey, J., Kirsch, D., Picard, R., and Pentland, A. (1997a). Augmented reality through wearable computing. *Presence*, 6(4):386–398.

Starner, T., Weaver, J., and Pentland, A. (1997b). A wearable computing based american sign language recognizer. In *Proc. ISWC '97 (First Int. Symp. on Wearable Computers)*, pages 130–137, Cambridge, MA.

Stoakley, R., Conway, M., and Pausch, R. (1995). Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings of Human Factors in Computing Systems (CHI '95)*, pages 265–272.

Suomela, R., Lehikoinen, J., and Salminen, I. (2001). A system for evaluating augmented reality user interfaces in wearable computers. In *Proc. ISWC '01 (Fifth Int. Symp. on Wearable Computers)*, pages 77–84, Zürich, Switzerland.

Suomela, R. and Lehikonen, J. (2000). Context compass. In *Proc. ISWC '00 (Fourth Int. Symp. on Wearable Computers)*, pages 147–154, Atlanta, GA.

Sutherland, I. (1968). A head-mounted three dimensional display. In *Proc. FJCC 1968*, pages 757–764, Washington, DC. Thompson Books.

Szalavari, Z., Eckstein, E., and Gervautz, M. (1998). Collaborative gaming in augmented reality. In *Proc. VRST '98*, pages 195–204, Taipei, Taiwan.

Tamura, H., Yamamoto, H., and Katayama, A. (2001). Mixed reality: Future dreams seen at the border between real and virtual worlds. *IEEE Computer Graphics and Applications*, 21(6):64–70.

Tan, H. Z. and Pentland, A. (2001). Tactual displays for sensory substitution and wearable computers. In Barfield, W. and Caudell, T., editors, *Fundamentals of Wearable Computers and Augmented Reality*, pages 579–598. Lawrence Erlbaum Assoc, Mahwah, NJ.

Tappert, C. C., Ruocco, A. S., Langdorf, K. A., Mabry, F. J., Heineman, K. J., Brick, T. A., Cross, D. M., and Pellissier, S. V. (2001). Military applications of wearable computers and augmented reality. In Barfield, W. and Caudell, T., editors, *Fundamentals of Wearable Computers and Augmented Reality*, pages 625–647. Lawrence Erlbaum Assoc, Mahwah, NJ.

Thomas, B., Close, B., Donoghue, J., Squires, J., De Bondi, P., Morris, M., and Piekarski, W. (2000). ARQuake: An outdoor/indoor augmented reality first person application. In *Proc. ISWC '00 (Fourth Int. Symp. on Wearable Computers)*, pages 139–146, Atlanta, GA.

Thomas, B., Demczuk, V., Piekarski, W., Hepworth, D., and Gunther, B. (1998). A wearable computer system with augmented reality to support terrestrial navigation. In *Proc. ISWC '98 (Second Int. Symp. on Wearable Computers)*, pages 168–171, Pittsburgh, PA.

Thomas, B. H. and Piekarski, W. (2002). Glove based user interaction techniques for augmented reality in an outdoor environment. *Virtual Reality: Research, Development, and Applications*, 6(3):167–180. Springer-Verlag London Ltd.

Thorp, E. O. (1998). The invention of the first wearable computer. In *Proc. ISWC '98 (Second Int. Symposium on Wearable Computers)*, pages 4–8.

Trimble Navigation Ltd (2002). Dual-frequency, real-time kinematic (RTK) receiver for precise dynamic positioning. http://www.trimble.com/ms750.html.

Van Laerhoven, K. and Cakmakci, O. (2000). What shall we teach our pants? In *Proc. ISWC '00 (Fourth Int. Symp. on Wearable Computers)*, pages 77–83, Atlanta, GA.

Vlahakis, V., Ioannidis, N., Karigiannis, J., Tsotros, M., Gounaris, M., Stricker, D., Gleue, T., Daehne, P., and Almaida, L. (2002). Archeoguide: An augmented reality guide for archaeological sites. *IEEE Computer Graphics and Applications*, 22(5):52–59.

Want, R., Hopper, A., Falcao, V., and Gibbons, J. (1992). The active badge location system. *ACM Trans. on Information Systems*, 10(1):91–102.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.

You, S., Neumann, U., and Azuma, R. (1999). Hybrid inertial and vision tracking for augmented reality registration. In *Proc. IEEE Virtual Reality '99*, pages 260–267, Houston, TX.

Zeleznik, R. C., LaViola, J. J., Jr., Feliz, D. A., and Keefe, D. F. (2002). Pop through button devices for ve navigation and interaction. In *Proceedings of IEEE VR '02*, pages 127–134, Orlando, FL.

Zhang, X., Genc, Y., and Navab, N. (2001). Taking AR into large scale industrial environments: Navigation and information access with mobile computers. In *Proc. ISAR '01 (Int. Symposium on Augmented Reality)*, pages 179–180, New York, NY.

Zhang, X., Navab, N., and Liou, S.-P. (2000). E-Commerce direct marketing using augmented reality. In *Proc. ICME2000 (IEEE Int. Conf. on Multimedia and Exposition)*, New York, NY.

Zhou, M. X. and Feiner, S. K. (1996). Data characterization for automatically visualizing heterogeneous information. In *Proceedings IEEE Symposium on Information Visualization*, pages 13–20. IEEE.

Zhou, M. X. and Feiner, S. K. (1997). The representation and use of a visual lexicon for automated graphics generation. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1056–1065, San Francisco, CA. Morgan Kaufmann Publishers.

Appendix A

Details of Hardware Prototypes 1996–2003

A.1 Columbia Touring Machine

In 1996/97 the author of this thesis participated in the design and testing of the Columbia Touring Machine, the first outdoor MARS allowing visual registration of computer-generated graphics with the physical world (Feiner et al., 1997). Figure A.1 shows a picture of the side and back views of this prototype. In the following paragraphs we describe the main components of the device.

Backpack Computer Having the goals of portability, easy maintainability, and ruggedness in mind, the designers of the Touring Machine chose a Fieldworks 7600 to be the MARS's main computer. This machine included a 133MHz Pentium, 64Mbyte main memory, 512K cache, a 2GB harddisk, and a card cage with expansion slots for three ISA and three PCI cards. The expansion slots allowed us to add a high-performance 3D graphics card and a sound card to the system. The Fieldworks hardware was designed to be rugged, to endure extended use in the outdoors.

Graphics Card We used a PCI-based Omnicomp 3Demon card, which was based on the Glint 500DTX chipset. It included hardware support for 3D transformations and rendering using OpenGL.

Headworn Display As discussed before in Section 2.4.2, video-based displays provide some advantages over optical see-through glasses, but restrict the resolution and field of view of the real world to that of the virtual world, forcing the limitations of mobile cameras onto the user. For this first outdoor prototype, we selected the relatively lightweight

Figure A.1: Alex Klevitzky wearing the Touring Machine (1997). (a) Side view. (b) Back view.

Virtual I/O i-glasses optical see-through head-worn display, which had a resolution of 263 $\times$ 230 (about 60,000) triads. We also experimented with a Virtual I/O 640x480 resolution greyscale display. For use in bright sunlight we added sunglasses in front of the display.

Handheld Computer and AR UI Control We used a handheld computer as an additional input/output device, displaying context-based information via dynamically created webpages, and allowing stylus-based user interaction. For this purpose we chose the Mitsubishi Amity SP, which had a 75MHz DX4 processor, 340MB disk, 16MB main memory, PCMCIA slot, 640x480 color display, and stylus operation. We allowed the user to control the headworn display menus through a Cirque GlidePoint trackpad that we mounted on the back of the handheld computer. The trackpad's x coordinates were inverted to preserve intuitive control of the menus.

Orientation Tracker For this first prototype we relied on the built-in tracking provided with the Virtual I/O i-glasses. This included a magnetometer to determine head yaw, and a two-axis inclinometer that uses gravity to detect head pitch and roll.

Position tracking We tracked the user's position using a Trimble DSM GPS receiver, obtaining position information for its antenna, which we placed on the backpack above

the user's head, as can be seen in Figure A.1. While back then regular GPS readings were only accurate within about 100 meters, differential GPS brought the position accuracy down to acceptable levels. In absence of a differential base station of our own at the time, we subscribed to a radio-based differential correction service provided by Differential Corrections Inc., which allowed us to track a user at about one-meter accuracy.

Network In order to enable communication with the rest of our infrastructure we used NCR WaveLan spread-spectrum 2Mbit/sec radio modems in both the backpack and handheld PCs (attached through PCMCIA interfaces). These radio modems operated as part of an experimental mobile IP infrastructure utilizing a skeleton network of base stations on campus (Ioannidis et al., 1991).

Power The backpack computer and its expansion cards needed more power than the standard battery for the Fieldworks machine could provide, given our need for sustained usage for about one- to two-hour testing periods. The tracking and display equipment had comparatively modest power requirements of under 10 watts each. We ended up powering all components but the handheld computer (which used its own batteries) with an NRG Power-MAX NiCad rechargeable battery belt. It had the added advantage of allowing a fully charged replacement belt to be plugged in prior to unplugging the depleted batteries, without interrupting power.

A.2 MARS 1999

In 1998, the author of this thesis took over the lead in the MARS prototyping and development activities following the Touring Machine. Designing and improving the MARS prototypes happened in collaboration with fellow Ph.D. student Elias Gagas at the time. While we kept the overall system concept the same, the choice among many new components required new drivers and extensive testing and tuning. Front and back views of the new system are shown in Figure A.2. The main differences of MARS 1999 as compared to the Touring Machine consisted in a new processor board, a new graphics accelerator, a new handheld computer that enabled multimedia applications, a new high-resolution AR display, new wireless network support, and much more accurate position and orientation tracking. The new tracking solution drastically improved overall system registration by using a real-time kinematic differential GPS system with a base station set up on campus, jointly administered and used with Columbia's mobile robotics group, and a new commercial hybrid orientation tracker manufactured by InterSense.

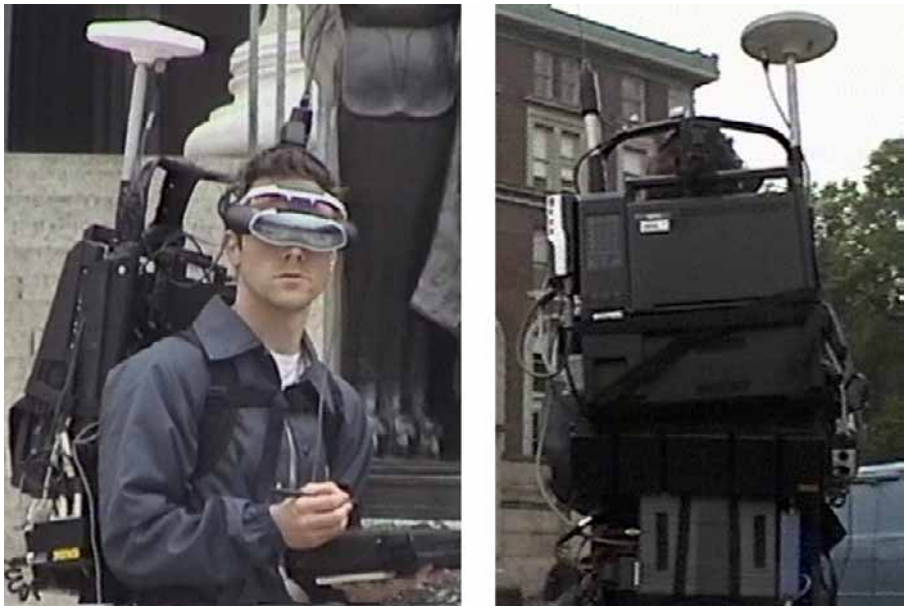


Figure A.2: Journalism student Dave Westreich wearing the MARS 1999. (a) Front view. (b) Back view.

Backpack Computer We upgraded the main board of our FieldWorks backpack computer using an Evergreen AcceleraPCI extension board that turned our computer into a 433Mhz Celeron-based system with 128Mbytes of RAM. While this greatly improved our processing and graphics rendering performance, we did not reap the full benefit of such a change in terms of graphics fill rate, since the new processor's and memory's physical location on a PCI extension board meant that the path from main memory to the graphics accelerator chips now crossed the PCI bus twice.

Graphics Card We upgraded to a Diamond FireGL 1000 Pro PCI board, whose graphics pipeline was based on 3Dlab's PERMEDIA2 chip technology. This card had 8MB RAM, was OpenGL 1.1 compliant and supported texture compression.

Head-worn Display A new see-through display led to huge improvements in AR display resolution, clarity, and brightness: The Sony LDI-100B color display provided a resolution of 800 $\times$ 600 triads. It had a dial to adjust its opacity from nearly totally opaque to about 20% transparent. In our experience, under a bright cloudy sky the preferred setting turned out to be close to the most opaque.

Handheld Computer We switched to a more powerful Fujitsu Stylistic 2300 handheld computer with a 233 MHz Pentium MMX CPU and a transflective 800 $\times$ 600 color dis-

play, designed to be readable in bright sunlight. The Fujitsu's performance was adequate for playing MPEG movies of up to VGA resolution at reasonable frame rates, but it was heavier than we would have liked (3.9 pounds). Weighing weight and size against performance, we also used a 2.2 pound Mitsubishi Amity CP pen-based computer with a 166 MHz Pentium MMX CPU and 640 $\times$ 480 color display.

Orientation Tracker The InterSense IS-300Pro inertial/magnetometer orientation tracker tracked up to four small cube-shaped sensors in 3DOF orientation. For this prototype we used a single sensor mounted rigidly on a head band that we attached to the head-worn display's temple pieces, as shown in Figure A.2.

Position Tracker Position tracking was implemented using an Ashtech GG24 Surveyor RTK differential GPS system, which used both US GPS and Russian Glonass satellite constellations to increase the number of visible satellites. We installed a base station on campus, from which we broadcast correction signals via radio modem. This system provided centimeter-level accuracy in open areas, where there was line-of-sight to more than six satellites. However, tracking degradation and loss remained a problem when passing too close to tall buildings or beneath trees.

Network A new campus-wide wireless network was deployed on Columbia's campus, starting in 1998, growing rapidly in connectivity over the next few years. In the beginning we maintained our own access points in the areas where we did our main testing. We used a WiFi communications infrastructure consisting of Lucent WavePoint II access points and WaveLAN PC cards for our backpack and handheld computers.

A.3 MARS 2000

The next milestone MARS was completed in the year 2000. Important improvements included a much lighter and much more powerful main computer, assembled by Elias Gagas from OEM components and custom-designed casing, a new Sony see-through display with stereo capabilities, a new backpack frame, a Compaq IPAQ PocketPC serving as a smaller version of our handheld computer, gyro-mice and wireless trackballs as further input devices, and a dead-reckoning module for position-correction indoors and outdoors. Figure A.3 shows the MARS 2000 from the side and the back. In the following we describe the new components in more detail.

Backpack Computer The custom-made mobile computer used an embedded PC motherboard (Advantech PCM-9570), a miniaturized version of a regular ATX motherboard,

Figure A.3: The author wearing the MARS 2000. (a) Side view. (b) Back view.

with fewer expansion slots, which supported Pentium II class CPUs. We first used a 466MHz Celeron processor, and later upgraded to a 600MHz Pentium II. We populated the board with 512MB of RAM. Expansion happened through a 32bit 33MHz PCI bus and PC104+ bus on the motherboard. We added a PC104 PCMCIA module (Advantech PCM-3112) to support storage (IBM Microdrive) and wireless networking, and a PC104 sound module (Diamond Systems Crystal-MM-HP). Four serial ports allowed us to attach devices, including position and orientation trackers and interaction devices.

Graphics The lack of sufficiently fast integrated 3D adapters drove us to use desktop graphics hardware once more. Since the class of small motherboards we considered did not support AGP, we were constrained to use PCI graphics cards. The best choice for our mainly OpenGL based software turned out to still be the relatively old Diamond FireGL 1000 Pro PCI board, which used 3Dlab's PERMEDIA2 chip with 8MB RAM. This card had excellent OpenGL 1.1 driver support, which set it apart from many newer competitors. It also had hardware quad buffering, enabling field-sequential SVGA stereo, which was important in order to make full use of our new stereo-capable display. We also used a Diamond/ATI FireGL 1 PCI board, which provided faster graphics generation, but no stereo support, and later an NVidia GeForce2 MX400 PCI board, as soon as it became available in late 2000.

Figure A.4: Elias Gagas wearing earlier version of MARS 2000 with first iteration mobile core computer.

Headworn Display We upgraded to the Sony LDI-D100B, which is the stereo version of the Sony LDI-100 used in the previous MARS prototype.

Interaction We experimented with quite a few alternatives for our handheld computers that since the beginning of our MARS work had accompanied the backpack computer. The most promising solution turned out to be a Compaq IPAQ Pocket PC with PCMCIA support sleeve to enable the use of a WiFi networking card. The interfaces we developed did not require huge amounts of text to be entered to the system, so we felt that text input via the Pocket PC was an acceptable solution. We also experimented with a small form-factor wrist-mounted keyboard, which ended up mainly being used for online debugging in the field. Some interfaces we developed on the MARS 2000 were controlled solely by a wireless Trackman Live trackball or a Gyration GyroMouse – a wireless mouse with integrated gyroscope and inclinometer.

Power We eliminated the need for the massive nickel-cadmium battery belt we were using with the previous systems by switching to Sony NP F750 InfoLithium Lithium-Ion batteries, connecting in parallel two sets of two serially linked batteries each. Such an arrangement of four batteries allowed the mobile computer and the orientation tracker and GPS subsystems to run for a period of about two hours. Figure A.4 shows an earlier

version of MARS 2000, which still relied on the NiCad battery belt of previous prototypes. Notice also that this version was realized with an earlier incarnation of the mobile core computer assembled by Elias Gagas.

Backpack Frame Now that we finally replaced the bulky ruggedized FieldWorks machine and battery belt that had been among the heaviest components of our previous two prototype iterations, we also switched to a smaller, more flexible Dana Design Flatbed backpack frame to keep all the remaining parts together (Figure A.3).

Point Research Dead-Reckoning Module When outdoors with line of sight to at least four GPS or Glonass satellites, our system continued to be position-tracked by an Ashtech GG24 Surveyor RTK differential GPS system. For dead reckoning in covered areas or urban canyons, and for indoor tracking, we employed a Point Research PointMan Dead-Reckoning Module (DRM). More information on our indoor tracking strategies can be found in Section 3.3.1.

A.4 MARS 2001/2002

The year 2001 brought a long awaited innovation. Finally, powerful 3D graphics accelerator chips became available as integrated components on laptop and notebook computers. This resulted in notebook computers matching desktop solutions in terms of graphics performance. The need to resort to external PCI graphics board solutions for our MARS prototypes vanished. We could now replace our self-constructed mobile computer with an off-the-shelf laptop, which brought with it advantages in sturdiness, reliability, power consumption, heat dissipation, and the return of an integrated full-size keyboard and screen (for in-the-field debugging) that the original FieldWorks machine had provided and that we had started to miss during our experiments with MARS 2000.

Because we opted to leave the display connected, there was not really a huge reduction in size compared to the previous system, but the way we mounted the laptop on a backpack frame, offering the choice of opening up the display vertically, brought about another benefit: we could now demonstrate to onlookers what the person wearing the device was seeing (minus the real-world component that replaces the black areas on the screen). We made use of this feature in many live demonstrations of the system, including our exhibition booth at ACM SIGGRAPH ’01 *Emerging Technologies* and outdoor demos at IEEE and ACM ISAR ’01. Figure A.5 shows the front and back views of MARS 2001.

Other additions/upgrades to the system included microphone and camera support, a new ruggedized mount for the display and orientation tracker, and a new handheld

Figure A.5: Hrvoje Benko wearing the MARS 2001. (a) Front view. (b) Back view.

computer.

Backpack Computer Our first notebook-based MARS used a DELL Inspiron 8000 computer with a 700MHz Mobile Pentium III processor and 512MB RAM. In 2002 we switched to a DELL Precision M50 laptop with a 1.9GHz and later 2.2GHz Mobile Pentium IV processor and 1GB main memory. Both computers came with high-performance on-board graphics chips from NVidia, as described in the following paragraph.

Graphics The Inspiron 8000 laptop had an integrated NVidia GeForce2 Go chip with 32MB on-board memory, connected via an internal AGP bus. The laptop outperformed our Geforce2 MX400 based self-assembled computer (MARS 2000) in both Viewperf benchmarks and MARS application performance (helped along by AGP vs. PCI and the faster main processor). The GeForce2 Go chip did not officially support stereo, but we patched the graphics driver, duping it into thinking that the chip was actually one of the virtually indistinguishable Quadro family, and thus achieved stereo support. The Precision M50 sported a Quadro4 500 GoGL with 64MB video RAM, internal VGA bus, and native stereo support. As mentioned before, this chip represented a more than 150-fold increase in graphics performance compared with the Touring Machine of 1997.

Display In 2002 we tested a MicroVision Nomad display (cf. Section 2.4.2) and added it as a choice to our MARS. The Nomad is a monocular retinal scanning display that

produces a very bright monochromatic (shades of red) image on top of a highly transparent background. It is very daylight-readable, but heavier than is comfortable over an extended period of time.

Handheld Computer We switched to a lighter and more powerful Fujitsu Stylistic LT C-500 handheld computer with a 500 MHz Celeron CPU, 256 MB RAM, 800x600 TFT Color LCD and anti-reflective digitizer for stylus input. This handheld weighs about 2.5 lbs including battery pack.

Interaction We added microphone and camera support to the MARS. The microphone was integrated into a a new ruggedized mount for the display and orientation tracker (see Figure A.5). The camera support was tested with an Orange Micro FireWire iBOT, which unfortunately is slightly too big to be comfortably mounted on a head set. We are currently using a Point Grey Dragonfly FireWire camera, which fits nicely on top of a pair of Sony Glasstron glasses.

Power Since the laptop comes with its own battery solution, we could scale back our chain of Sony NP F750 InfoLithium batteries from four to one. While we provided for the possibility to use two InfoLithium batteries in parallel (which has the pleasant side-effect of allowing hot-swaps of batteries), one Sony NP F750 battery is sufficient to power the GPS and orientation tracker subsystems for a period of about 1.5 hours.

Backpack Frame In order to mount the open laptop onto a flat backpack board (see Figure A.5b), we covered the board and the laptop's bottom with industrial strength Velcro™.