Study on Software Architectures for Augmented Reality Systems

Report to the ARVIKA consortium

Prof. Bernd Brügge, Ph.D
Asa MacWilliams, Thomas Reicher

Chair for Applied Software Engineering
Institut für Informatik
Technische Universität München

Technical Report TUM-I0410

Prof. Bernd Brügge, Ph.D.
Asa MacWilliams *(macwilli@in.tum.de)*
Thomas Reicher *(reicher@in.tum.de)*

Chair for Applied Software Engineering
Institut für Informatik
Technische Universität München
Boltzmannstr. 3
D-85748 Garching b. München

Contact: Asa MacWilliams, Thomas Reicher – *(macwilli,reicher)@in.tum.de*

This study was published internally within the ARVIKA project in October 2002 and publically as Technical Report TUM-I0410 in July 2004.

Contents

List of Figures

List of Tables

1. Introduction

This study was conducted from April through July of 2002 at the Chair for Applied Software Engineering of the Technische Universität München. It was commissioned by the ARVIKA project [arv01] as a contribution to the ongoing analysis of the ARVIKA software architecture and in response to the recommendations of the ARVIKA review board in October 2001.

In this document, we present the results of the study. These include an overview of the Augmented Reality (AR) research landscape from the perspective of software architecture; in-depth descriptions of individual architectures; a reference architecture for comparing AR systems; and a classification of the architectural approaches used to address the subproblems of AR.

This report targets architects of new and of existing augmented reality systems wishing to chose an architecture for their system or evaluate and improve their own architecture.

1.1. Outline

This report is organized as follows:

Introduction Explains the purpose, organization and methods of the study.

Overview of Projects Gives short descriptions of 18 different AR research projects and the systems they have built.

Desired Quality Attributes Explores the requirements for AR systems, as indicated by members of the research groups themselves.

System Analysis Describes a reference architecture for AR and shows how 5 individual AR systems apply variations of this architecture.

Comparison of Approaches Describes alternatives for implementing the basic subsystems within an AR architecture, describing advantages and disadvantages of each.

Conclusion Summarizes the results and points to future work.

1.2. Obtaining Data

The data of the study was obtained from three sources:

- First, we gathered published papers of the various research groups and material from projects' web sites.
- Second, we distributed a questionnaire to 20 groups asking about the software architecture used within their projects and the rationale behind it.

- Third, we were able to make personal visits to several labs to conduct more detailed first-hand interviews with the software architects, based upon the structure used in the questionnaire.

1.3. Methods

Our methods were inspired by ATAM, the Architecture Tradeoff Analysis Method, as described in [CKK02], and its predecessor SAAM, the Structured Architecture Analysis Method, described in [BCK98].

Both methods are designed for internal architecture reviews within a software development project, as this is the point at which the careful analysis of an architecture proves most fruitful. However, with some modifications, the methods used can also be adapted to comparing architectures of different development projects.

One of the basic tenets of architecture evaluation is that an architecture cannot be evaluated without knowledge of the purpose of the system it supports [CKK02]. Thus, ATAM attempts to extract a list of the desired quality attributes (or requirements) from the business or research goals of the project. It then analyzes the architectural approaches used in the software architecture, and finds the tradeoffs between quality attributes induced by the architecture.

In comparing different architectures, we were able to identify common architectural approaches by mapping the individual architectures onto an abstract "reference" AR software architecture we developed as a generalization of the individual architectures. The subsystems of this reference architecture correspond to subproblems of AR that can be more or less easily isolated from each other.

We then classified the approaches used within the subsystems following an architectural pattern scheme, describing why, when and how to use each approach, and showing which systems they were used in.

Each of the following chapters contains a section titled "Methods" in which we go into more detail about the methods and concepts used within the study.

1.4. How to Use This Report

With the material in this report, software architects building Augmented Reality systems should be able to evaluate architectural approaches for their system based on the specific properties of their application domain and project background.

As an example, imagine a developer who needs a rendering approach for a medical AR system. Chapter 3 shows that medical applications place a high priority on a high quality of rendering and on a low tracking/rendering latency. Chapter 5 shows that using a custom OpenGL rendering component satisfies these requirements. It also shows that this can have a negative impact on the modifiability of the system; the developer would then have to consider whether this trade-off is acceptable.

On the other hand, for a maintenance application, modeling repair application logic with a multimedia flow description provides the greatest modifiability to the application logic; this requires a scene-based rendering component, e.g. a VRML browser. A browser, however, generally does not provide the same level of performance and fine-grained control that a native OpenGL rendering component does.

The analysis of software architectures for the field of AR is still at the beginning, so we cannot give definitive recommendations. However, we can show what approaches are being used in the field and provide evidence as to which of them are appropriate under which circumstances.

1.5. Acknowledgements

We are very much obliged to the ARVIKA consortium, without which this study would not have been possible. Special thanks go to Wolfgang Friedrich, Siemens A&D, the consortium leader; Didier Stricker, Fraunhofer IGD, the project leader for basic technologies; and Dr. Rüdiger Krahl, Deutsches Zentrum für Luft- und Raumfahrt, the ARVIKA project coordinator for the German Federal Ministry of Education and Research.

We would like to thank all AR researchers who filled out our questionnaire, providing us with the all-important raw data for our architecture evaluation. Particularly we wish to thank Yohan Baillot (Naval Research Laboratory), Wolfgang Birkfellner (University of Vienna), Eric Foxlin (InterSense, Inc.), David Mizell (Intel Corp., formerly The Boeing Company), Wouter Pasman (Delft University of Technology), Wayne Piekarski (University of South Australia), Artur Raczynski (Siemens AG), and Jong Weon Lee (University of Southern California).

Thanks also to the members of the Software Engineering Institute of Carnegie Mellon University, especially Mario Barbacci, who provided us with helpful insights into our architecture evaluation methods.

Special thanks goes to all those who spared some of their time to discuss their AR architecture ideas with us, especially David Garlan (Carnegie Mellon University), Yakup Genc (Siemens Corporate Research), Tobias Höllerer (Columbia University), Blair MacIntyre (Georgia Institute of Technology), Gerhard Reitmayr and Dieter Schmalstieg (Technical University of Vienna), Frank Sauer (Siemens Corporate Research), and Richard DeVaul (Massachusetts Institute of Technology).

2. Overview of Projects

In this chapter, we give an overview of existing AR systems, their goals and current status.

2.1. Methods

The system descriptions given within this chapter were collected from the Web and from published papers, and then enriched with additional material we gathered from the returned questionnaires and within personal interviews. For each system, we give a short description of the project and a classification according to the following parameters:

Application domain background The domain in which the system is planned to be used, including:

- Construction
- Maintenance
- Inspection
- Medicine
- Military
- Automotive
- Collaboration and communication
- Navigation and surveying
- Training and instruction
- Visualization for marketing
- Games
- Consumer applications

Project type Academic research, applied research or industrial development.

Time to market The time frame of the first commercial products planned using the project's technology.

Website The project's web site.

Contact A contact person for questions about the project.

Where possible, we also give a short description of the system's architecture.
Table 2.1 on page 18 provides a summary of the projects mentioned in this chapter.

2.2. AIBAS

The goal of AIBAS (Adaptive Intent-Based Augmentation System) is to understand how knowledge of the communicative intent of an augmentation can be leveraged to simplify the creation of AR applications that work well in real-world situations with "good enough" tracking and

registration. The underlying insight is that tracking and registration will always be afflicted with errors. If the system can estimate the registration error and knows the user's context and the intent of the augmentation, it can adapt to the error.

The results of this project will be a conceptual framework and associated tools that support the creation of augmentations that can handle the presence of registration errors. A longer-term goal is the development of a toolkit with "AR widgets" for a wide variety of AR applications.

Application domain background.

- Industrial inspection, construction, and maintenance
- Military
- Automotive
- Building industry

Project type. Academic research with focus on human-computer interaction.

Time to market. Long term; mainly research.

Website. http://www.cc.gatech.edu/projects/ael/

Contact. Blair MacIntyre (blair@cc.gatech.edu)

2.3. ArcheoGuide

ArcheoGuide (Augmented Reality based Cultural Heritage On-site Guide) is a project funded by the EU with the goal of providing new ways of information access at cultural heritage sites. Project partners are INTRACOM S.A., Fraunhofer IGD, ZGDV Darmstadt, Centro do Computação Gráfica, Arts & Crafts, Post Reality, and the Hellenic Ministry of Culture.

"The ARCHEOGUIDE project intends to provide new approaches for accessing information at cultural heritage sites in a compelling, user-friendly way through the development of a system based on advanced IT including augmented reality, 3D-visualization, mobile computing, and multi-modal interaction techniques. The system will be tried out in one major European cultural heritage site. In this site particular emphasis will be given to virtual reconstruction of the remains.

The ARCHEOGUIDE system will address the requirements of a wide user selection that includes cultural site visitors, cultural site managers, researchers, and content creators. Cultural site visitors will be provided with a see-through Head-Mounted Display (HMD), earphone, and mobile computing equipment. A tracking system will determine the location of the visitor within the site. Based on the visitor's profile and his position, audio and visual information will be presented to guide and allow him/her to gain more insight into relevant aspects of the site." [Ioa02].

Application domain background. Tourist guidance.

Project type. Applied research with academia, industry and users from the tourist domain.

Time to market. Prototypes before 2003.

Website. http://archeoguide.intranet.gr

Contact. Nikos Ioannidis (nioa@intranet.gr)

2.4. AR-PDA

The goal of this project is to develop new mobile devices for the consumer market that use Augmented Reality techniques to efficiently support end users during their daily tasks. The AR-PDA will augment real camera images with virtual objects (animated 3D, 2D, text) and allows personalized user interaction with the augmented scene. Dedicated annotations will help the user, for example, to repair a (real) washing machine or program a (real) VCR, or visualize new furniture from an online store in the user's living room. The project will design and implement a framework consisting of several hardware and software components. Key aspects forming the different parts of the AR-PDA framework include:

- Mobile device
- Client-server infrastructure
- Computer vision and graphics
- AR authoring support and usability
- Application scenario

The AR-PDA supports consumers during their daily tasks by using Augmented Reality. The technical realization will be based on new mobile phones with integrated video cameras. The user points at an object with the mobile phone; the video stream is sent via wireless network standards like UMTS to a server; this analyzes the image, recognizes the object and determines the context-sensitive annotations; the server then augments the real images and sends them back to the AR-PDA. The user may also call in an additional consultant via video.

In addition to the technical system AR-PDA, a general process model and an authoring tool for efficient generation of Augmented Reality contents will be developed.

The project started in March 2001 and ends in March 2004. It is sponsored by the BMBF (German Federal Ministry for Education and Research). Project participants are UNITY AG, Siemens SBS C-Lab, the University of Paderborn, the Heinz Nixdorf Institute, Lunatic Interactive Productions, Miele & Cie, and the Technical University of Ilmenau.

Application domain background.

- Consumer applications
- Visualization for marketing
- User instruction

Project type. Applied research with academia and industry.

Time to market. First prototypes by 2004.

Website. http://www.ar-pda.com

Contact. Dr.-Ing. Peter Ebbesmeyer (ebbesmeyer@unity.de)

2.5. ARToolKit

ARToolKit is part of the Shared Space project. It is a software library that can be used to calculate camera position and orientation relative to physical markers in real time. This enables the easy development of a wide range of Augmented Reality applications. It was developed

jointly by the the Human Interface Technology Laboratory (HITLab) at the University of Washington, USA, and ATR Media Integration & Communication in Kyoto, Japan.

Application domain background. The background of the Shared Space project which originated the ARToolKit is distributed user collaboration. Meanwhile, however, the AR component ARToolKit is used in several application domains.

- Consumer applications
- Maintenance
- Interface design
- Collaboration

Project type. Academic research; cooperation with industry.

Time to market. Already used in several research and industrial systems.

Websites.

- The Shared Space Project: http://www.hitl.washington.edu/projects/shared_space/
- The ARToolKit download page: http://www.hitl.washington.edu/projects/shared_space/download/

Contact. Marc Billinghurst (grof@hitl.washington.edu).

2.6. ARVIKA

ARVIKA is a project supported by the BMBF (German Federal Ministry for Education and Research) consisting of members from industry and research institutes. The target is to research and develop Augmented Reality technologies that will support development, production and service of complex technical products. This project is application-driven with the focus on the following areas: automobile and aircraft development; production in automobile manufacture and aircraft construction; and service for large technical systems, particularly power stations and machine tools. The development of applications is supported by research on basic Augmented Reality technologies and a user-driven system design. The basic technologies support both high-end applications in product development and low-end mobile applications for skilled workers using belt-worn equipment in production and service environments [arv01].

The basic technologies provide the foundation for the development of Augmented Reality applications. The development focus is on three areas: Augmented Reality basic techniques (tracking and video superimposing), information management (provision of situation-dependent data access), and human-computer interaction techniques with the focus on speech-based interaction.

The architecture of the AR basic technologies is component-based and modular, allowing rapid application assembly by component reuse or extension. Despite the use of different hardware and operating system platforms for the stationary and mobile systems, all software libraries concerning rendering, tracking and device interface can be reused from a single source tree.

Application domain background.

- Product development with the focus on the automobile industry
- Workshop floor production with the focus on automobile and airplane industries
- Machine tools servicing
- Airplane industry
- Maintenance of industrial installations with the focus on nuclear power plants

Project type. Applied research; industry/academic cooperation.

Time to market. First products expected within project duration, i.e. by 2003.

Website. http://www.arvika.de

Contact. Wolfgang Friedrich (wolfgang.friedrich@nbgm.siemens.de)

2.7. Aura

Aura is a project at Carnegie Mellon University, Pittsburgh, which describes itself as follows:

"The most precious resource in a computer system is no longer its processor, memory, disk or network. Rather, it is a resource not subject to Moore's law: User Attention. Today's systems distract a user in many explicit and implicit ways, thereby reducing his effectiveness.

Project Aura will fundamentally rethink system design to address this problem. Aura's goal is to provide each user with an invisible halo of computing and information services that persists regardless of location. Meeting this goal will require effort at every level: from the hardware and network layers, through the operating system and middleware, to the user interface and applications.

Project Aura will design, implement, deploy, and evaluate a large-scale system demonstrating the concept of a "personal information aura" that spans wearable, handheld, desktop and infrastructure computers." [aur02].

Augmented Reality is seen as a technology that supports human computer interaction. There is no explicit AR related research now but the software architecture is designed to take advantage of Augmented Reality in the future.

Application domain background.

- Consumer applications
- Maintenance
- Collaboration

Project type. Academic research; cooperation with industry

Time to market. Not planned; mainly research project.

Website. http://www-2.cs.cmu.edu/ aura/.

Contact. M. Satyanarayanan (satya@cs.cmu.edu).

2.8. BARS

The Battlefield Augmented Reality System (BARS) project examines how three-dimensional strategic and tactical information can be transferred between a command center and individual soldiers operating in an urban environment. It is a multi-disciplinary project encompassing a number of research and technical issues. These include: (i) the design of novel user interfaces; (ii) the design of new interaction methods; (iii) the development of an interactive, scalable three-dimensional environment; (iv) tracking and registration systems of sufficient accuracy; (v) a prototype demonstration system.

Application domain background.

- Military intelligence visualization

Project type Academic research with focus on human-computer interaction and visualization.

Time to market Mainly research system.

Website. http://ait.nrl.navy.mil/vrlab/projects/BARS/BARS.html

Contact. Yohan Baillot (baillot@ait.nrl.navy.mil).

2.9. Boeing Wire Bundle Assembly

The AR wire bundle assembly project at Boeing was the first application for AR. The wearable system uses a see-though head-mounted display to overlay instructions and diagrams on how to assemble an aircraft wire bundle.

For this system, consistency and accuracy of tracking were very important, and the system had to be wearable enough to be accepted by the users who would work with the system all day. Since this was an early system, the head-mounted display was still cumbersome; several workers even refused to wear it.

The hardware setup used a wearable VIA IIs with 250MHz, running DOS, and a TriSen optical tracking system, with markers attached to the wire bundle assembly board.

Translation of the legacy bundle diagrams and connectivity information was done offline and loaded onto the AR system's wearable PC. The three main software components of the AR system were the tracker software (from TriSen), the GUI and the draw loop.

Application domain background.

- Aircraft construction.

Project type. Industrial research and development.

Time to market. Developed starting in 1990 at Boeing; in experimental use; later discontinued.

Website. http://www.boeing.com/defense-space/aerospace/training/instruct/augmented.htm

Contact. Dave Mizell (david.mizell@intel.com).

2.10. DWARF

Goal of the DWARF project is to build a framework for the development of Augmented Reality based a collection of intelligent modules that can be combined flexibly depending on the needs of the desired application.

These AR systems are supposed to be distributed and wearable—they can consist of several small computers that the user can carry around with him. The system will be able to dynamically integrate the services offered by stationary computers in the environment to extend its own capabilities.

AR system are seen as part of an intelligent ubiquitous computing environment. Coming from the research field of intelligent buildings, the DWARF project sees the user with a wearable AR system as an integrated part of his intelligent environment. With wireless communication, systems the user is carrying can contact services provided by systems around him and make use of them. One example is a powerful tracking service that supports the user's own tracking on the wearable.

Application domain background.

- Maintenance
- Production
- Prototyping
- Consumer applications

Project type. Academic research; reused in cooperation with industrial partners.

Time to market. First commercial products using DWARF planned for 2005.

Website. http://wwwbruegge.in.tum.de/projects/DWARF/main/

Contact. Martin Wagner (wagnerm@in.tum.de).

2.11. EMMIE

EMMIE (Environment Management for Multi-User Information Environments) is a project of the Columbia University Computer Graphics and User Interface group.

The goals are to provide services similar to those of a conventional window manager in a Multi-User Augmented Environment; and to actively assist the user by dynamic layout mechanisms.

EMMIE is based on Coterie, a project which started in 1994 and was used to build many AR prototype applications and explore environment management systems. The rationale behind the development of Coterie was to reuse components, abstract from driver code, and handle distribution.

Coterie is written in Modula-3 and handles replication of objects in the network, providing a distributed shared memory model (Repo). Its main function is to replicate the scenegraph (Repo-3D) between different display clients.

Applications for Coterie are written in Obliq, an interpreted language similar to Modula-3, and its extension, Obliq-3D. The application developer uses this language to define properties that are evaluated at run-time, e.g. the position and orientation of a virtual object or user

interface component. The control flow stays within the Obliq-3D system itself. In order to provide animation, Obliq-3D allows timers and triggers that lead to the re-evaluation of application-defined properties.

The Coterie system was very useful for prototyping and application development. A drawback for further development was the rather steep learning curve of the Obliq and Modula-3 languages for student projects.

Application domain background.

- Consumer applications
- Collaboration
- Navigation

Project type. Academic research with focus on human-computer interaction.

Time to market. Mainly research; commercial plans unknown.

Website. http://www.cs.columbia.edu/graphics/projects/emmie/emmie.html

Contact. Tobias Höllerer (htobias@cs.columbia.edu)

2.12. ImageTclAR

ImageTcl is a Multimedia Software Development Environment under development by Charles B. Owen at Media and Entertainment Technology Laboratory at Michigan State University. ImageTclAR extends ImageTcl with new features in support of Augmented Reality and Virtual Reality Development.

Some of features of ImageTclAR are:

- High performance for real-time applications.
- Threaded and multiprocessor support.
- Drivers for Polhemus tracking hardware (ISOTRAK and FASTRAK).
- Camera to workspace calibration.
- Tracker to workspace calibration.
- Head-mounted display calibration.
- A simple programming interface based on Tcl and the Tk Toolkit.
- Modular design which facilities the easy construction of new components for rapid prototyping.
- Automatic build tools.
- Automatic tools for assistance in creation of new data types and commands.

Application domain background.

- Consumer applications

Project type. Academic research.

Time to market. Not expected to create commercial products; open source system.

Website. http://metlab.cse.msu.edu/imagetclar/

Contact. Charles B. Owen (cbowen@cse.msu.edu)

2.13. MARS

The research on the Mobile Augmented Reality Systems (MARS), also at the Columbia University Computer Graphics and User Interfaces Group, is aimed at exploring the synergy of two promising fields of user interface research: Augmented reality (AR), in which 3D displays are used to overlay a synthesized world on top of the real world, and mobile computing, in which increasingly small and inexpensive computing devices and wireless networking allow users to have access to computing facilities while roaming the real world. In exploring different application scenarios for mobile AR, the main focus is on the following lines of research:

- Identifying generic tasks a mobile user would want to carry out on a context-aware computing system.
- Defining a comprehensive set of reusable user interface components for mobile Augmented Reality applications.
- Making combined use of different display technologies ranging from head-worn to hand-held to palm-top devices in order to optimally support a mobile user.

The MARS system reuses the Coterie-based user interfaces, but adds new ones based on Java3D. The Coterie system communicates with the new components via a wrapper.

For replication between different systems, MARS uses a custom socket-based TCP and UDP communication protocol, allowing rapid distribution of tracking data. JDBC database access is used to obtain application information, e.g. for a campus information system.

User interface components are written as Java classes with well-defined interfaces (e.g. menu, calibration) that directly modify the Java3D scene.

This system was not used to build as many applications as Coterie, although it would easily have been possible. Application logic is written in Java code, and accesses the Java user interface components directly. This fine-granular control allows adaptive user interfaces and advanced view management.

A central server coordinates the various Java-based and Coterie-based systems, synchronizes between users and stores the position of objects etc. in a central database.

Application domain background.

- Consumer applications
- Navigation
- Construction

Project type. Academic research with focus on human-computer interaction.

Time to market. Mainly research; commercial plans unknown.

Website. http://www.cs.columbia.edu/graphics/projects/mars/mars.html

Contact. Tobias Höllerer (htobias@cs.columbia.edu)

2.14. MR Platform

"MR Platform offers the newest video see-through HMD and a powerful and comprehensive SDK for developing and creating an MR application." [mrl02].

The MR Platform is a research result of the Mixed Reality Systems Laboratory Inc., a Japanese research consortium supported by the Japanese government and Canon Inc. The successor is MR Systems Laboratory, Canon Inc. The COASTAR head-mounted display is available together with software development kit for constructing AR applications. The HMD included in the product offers an integrated stereoscopic camera whose optical axes coincide with the display axes, stereoscopic display, simultaneous capture, and other features. The SDK is composed of a C++ class library and calibration utilities for a standard Linux (i386) environment. This flexible SDK lets you create an MR application with other graphics libraries or toolkits such as OpenGL, GLUT, and Open Inventor.

AquaGauntlet is an application that demonstrates the capabilities of the MR platform [aqu00].

Application domain background.

- Multiuser multimedia applications, arcade games
- Construction

Project type. Originally academic research, now transferred to industry.

Time to market. Available today.

Website. http://www.mr-system.co.jp/mr-platform/

Contact. Mixed Reality System Laboratory Inc. office (office@mr-system.co.jp).

2.15. Siemens Corporate Research Princeton

"Building and maintaining 3D computer models of scenes is a major impediment in the broad acceptance of virtual reality. The focus of our Industrial Augmented Reality group is to develop technologies for the image-based creation of computer models, augmenting computer models with images of the real world, and the transfiguration of image/video by computer models. The goal is to have transparent access and control of complex systems and processes via virtual 3D interaction for industrial applications such as process control, factory update, and sales and maintenance." [scr02].

Currently, several different demonstration AR systems have been built at Siemens Corporate Research. The components for maintenance systems integrate with the ARVIKA project; the medical systems use a dedicated high-performance architecture of their own.

Application domain background.

- Maintenance
- Industrial service
- Medicine

Project type. Industrial research.

Time to market. Technology to be integrated with future corporate products; e.g. with ARVIKA after 2003.

Website. http://www.scr.siemens.com/2a.html.

Contact. Nassir Navab (navab@scr.siemens.com).

2.16. STAR

STAR (Services and Training Through Augmented Reality) is an EU funded project with Siemens, TU Delft, KU Leuven, University of Geneve, Realviz and EPFL. The project focuses *"... on developing Augmented Reality techniques with a view to developing commercial products for training, on-line documentation and planning purposes. With its two industrial partners, STAR will focus on developing the technology so that it is applicable to the industrial world."* [con02].

Key technological goals of the STAR project are:

- Rapid 3-D and 2 (1/2)D modelling of industrial facilities from images. Goals are cheap and effective tools for creating such models for Augmented Reality simulations, with seamless blending of real and virtual objects.
- Interaction between human operators and mixed reality environments including virtual people. STAR is developing new tools for human-machine interaction between an operator and a mixed/augmented environment.
- Virtual editor. Based on 3D scene models, STAR will produce a tool that automatically picks the best views, given the action of an instructor, and then handles camera handover, for example to minimize occlusions.
- Integration of all these tools into an Augmented Reality system to validate and test for both on-site and on-line remote training.

Application domain background.

- Training
- Services

Project type. Applied research.

Time to market. Business plan development is part of the project.

Website. http://www.realviz.com/STAR/index.htm

Contact. Artur Raczynski (artur.raczynski@mchp.siemens.de)

2.17. Studierstube Augmented Reality Project

The Studierstube project addresses the question of how to use three-dimensional interaction and new media in a general work environment, where a variety of tasks are carried out simultaneously. It searches for an 3D interaction metaphor as powerful as the desktop metaphor for 2D. The Studierstube system uses collaborative Augmented Reality to embed computer generated images into the real work environment.
The collaborators on this project include:

- Vienna University of Technology, Austria
- VRVis Research Center for Virtual Reality and Visualization, Austria
- Graz University of Technology, Austria
- Fraunhofer Center for Research in Computer Graphics, Providence, Rhode Island, USA
- Fraunhofer Institut für Graphische Datenverarbeitung, Darmstadt and Rostock, Germany

- University of Tübingen, Germany
- Human Interface Technology Lab, University of Washington, Seattle, Washington, USA

Application domain background.

- Consumer applications
- Distributed collaboration
- Medicine

Project type . Academic research; application partners especially in medical research.

Time to market. Mainly academic research; commercialization plans unknown.

Website. http://www.studierstube.org/index.html

Contact. Dieter Schmalstieg (schmalstieg@ims.tuwien.ac.at).

2.18. Tinmith

The Tinmith project at the University of Southern Australia describes itself as follows: *"The Tinmith system is a complete software architecture designed to develop Augmented Reality and other software that deals with trackers, input devices, and graphics. Tinmith is based on a completely free software system comprising the Linux kernel, GNU tools and libraries, the GNU C/C++ compiler, XFree86 graphics server, GGI graphics interface, OpenGL 3D renderer, PostgreSQL database, and Freetype font renderer. The current version of Tinmith is "Evo 5"."* [tin02].

Application domain background.

- Consumer applications
- Military
- Gathering of geographical data
- Games

Project type. Academic research; several iterations built; cooperation with Australian office of defense.

Time to market. Long-term.

Website. http://www.tinmith.net/.

Contact. Wayne Piekarski (wayne@cs.unisa.edu.au).

2.19. UbiCom

The Ubiquitous Communications (UbiCom) program is a multidisciplinary research program at Delft University of Technology. The program aims at carrying out research needed for specifying and developing wearable systems for mobile multimedia communications. In particular the focus is on:

- real-time communication and processing of visual information for context-aware applications such as Augmented Reality,
- high bit rate communication at 17 GHz,
- architectural issues and performance optimization for heterogeneous communication and computation systems.

In the vision of UbiCom, a user carries a wearable terminal and a lightweight see-through display with virtual information that augments reality, projected over and properly integrated with the real world. The wearable system contains a radio link that connects the user to the ubiquitous computing resources and the Internet.

Camera images of the user's environment are sent to the backbone and matched to a 3-D description of the real world to determine the user's position and to answer user questions relating to the environment [Lag00].

Application domain background.

- Consumer applications
- Telecommunication

Project type. Academic research.

Time to market. Mainly academic research; commercialization plans unknown.

Website. http://www.ubicom.tudelft.nl.

Contact. Prof. Dr. ir R. L. Lagendijk (R.L.Lagendijk@its.tudelft.nl).

2.20. Further Projects and Groups

In this section we list projects and groups we know of, but, due to time and space limits (or scarcity of published information), have not described in this chapter.

- Human Computer Interaction at Rockwell Scientific Company (http://hci.rsc.rockwell.com/)
- PacificVR (http://www.pacificvr.com)
- TriSen Augmented Reality Company (http://www.trisen.com/)
- AT&T Laboratories Cambridge (http://www.uk.research.att.com)
- HRL Information Sciences Laboratory (http://www.hrl.com/TECHLABS/isl/index.html)
- MIT Medial Lab (Sandy Pentland, Hiroshi Ishii)
- Oxford University Visual Geometry Group (http://www.robots.ox.ac.uk/ vgg/)
- Computer Graphics and Immersive Technologies group at the University of Southern California (Ulrich Neumann) (http://graphics.usc.edu/cgit/research/research.html)
- The UNC tracker project (Henry Fuchs) (http://www.cs.unc.edu/ tracker/)
- ETC-LAB (Paul Milgram) (http://vered.rose.toronto.edu/etc-lab.html

- The Varioscope AR project at the Department of Biomedical Engineering and Physics, Vienna General Hospital (Wolfgang Birkfellner)
  (http://www.lifeoptics.com/variohtml/varioar.htm)

2.21. Conclusion

Almost all of the projects we described in this chapter are still in the research stage; very few systems are actually commercially available. Given this research focus, many groups are concentrating on one particular aspect of AR such as tracking, rendering, multimodal user interfaces or context awareness. Complete commercially viable system architectures are rare.

Those projects close to commercialization fall into two groups: on the one hand are technology providers who sell tracking rendering hardware and software, and on the other are system integrators such as the ARVIKA application domain subprojects in industrial service and maintenance.

This last group of projects is the one that must consider their software architecture the most carefully; it has to allow for future modifications, but most importantly, it must be able to deliver the quality attributes for the commercial application.

Project	Application domain	Institution	Research type	Availability
AIBAS	Industrial inspection, military, automotive, building industry	Georgia Institute of Technology (USA)	Academic	—
ArcheoGuide	Tourist guidance	Fraunhofer IGD, EU partners (EU, EU funded)	Applied	2003
AR-PDA	Visualization, instruction	C-Lab Siemens, Uni Paderborn (DE, BMBF funded)	Applied	2004
ARToolkit	Maintenance, interface design, consumer, collaboration	University of Washington (USA)	Academic	available now
ARVIKA	Industrial maintenance, production and service,	Siemens A&D, partners (DE, BMBF funded)	Applied	2003
Aura	Maintenance, consumer applications	Carnegie Mellon University (USA)	Academic	—
BARS	Military intelligence visualization	Office of Naval Research (USA)	Academic	—
Boeing	Aircraft construction	Boeing (USA)	Industrial	1990s
DWARF	Maintenance, production, prototyping, consumer	Technische Universität München (DE)	Academic	2005
EMMIE	Consumer applications, collaboration, navigation	Columbia University (USA)	Academic	—
ImageTclAR	Consumer applications	Michigan State University (USA)	Academic	—
MARS	Consumer, navigation	Columbia University (USA)	Academic	—
MR Platform	Multiuser multimedia applications, games, construction	Mixed Reality Systems Laboratory (JP)	Industrial	available now
Siemens CR	Maintenance, industrial service, medicine	Siemens CR (USA)	Industrial	2004
STAR	Training, services	Siemens ZT and EU partners (EU, EU funded)	Applied	being planned
Studierstube	Consumer, collaboration, medicine	Technische Universität Wien (AT)	Academic	—
Tinmith	Consumer, military, surveying, games	University of Southern Australia (AU)	Academic	—
UbiCom	Consumer, telecommunication	Technical University of Delft (NL)	Academic	—

Table 2.1.: Overview of AR Projects

3. Desired Quality Attributes

In this chapter, we describe the quality attributes that project members desire of their Augmented Reality systems, depending on the desired application domains. A system's software architecture is measured by the degree to which it helps the system achieve the desired quality attributes.

3.1. Methods

In ATAM, the desired quality attributes of a system are derived from the business drivers of the project—target market, timeframe, and application domain.

A central component of ATAM is the *quality attribute utility tree*. This describes the desired quality attributes of the system, grouped into broad categories, preferably quantified, and ranked according to priority and difficulty to achieve architecturally.

We assembled a list of architecturally relevant quality attributes, based on [BCK98], but specially adapted for AR systems. For each attribute, we asked in our questionnaire for short descriptions of how stringent the requirement was. As an example, in a medical AR system, reliability must be extremely high. The full list of attributes we asked about is in Appendix B.

We then classified the quality attributes based upon how often they were listed as important in the returned questionnaires:

- Infrequently ranked quality attributes were ranked as important by fewer than 50% of the respondents,
- Frequently ranked priority quality attributes by 50% or more.

The results are listed in this chapter in a structured form; this forms a generic quality attribute utility tree for AR systems.

Where possible, we also identified which application domains or which types of projects led to which choice of desired quality attributes.

3.2. Frequently Ranked Quality Attributes

These quality attributes were ranked as important by more than half of the respondents to our questionnaire. Most of them are important for nearly all AR systems. We conclude (although with caution) that these are requirements that any serious software architecture for Augmented Reality must address.

3.2.1. Run-Time Attributes

The following attributes are observable at system run time; thus, the directly affect the user of the system.

Performance

Latency of tracking and rendering Considered important in almost all systems; the maximum limit of 10 ms was given several times.

Complexity of the scene that can be rendered For most systems, simple models such as wireframes are enough, a few (e.g. medical) systems require complex scenes such as volume rendering.

Reliability

Repeatability of tracking data Important for many systems. Especially for assembly, the location indicated must remain consistent between construction steps.

Tracking accuracy Important for many systems; however, the required degree of accuracy varies from below 1 mm (for medical applications) to several meters (for outdoor navigation).

Tracking range Important for most systems; however, the required range varies from a single room to wide-range outdoor operation.

Robustness Important for most systems, often a significant acceptance factor.

Mobility: Wireless operation Important for most systems, often using WaveLAN networks.

Mobility: Indoor operation Most systems support this mode of operation.

Mobility: Outdoor operation Important for many systems, determined by the application.

Mobility: Network-disconnected operation Important for many systems, mainly outdoors.

Usability

Clear field of view Important for many systems, although limited by head-mounted display technology.

Richness of information presented Important for many systems; for a few, the information needs to be filtered first to avoid overloading the user.

Functionality

Track the user's head Important for almost all systems.

Track objects in the environment Important for many systems, e.g. surgical instruments or machine tools.

Inside-out tracking Using trackers carried by the user is important for many systems, especially outdoors.

Outside-in tracking Trackers mounted in the environment are important for many systems, especially indoors.

Multiple tracking devices Important for many systems, mostly in conjunction with sensor fusion.

Speech recognition Important for many systems, often as third-party components.

3.2.2. Non-Run-Time Attributes

These attributes are not observable at system run time. They do not affect the user of the system, but rather the developers, maintainers, and project managers.

Modifiability

Adapt the system to new requirements Important for most systems, especially in research; some use a framework or component approach.

Reuse components in new systems Important for many systems, especially in evolving research prototypes.

Add components Important for many systems; again, sometimes using a framework approach.

Integrability

Integrate other AR components Important for many systems; the AR Toolkit is cited frequently.

Testability

Test individual components Important for many systems, especially for testing tracking components.

3.3. Infrequently Ranked Quality Attributes

These quality attributes were ranked as important by less than half of the respondents to our questionnaire. This often means that they are only needed in ceratin application domains or ceratin types of research projects.

3.3.1. Run-Time Attributes

As above, these attributes affect the systems' users.

Performance

CPU load For several systems, keeping the CPU load low enough (for a workstation, a wearable or a palm-sized device) is a significant requirement.

Reliability

Rendering quality Important for several systems, especially in medical applications.

Fault tolerance Important for several systems, mainly with regard to a tracker failing.

Availability

Uptime Important for several systems; most others are mainly demo setups.

Security

Support for users with different access rights Important for only a few systems.

Information privacy Important for only a few systems, mainly in outdoor roaming applications.

Withstand attacks Generally not considered important.

Usability

Limiting the hardware on the user's body Important for several systems, several others currently have heavy or bulky setups.

Freedom of movement Important for several systems, mainly limited by wearable setups.

Reconfigurability at runtime Important for several systems; for some, this includes reconfiguring the hardware.

Safety Not considered important at the current state of development.

Use in harsh environments Important for a few systems, e.g. medical or military.

Support for different simultaneous input modalities Important for a few systems; for some, this is the research focus.

Functionality

Track multiple users Important for several systems, e.g. collaborative applications; often implemented with separate trackers.

Track users' hands Important for several systems, often in conjunction with gesture recognition.

Adapt to user's preferences Important in several systems, mainly in systems with a clear target market, e.g. medical or maintenance.

Adapt to user's abilities Only important in a few systems, especially for maintenance.

Adapt to environmental context Important in several systems, sometimes research focus on context-awareness.

Stereo rendering Important for several systems, e.g. surgical assistance.

Support for multiple viewers Important for a few systems, mainly in caves.

Projection onto non-worn surfaces Important for a few systems, especially in collaborative applications.

Speech synthesis Important for several systems, often as third-party components

Gesture recognition Important for several systems; e.g. hand gestures for menu operations and selection by direction of gaze.

Tangible interaction Important for only a few systems.

Interaction between users Only important for a few systems, for collaborative applications.

3.3.2. Non-Run-Time Attributes

As above, these attributes everyone involved with the system besides the users.

Portability

Number of supported hardware platforms Several systems support only a single platform (usually Windows 2000 or Linux); several others support multiple platforms.

Ability to port to new platforms Important for several systems, often wearable ones.

Integrability

Ease of integrating components to form the entire system Important for only a few systems.

Integrating legacy components Important for several systems, especially for industrial applications.

Testability

Ability to detect faults in the system Important for several systems; often achieved with assertions.

Proven correctness Important for a few systems, e.g. medical applications.

3.4. Conclusion

Given that the number of questionnaires we received was not very high, we cannot make statistically significant statements about which attributes are specifically important for which types of applications or projects. This could be determined with a greater number of data points.

Nevertheless, for a few application domains, we can list some requirements that where listed several times in the answers to our questionnaire.

Medical surgery applications have particularly stringent requirements for *tracking/rendering latency* and *tracking accuracy*. Also, the *quality of rendering* has to be quite good (one system description mentioned volume rendering). Of course, the system must be usable in *harsh environments*; however,*restriction of field of view* is often not an issue, as the display can be built into surgical microscopes. The system must be *robust*, otherwise user acceptance is a problem.

Maintenance applications need to *adapt to the user's preferences* and *adapt to the user's abilities*. *Freedom of movement* is an issue; but *quality of rendering* may be limited to e.g. wireframes. Again, the system must be *robust*, otherwise user acceptance is a problem.

Outdoor applications have specific requirements towards the *range of tracking* and *network-disconnected operation*. Here, *information privacy* can be an issue, and limiting *CPU load* is important for wearable devices.

Collaborative applications have to address multiple users. They thus need to *track multiple users*, allow *interaction between users* and also use different display techniques such as *projection onto non-worn surfaces*. *Freedom of movement* is often greater than for single-user setups, and *security* becomes an issue with different access rights.

There are also some general statements that can be made about how quality attributes depend on the type of project:

Academic research projects can afford to relax certain requirements in order to concentrate on their specific research area. This may lead to a system which is not actually useful in a realistic environment, but may also provide great benefits in specific other quality attributes. Essentially, research systems can make larger trade-offs.
For example, an architecture could trade off *latency* against *rendering quality* or *tracking accuracy* in order to test new image processing or graphics algorithms. Similarly, a flexible framework architecture might trade off *CPU load* against *reusability* and *modifiability*.

Industrial projects do not have quite this freedom; the final system must actually be useful. Thus, trade-offs have to be considered more carefully. Also, in industrial systems, certain

quality attributes such as *legacy system integration*, *robustness* and *testability* are much more important.

4. System Analysis

In this chapter, we analyze several different AR software architectures in detail. We do not attempt to make comparative statements such as "system X is better than system Y"—indeed, such a broad claim would be impossible. Software architectures designed to solve different application problems cannot be compared on such broad a level. However, with the material in this chapter, we provide a basis for discussing the relative merits of different architectures. In the future, it may be possible to build specific AR "benchmark" applications which could be used to compare different architectures quantitatively.

4.1. Methods

The descriptions for the various Augmented Reality systems are written by various authors with various description techniques and tools. This makes it quite difficult to compare the developers' approaches.

In order to be able to make a systematic comparison, we developed an abstract Augmented Reality reference software architecture, as a superset of the architecture descriptions we received. No single system actually uses this architecture, but each system can, to some extent, be mapped onto a subset of the reference architecture.

Before introducing the reference architecture, however, we will introduce the concept of abstraction levels to make the scope of our discussion clearer.

4.2. Abstraction Levels

Hofmeister et al. [HNS00] identified four different meanings of the term *software architecture*: *(application) software architecture*, *product line architecture*, *reference architecture* or *domain-specific software architecture*, and *architectural style* or *architectural pattern*. A software architecture describes the architecture for a particular system or product. There are application specific components, adaptations, and configurations. A product line architecture describes an architecture for a set of products that can be adapted for a series of similar applications where only minor adaptations for a customer or platform are needed. A domain specific architecture describes an architecture for a system or subsystem in a particular domain. Such an architecture can be used as a starting point to develop an application in that domain. An example would be a reference architecture for web applications or Augmented Reality applications. And finally, an architectural style is usually not domain specific. It describes a general solution on the design level. Hofmeister et al. describe these terms and relate them to each other. One result of their analysis is that these terms are not mutually exclusive and there is a *builds-on* relationship. A software architecture, a product-line architecture and a reference architecture may build on several architectural styles in their designs. A software architecture and a product-line architecture in turn may use reference architectures. And finally, a software

architecture may be based on the design of a product-line architecture. Figure 4.1 shows the relationships of the different types of architectures.

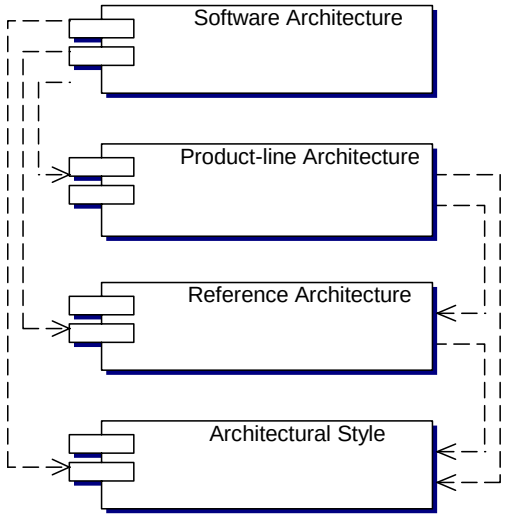


Figure 4.1.: Software architecture, product line architecture, reference architecture and architectural style

We define *framework* as an architecture plus a partial implementation. With this definition we can apply the attributes and the relationships of the different types of architectures to a corresponding *framework*. The exception is of course the (application-specific) software architecture because the architecture for a specific application is not intended for reuse by definition. For architectural styles there are frameworks that support the usage of this style, the *architectural style framework*. For a reference or domain-specific architecture there is a framework that implements parts of it and provides part of a solution. So we call it *solution domain framework*. And a product-line architecture corresponds to a framework that provides components and structures that can be reused in a set of applications in the same application domain. So we call *application domain framework*. In table 4.1 we compare the different types of frameworks.

We describe the following systems on each architectural level. For this study we focus on the solution domain, specifically the domain *Augmented Reality*. We examine which solutions exist and which attributes they have. Application specific components or domain specific components are described where they exist, but we do not go into detail. Additionally, we address the underlying architectural styles that describe the general underlying approaches, for example client/server communication or local method calls.

4.3. An Augmented Reality Reference Architecture

The study of the various Augmented Reality systems revealed that in spite of being quite different in detail, all of them share a common basic architectural structure. In addition, many basic components and subsystems can be found in many different systems, e.g. various trackers or a scene graph. Based upon this notion, we propose a reference architecture for Augmented

Term	Elements	Granularity of Elements	Application Specificity
Application	specific for a particular application	element specific, usually complexer	high
Application Domain Framework	for applications of the same type, e.g. maintenance	high	middle
Solution Domain Framework	building blocks of applications of the same technical domain, e.g. Augmented Reality	middle	low
Architectural Style Framework	building blocks for a technical concept, e.g. distributed peer-to-peer systems	fine granular	none

Table 4.1.: Comparison of framework types

Reality systems. This reference model is an abstract high-level model that gives us a vehicle to compare particular systems. It is not intended to be a "perfect solution" for all kind of AR applications—indeed, a system implementing the full reference architecture would probably be far too complex.

The reference model is at the solution domain level of abstraction, meaning that it addresses neither lower-level issues such as the operating system and middleware nor higher-level issues such as support for a particular application domain.

In the following sections, we will map particular AR systems onto this reference architecture in order to describe them consistently and make them comparable.

4.3.1. Subsystem Model

As a first step, we identified the core subsystems of Augmented Reality systems. Each of them provides a particular functionality for the whole system. In our study we made out the following core subsystems:

Application subsystem. An important issue of frameworks is the integration of application specific logic. We introduce an abstract application subsystem as a placeholder for all application specific logic.

Tracking subsystem. Tracking the user's pose is a key element of AR systems. An important issue is that the calculation of the pose must be done regularly in parallel to the other system tasks.

User input subsystem. The user input subsystem gathers and processes any input that the user makes deliberately. We distinguish it from other input such as by moving and changing the position.

User output subsystem. The user output system displays system output for the user. Besides 3D augmentation this also other means media such as 2D text or speech.

Context subsystem. The context subsystem collects different types of context data and makes it available to other subsystems.

World model subsystem. In AR the user moves in the real world and obtains information linked to real world objects or user positions. Information about the world around the user are stored in a world model.

Note that these subsystems comprise a "bare-bones" AR system, without regard to the integration of the system with other enterprise applications. Indeed, most AR research projects do not address this issue of how to retrieve the information to be displayed. The ARVIKA software architecture defines additional subsystems for access to enterprise applications and legacy data.

Each subsystem consists of several components. In Figure 4.2 we show the identified subsystems and the main components for each of them as a UML component diagram. There are dependency relationships between the subsystems, illustrated with dashed lines. A dependency shows that a subsystems relies on data and functionality of another.

4.3.2. Overview Class Model

In a next step we elaborated each subsystem. For an overview model see figure 4.3. The UML class diagram shows details of the reference model and the relationships between the components of the architecture. The connections between classes are annotated with labels that show what type of data is exchanged. To keep the complexity of the diagram reasonable, we left out some details and swapped them out to detailed diagrams of the individual subsystems.

Note that we kept the layout of the subsystem diagram in figure 4.2. We will keep the same layout when we describe the respective AR systems, to make clear which system implements which subsystems and components of the reference model.

In the following we describe each subsystem in more detail.

Application Subsystem

The Application subsystem consists of application specific code, configuration data, and content data. It provides the end user functionality and is responsible for the bootstrapping. An end user application may consist of several other sub-applications and components that also provide user functionality. Figure 4.4 illustrates the principal structure of that subsystem. This subsystem is not specific to Augmented Reality, but does communicate with other subsystems, particularly the tracking and rendering subsystems.

Tracking Subsystem

Tracking is one of the most important subsystems of Augmented Reality. Figure 4.5 is an abstract picture of the Tracking subsystem. Tracking can be achieved with several techniques. For example, there are video-based trackers, magnetic and inertial trackers, GPS, external trackers from the environment, or combinations thereof, i.e. hybrid trackers. The trackers can use the WorldModel to get information about the environment they are working in. For example, the VideoTracker needs information about the features it should look for in the images. The result of the tracking process, the pose, is forwarded to the ThreeDRendering component that updates the visualization and the ContextManager component for use in other subsystems.

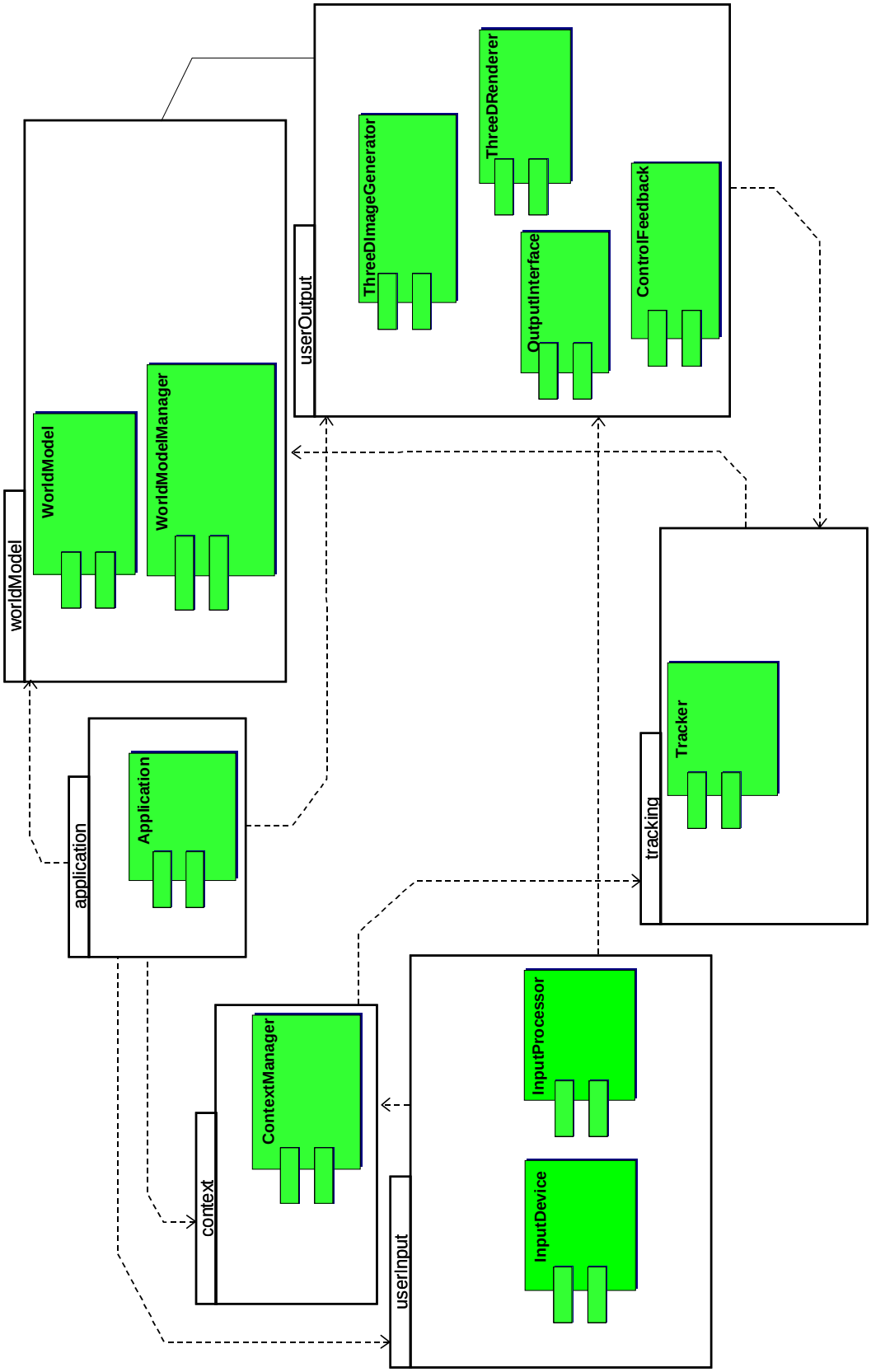


Figure 4.2.: Subsystem decomposition of reference model

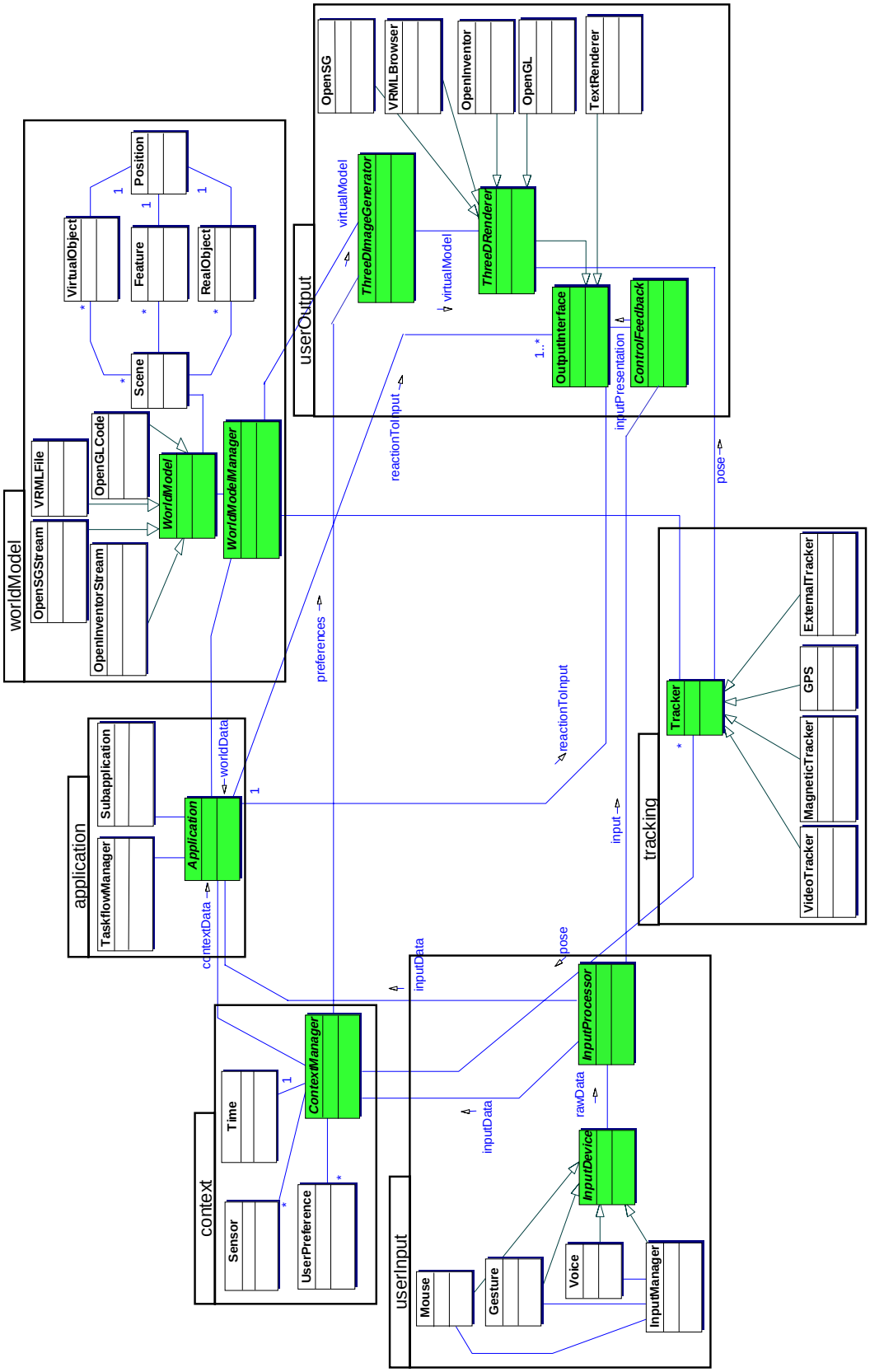


Figure 4.3.: Augmented Reality system reference architecture

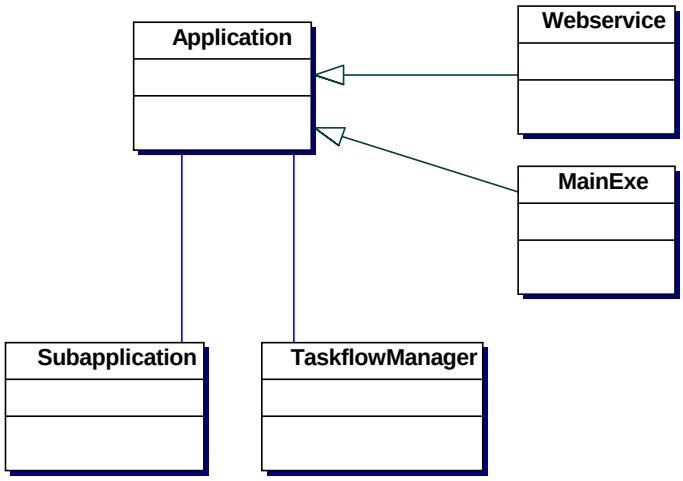


Figure 4.4.: Reference model application subsystem. The application subsystem is represented by the abstract Application class. A particular application can be implemented as a Web service, in a Main executable or in an other way. Also, an application can consist of sub-applications or other components such as a TaskflowManager.

User Input Subsystem

We understand as user input any input that the user consciously makes to control the system. Therefore input through the Tracking subsystem is not taken into account. Tracking the user's hand position becomes an input only after it was interpreted as a gesture. Figure 4.6 illustrates the structure of this subsystem. Generally user input can be achieved through several input devices, e.g. Mouse, Gestures, Voice or combinations thereof through an input manager component. The input data is forwarded to a input processor that can use data analyzing and modality fusion to interpret the input data. The input is then forwarded to the application and the context manager for further processing. For feedback the recognized input can be forwarded to the user output system, particularly to a ControlFeedback component that visualizes the input, e.g. movement of the mouse pointer.

User Output Subsystem

The User Output subsystem is the human-machine interface. User output can be achieved through several channels, as 3D graphics, as text, voice or sound. For Augmented Reality the graphical presentation is the most important one, but not the only one. Nevertheless, here we concentrate on 3D graphics output. Figure 4.7 illustrates this subsystem. Core of this system is the ThreeDRenderer component, an OutputInterface which draws the virtual objects that augment the user's perception in the correct distance, position and orientation based on the user's pose. The virtual objects are created by the ThreeDImageGenerator. This component adapts models from the WorldModel to context information from the ContextManager, e.g. needed level of detail. The user's pose comes from the Tracker. For rendering in 3D usually a SceneGraph component is used. If not only 3D object should be drawn, but also 2D data from a TextRenderer, a GraphicsMixer component integrates 3D and 2D data into one representation.

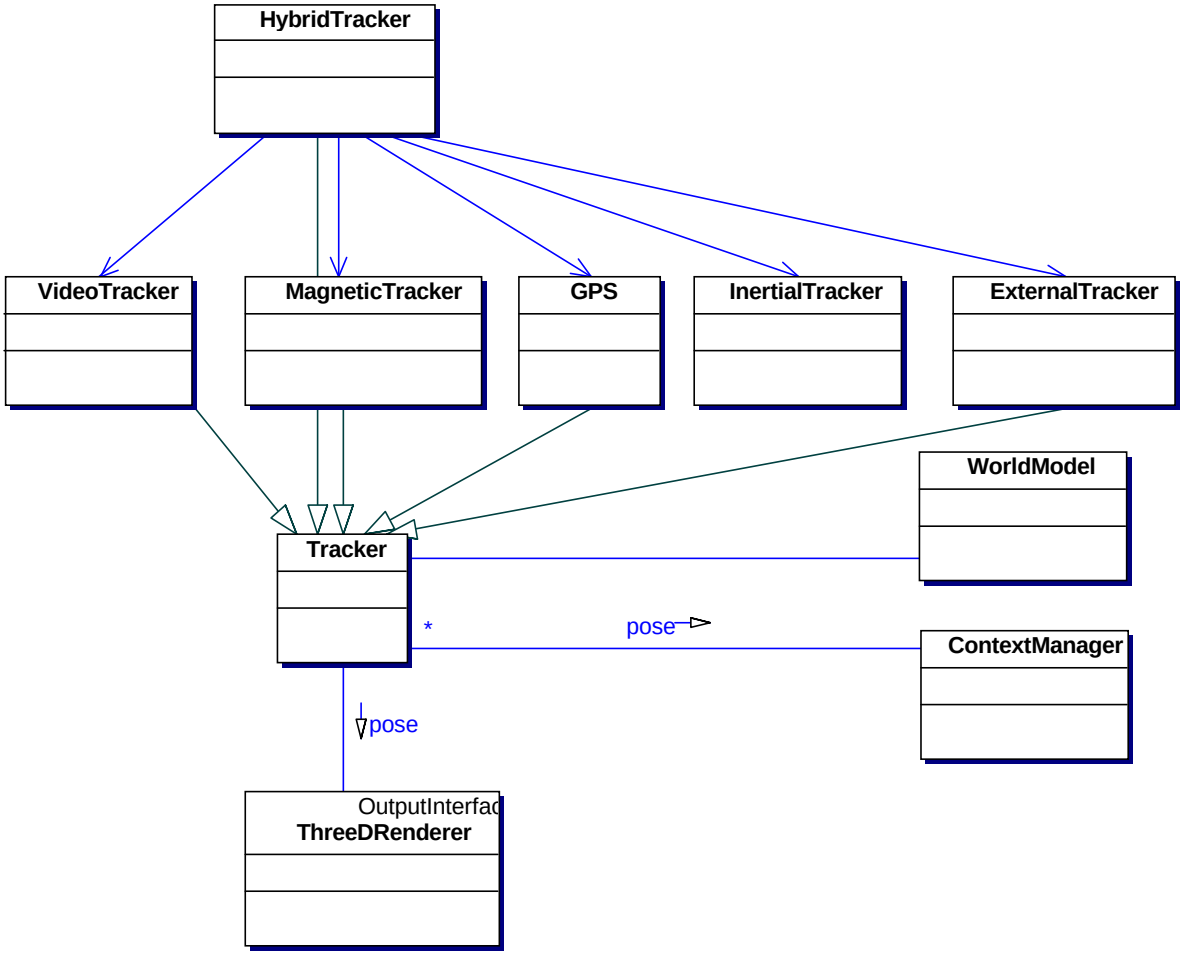


Figure 4.5.: Reference model tracking subsystem. A Tracker component can be a VideoTracker, a MagneticTracker, an InertialTracker, an ExternalTracker, GPS or a HybridTracker. This is not a complete enumeration but gives examples for tracking systems. For the initialization the trackers need information from the WorldModel. The result of the tracking process, the pose, is forwarded to the ThreeDRendering component that updates the visualization and the ContextManager component for use in other subsystems.

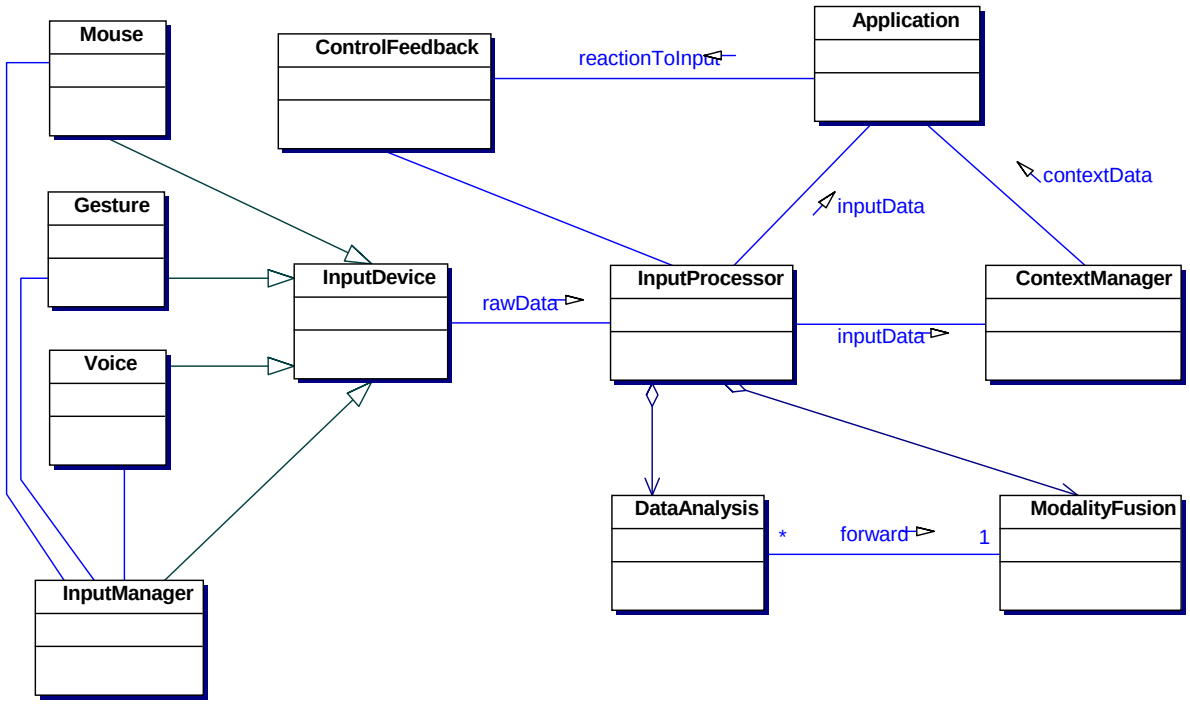


Figure 4.6.: Reference model user input subsystem. Input data are created by an InputDevice, specifically Mouse input, Gesture, Voice or combinations thereof from an InputManager that combines several input devices to an abstract input device. An InputProcessor further processes the input data with DataAnalysis and ModalityFusion and forwards it to the Application, the ContextManager and the ControlFeedback component. The ControlFeedback component is part of the User Output subsystem and visualizes the recognized input.

The output data is then forwarded to a VideoMixer, which may be needed to mix real world videos with virtual images. A VideoMixer is used for video see-through Augmented Reality. The mixed images are then emitted by an OutputDevice.

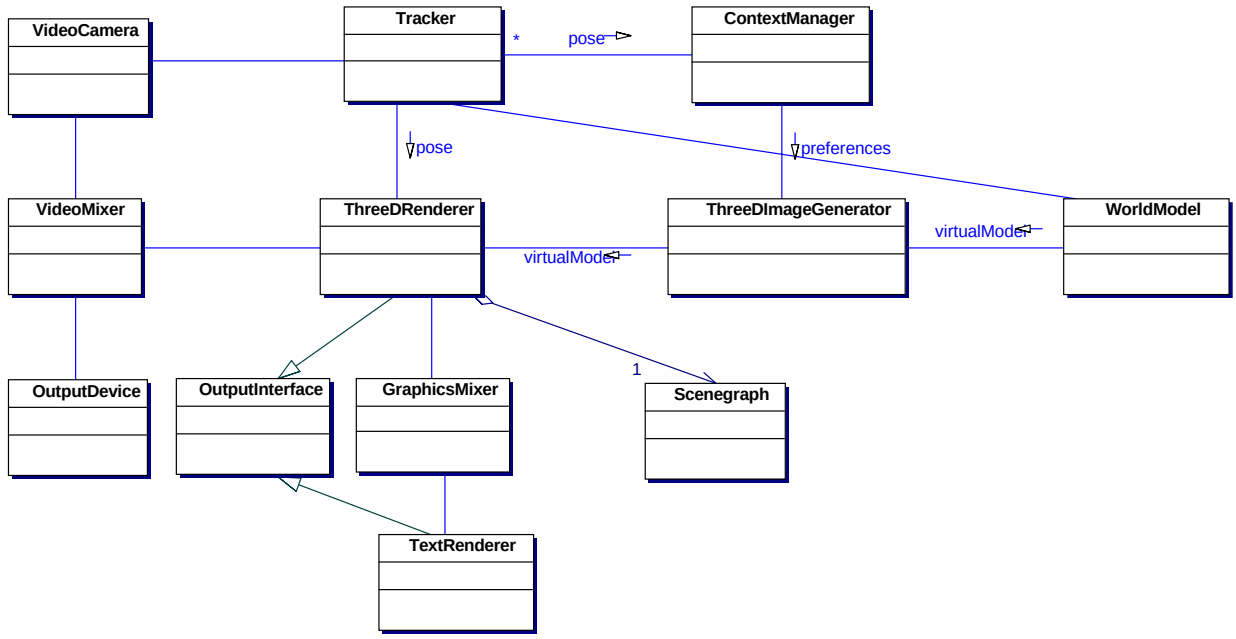


Figure 4.7.: Reference model graphics output subsystem. The core is the ThreeDRenderer, an OutputInterface for 3D graphics. Images are generated by a ThreeDImageGenerator, which adapts virtual objects from the WorldModel to the context. The ThreeDRenderer can use a GraphicsMixer to include 2D graphics. Trackers send the pose to the ThreeDRenderer to enable an update of the view. For video see-through a VideoMixer overlays images from the VideoCamera and the images of the ThreeDRenderer. For optical see-through, this component is not needed. In both cases, the images are emitted by an OutputDevice which might be a head-mounted display (HMD).

Context Subsystem

Modeling the user's context is still a field of research. For different applications, different types of information about the user's context are needed. Therefore we model the Context subsystem as an abstract system with a ContextManager component that collects different types of context information, e.g. user preferences, sensor data, the time, tracking data, resource information, domain knowledge or knowledge of the user's current situation. The ContextManager stores this information and makes it accessible to other components. ContextProcessors may read context information, process it and generate new context information. Figure 4.8 illustrates the Context subsystem.

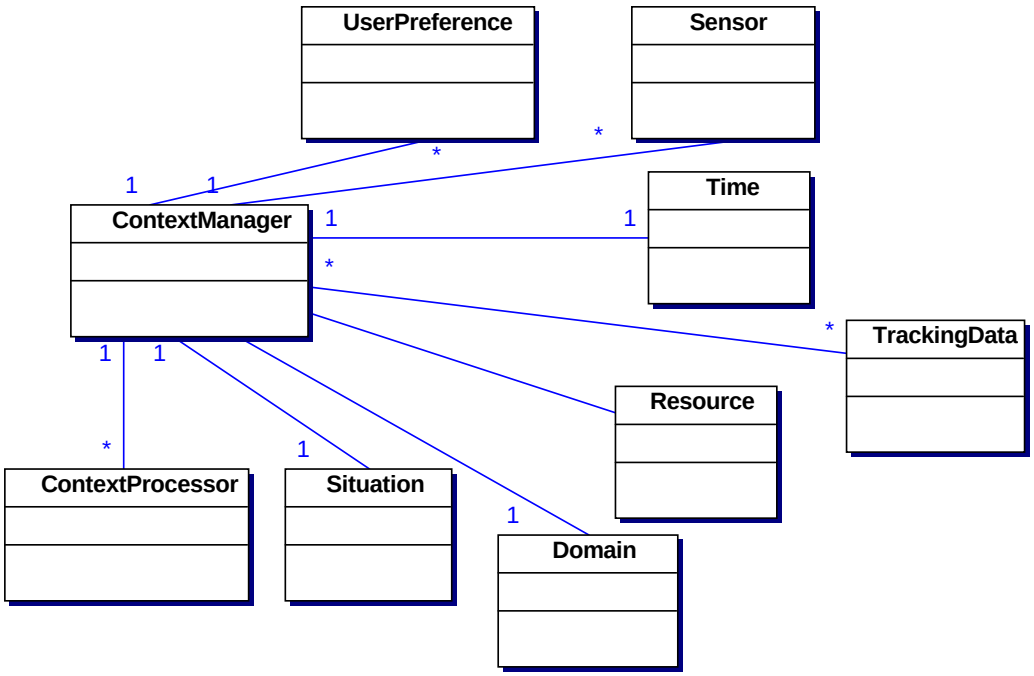


Figure 4.8.: Reference model context subsystem. Core of this subsystem is the ContextManager. It collects different types of context information, e.g. user preferences, sensor data, the time, tracking data, resource information, domain knowledge or knowledge of the user's current situation. The ContextManager stores this information and makes it accessible for other components. ContextProcessors may read context information, process it and generate new context information.

World Model Subsystem

The World Model subsystem (figure 4.9) stores and provides information about the world around the user. It may be represented in various formats such as a VRML model, with OpenGL code, an OpenInventor stream. At runtime a world model manager controls the access to the world model. A world model consists of scenes, a collection of virtual objects, feature information for trackers, and representations of real world objects. A common important attribute of these objects is their position registered to a common world coordinate system.

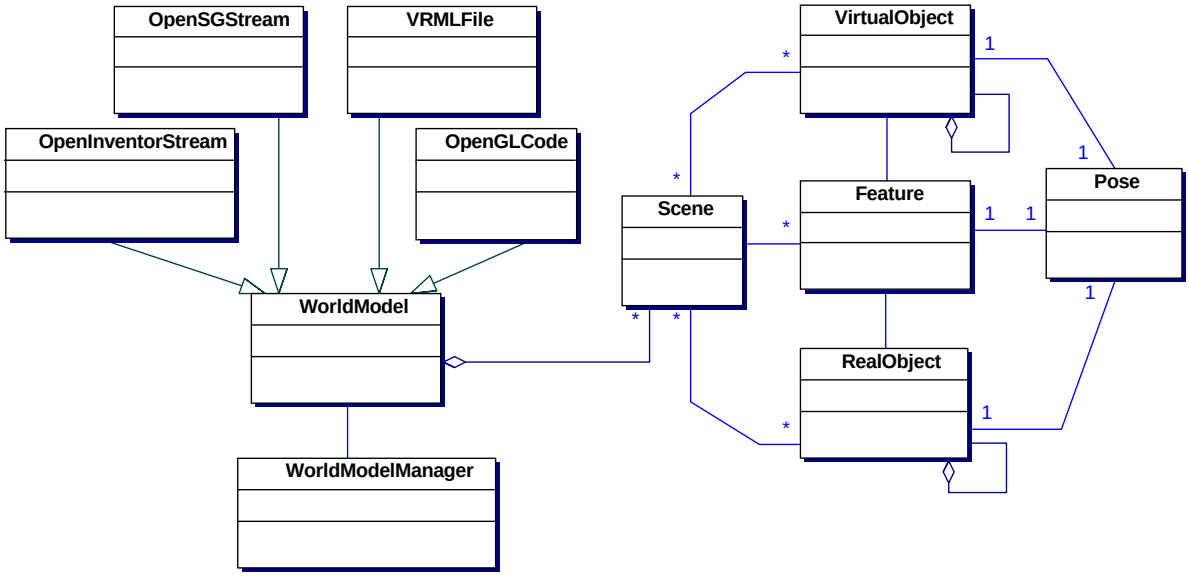


Figure 4.9.: Reference model world model subsystem. The actual WorldModel can be a VRMLFile, OpenGLCode, OpenInventorStream or OpenSG, to name some. A WorldModel consists of Scene objects, in each scene are VirtualObjects, Features, and RealObjects. Each of them as a Position in the coordinate system of the WorldModel. A WorldModelManager controls the access to the WorldModel.

4.4. System Descriptions

In this section we describe systems in more detail that showed to be most relevant for this study. Specifically, we describe ARVIKA, DWARF, the MR Platform, STAR, and UbiCom. These software architectures are are more or less representative; many other projects use architectures that are similar to one of these five, or simpler architectures restricted to individual AR subproblems. A detailed analysis of each such architecture was beyond the scope of this study.

For ARVIKA and DWARF, we have enough architectural information to describe the components on all four levels and give a mapping to the reference architecture. MR Platform, STAR and UbiCom are described only informally; a more formal description would be possible with more detailed information on the respective software architectures.

4.4.1. ARVIKA

The ARVIKA architecture is component-oriented and is used to build two different solutions: a high-end solution for lab environments and a mobile solution in a web-based environment. The high-end solution can be deployed on Windows 2000/NT, Linux, and SGI IRIX platforms, whereas the mobile solution runs on Windows 2000 and WindowsCE.

By reusing code, interfaces and protocols, synergy effects between the two solutions can be achieved.

Stationary high-end solution

The stationary high-end solution is intended for lab environments where high accuracy of tracking is mandatory.

All components of the high-end solution run on one system, either on Windows 2000/NT, Linux, or SGI IRIX. On top of the operating system are the device integration interface IDEAL, the AR browser, and the tracking component. On top of these AR specific components is the application-specific software. Figure 4.10 illustrates the subsystems in a deployment diagram.

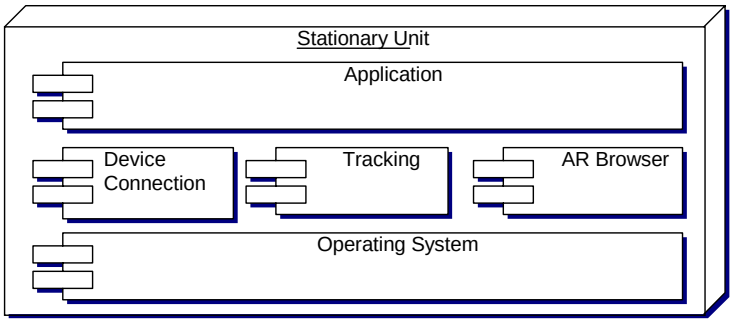


Figure 4.10.: Deployment of ARVIKA stationary solution

IDEAL allows network-wide access to various tracking and interaction devices, using a system of servers and daemons A demon listens for device requests of the AR Browser and starts a new server for the requested device on demand. The AR browser then connects to the server via a network sockets A common interface, the Low Level Device Interface (LLDI), provides an abstract model of different devices.

The whole system runs on one physical machine, except for some tracking devices that are connected by socket interfaces. Thus, the underlying architectural styles are local method calls and socket interfaces.

Mobile web-based solution

The overall architectural concept for ARVIKA is data-centered. Documents with Augmented Reality content are one type of media among others (HTML documents, PDF documents, or CAD data). For this type of application, web-based solutions are common.

The client-side platform is targeted at Windows 2000 and Windows CE and provides the typical AR functionality with localization, tracking, graphics, and human-computer interaction techniques with the integration of interaction devices. Technically, the client environment is

the Microsoft Internet Explorer 5 driven in kiosk mode, i.e. the client is a so-called thin client with a web browser as the interface. The content sub-window is extended to the complete user screen and hides the usual web-browser interface. This allows the full control over the user interaction. The AR Browser needs access to operating system resources, e.g. access to the video camera. So basically, the actual Augmented Reality subsystem (tracking and 3D visualization) is a local component with a wrapper around it to make it an ActiveX component. ActiveX is a component model defined by Microsoft and allows to embed new components into web-browsers. When AR scenes are reference in web pages, the AR Browser is started and displays the registrated scene.

The server side handles information management. It is based on the Apache Tomcat application server. Each component is realized following the Java Beans specification. Each Bean has a Java Servlet interface as a wrapper that provides a HTTP-based access. Internally the components ought to use the Beans interface. For server-independent use the typically server-side components can be deployed on the client platform.

Figure 4.11 illustrates the ARVIKA web-based system with its subsystems and the deployment on client and server.

Figure 4.12 shows the ARVIKA architecture mapped onto the reference architecture. Note that only those components that are central to AR are included—most of the web services, e.g. for legacy application integration, are left out.

The mapping shows that ARVIKA covers each subsystem of the reference model.

Client. When a user loads a document with AR support, the AR subsystem is automatically loaded and the document is shown in a special browser, the AR Browser. The Windows-based AR Browser reuses code from the stationary solution, with an additional ActiveX interface.

Although it is a web-based solution were every component could be downloaded and installed via web-server, due to the components' size some of them should to be installed locally by the user. These components are Localization, Video Server, optical tracking and other tracking (Intersense, hybrid tracking, etc.), and the IDEAL device integration interface.

Following the web-based concept, different types of documents must be viewed with viewer plug-ins for the Microsoft Internet Explorer. Examples are a PDF viewer and a viewer for CAD documents. A second ActiveX control was realized for net collaboration.

The following components have been developed on the client side:

AR Browser. Bases on a virtual reality system that uses OpenSG for rendering; manages the animation of images and integration of tracking and interaction devices into the scene graph.

NetCollaboration. Builds on AR Browser, the client side of NetCollaboration, allows communication between local technician and remote expert. The technician's current situation is transmitted as video stream to the expert who can augment the video with help information during a maintenance session.

Collaborative AR. Manages the information exchange between multiple ARVIKA client systems. NetCollaboration and Collaborative AR are based on a common framework for video and augmentation exchange called ARMIN.

Localization. For the determination of the user's current location. The Localization allows information adaption and preselection based on the current user position.

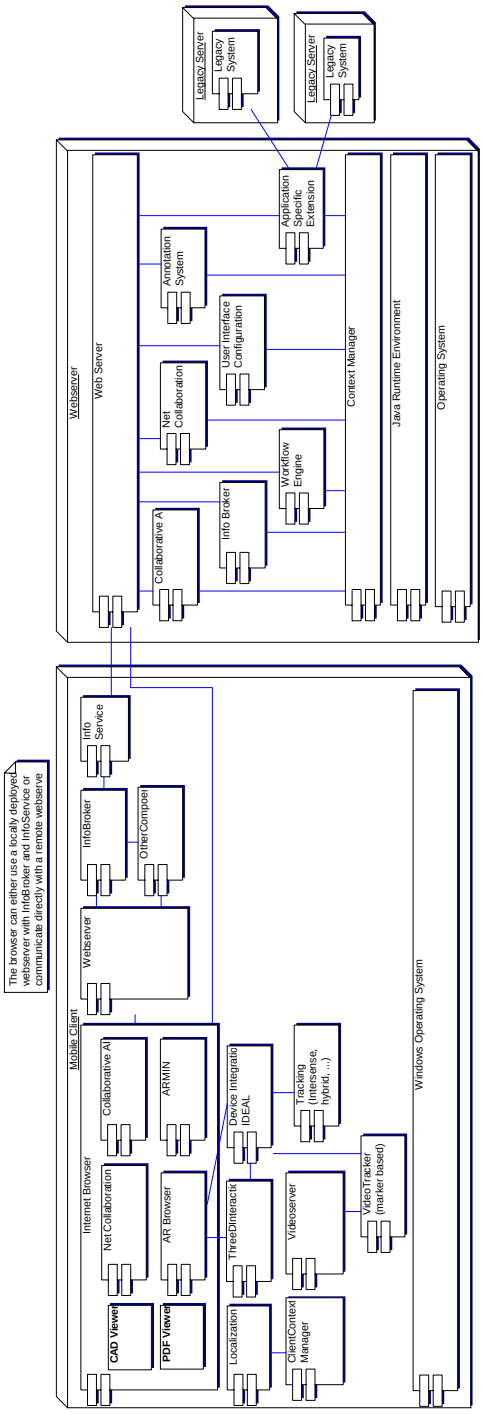


Figure 4.11.: Deployment of ARVIKA web-based solution

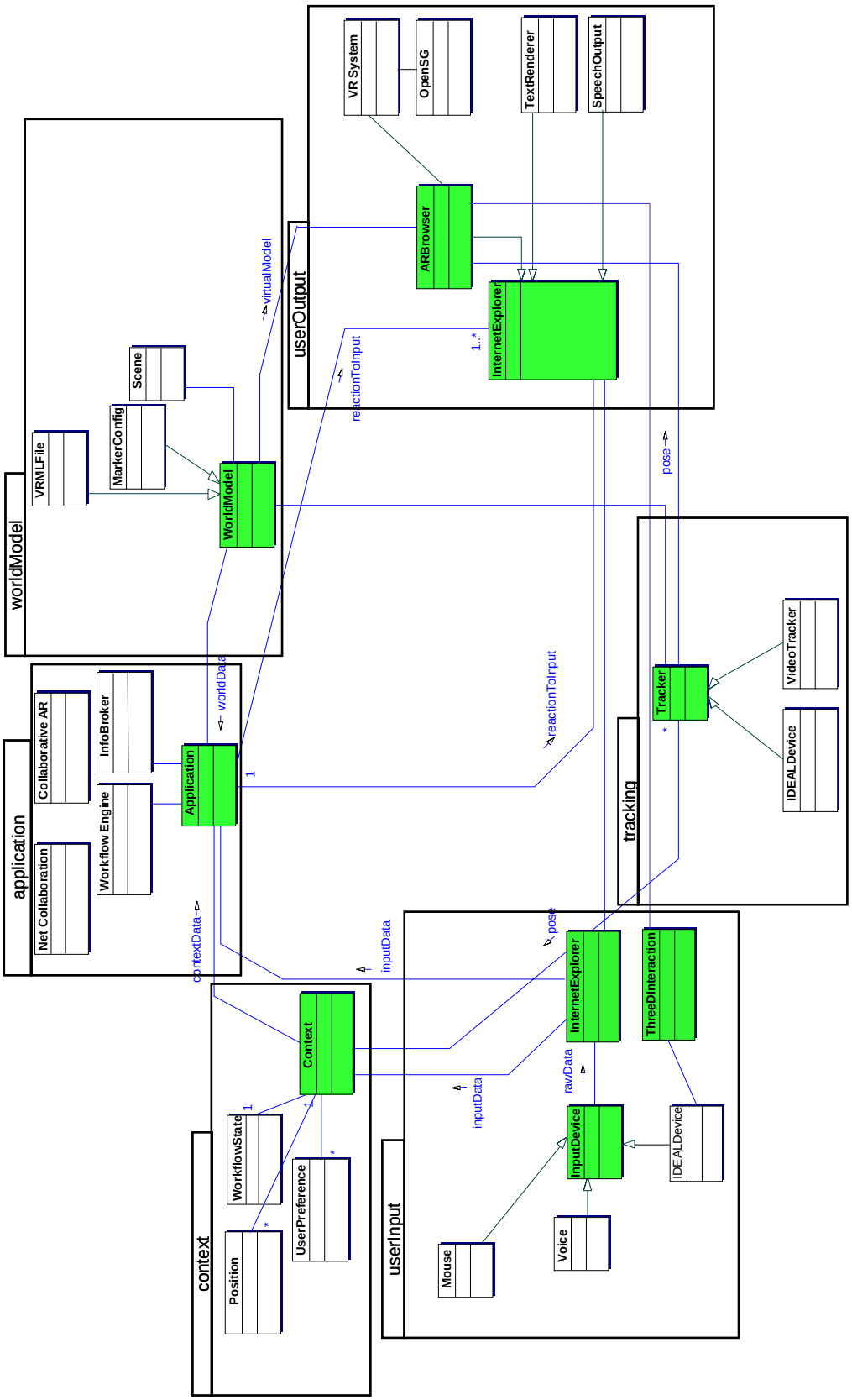


Figure 4.12.: ARVIKA architecture mapped onto reference architecture

Tracking. For the determination of the user's pose.

Video Server. Delivers the video images gathered by a video camera. The same image can be streamed over the network for Net Collaboration.

Device Integration Interface IDEAL. Provides a localization and instance independent interfaces for the integration and management of hardware devices such as trackers and input devices.

3D Interaction. User input in a 3D world, for example through a space mouse.

Context Manager. The client part of the Context Manager; provides context information to client and server side components.

InfoService. Acts as proxy server for server side components, e.g. InfoBroker.

Server. For the server side ARVIKA chose a standardized web-based solution with Java Servlets and Java Server Pages (JSPs). The implementation was done with Java 2 Standard Edition and the Apache Tomcat 4 Servlet engine. It can be deployed on different platforms and application servers (Linux/Windows 2000, Apache/Microsoft Information Server).

The following services have been developed:

Context Manager. The server side of the Context Manager; exchanges context information between server side components and information push from server-side components to client-side compoents.

InfoBroker. Provides a document model based on the structure of machines (called *Information Space*). The actual data are read from databases, files or third-party systems as information sources.

Workflow Engine. Workflow models describe an AR supported process, e.g. checking a machine. The Workflow Engine processes workflow descriptions.

NetCollaboration. Server part of NetCollaboration; displays videos from the client side to a remote expert which can annotate the image.

UI Configuration. Adaptation of the information to be displayed to the user context, e.g. the device.

Annotation System. Component to gather user annotations and save it.

Each component is working on content and configuration specified using XML formats.

The individual components are not directly connected. There is an indirect connection via the Context Manager that collects and distributes messages generated by the components to other interested components.

Architectural Analysis.

The ARVIKA components can be classified into the four levels.

Application level. Several applications from maintenance, service, and production.

Application domain level. InfoBroker, Workflow Engine, NetCollaboration, and Annotation System are not AR specific but specific to the application domain of maintenance and service systems.

Solution domain level. Reusable components for AR systems are Localization, Context Manager, UI Configuration, Device Integration Interface IDEAL, Video Server, Tracking and 3DInteraction.

Architectural style level. The overall structure of the ARVIKA system is web-based. The typical Augmented Reality subsystems (tracking and 3D rendering) are running locally on the client system. Via the IDEAL device interface various devices can be connected locally or remotely via sockets.

4.4.2. DWARF

DWARF is a representative of peer-to-peer systems. Peer-to-peer systems are an architectural approach towards integrating systems deployed on wearable computers into a ubiquitous computing environment.

Architecture. The DWARF framework has three basic aspects: services, middleware and architecture. First, DWARF consists of software services such as trackers, running on hardware modules. Each service provides certain abilities to the user or to other services. Second, DWARF contains the middleware necessary to match these services dynamically and set up the communication between them, so that the system configuration can change at runtime. Third, the conceptual architecture of DWARF describes the basic structure of AR systems that can be built with it. This ensures that service developers agree on the roles of their own services within the system and on interfaces between them. The architecture is also easy enough for end users to understand, so that they can reconfigure their wearable system by simply plugging together the appropriate hardware modules.

Functionally, the services within the framework can be divided into four areas: modeling the world and things in it; accessing information; interacting with the user; and accessing external (non-DWARF) services. The services currently developed in these areas, together with an example application for indoor and outdoor navigation, are shown in Figure 4.13.

The application is shielded from the low-level services, such as for user interface or tracking hardware, and accesses these at a higher level of abstraction using the various DWARF services. It includes a special service which provides bootstrapping functionality and "glue logic". This provides the other services with models of the world and of tasks the user wishes to perform.

Connections for communication among the framework and application services are set up automatically by the middleware, and many of these bypass the application entirely. Thus, an optical tracker can provide position data to the head-mounted display, since it has an ability of type 'PositionData' which matches a need of the display service.

Figure 4.14 shows the DWARF architecture mapped onto the reference architecture.

The diagram shows that DWARF covers all subsystems of the reference model except the Context subsystem.

Modeling the application. The first DWARF demonstration system, Pathfinder, implements an indoor and outdoor navigation scenario. Another application, TRAMP, has been built since.

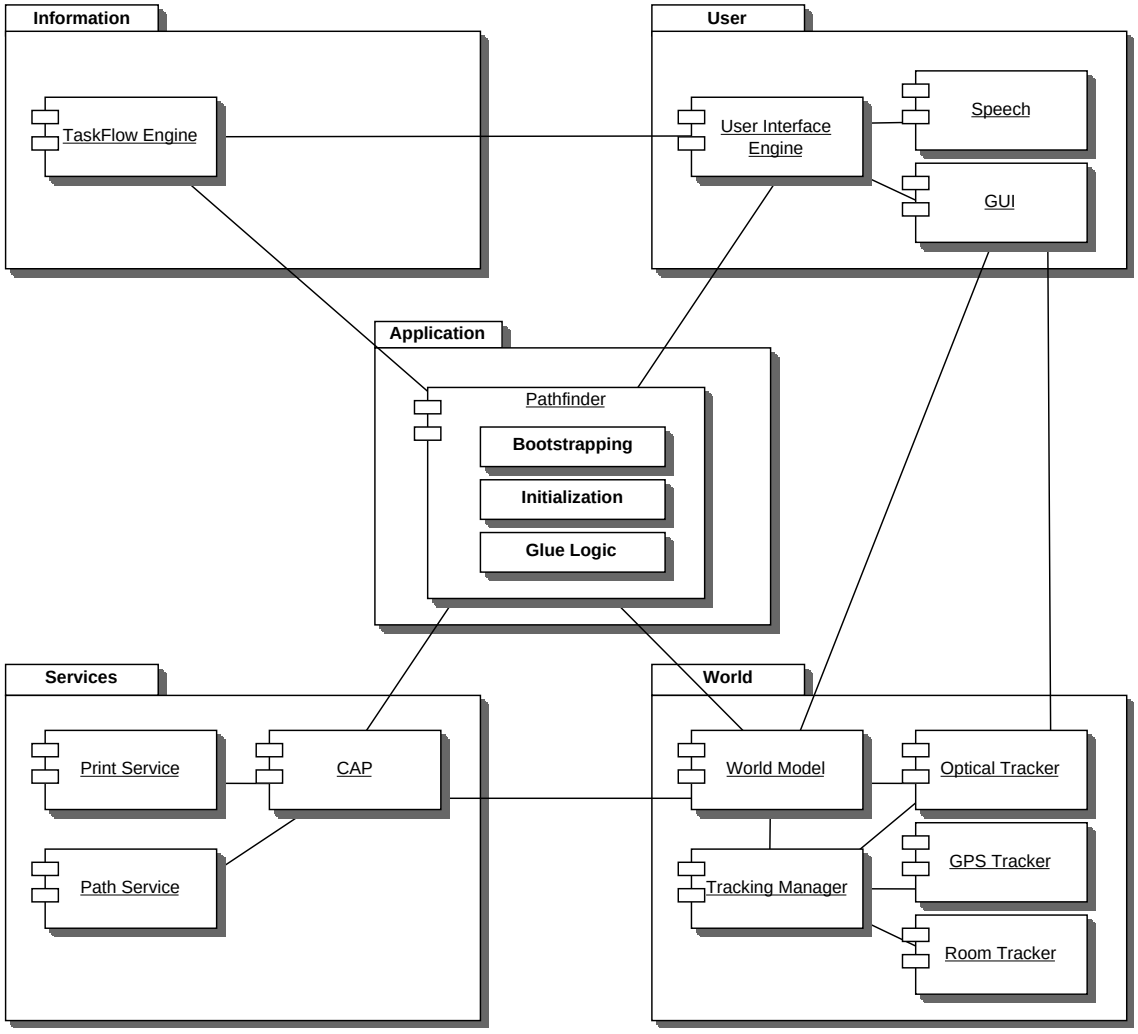


Figure 4.13.: Services of the DWARF framework

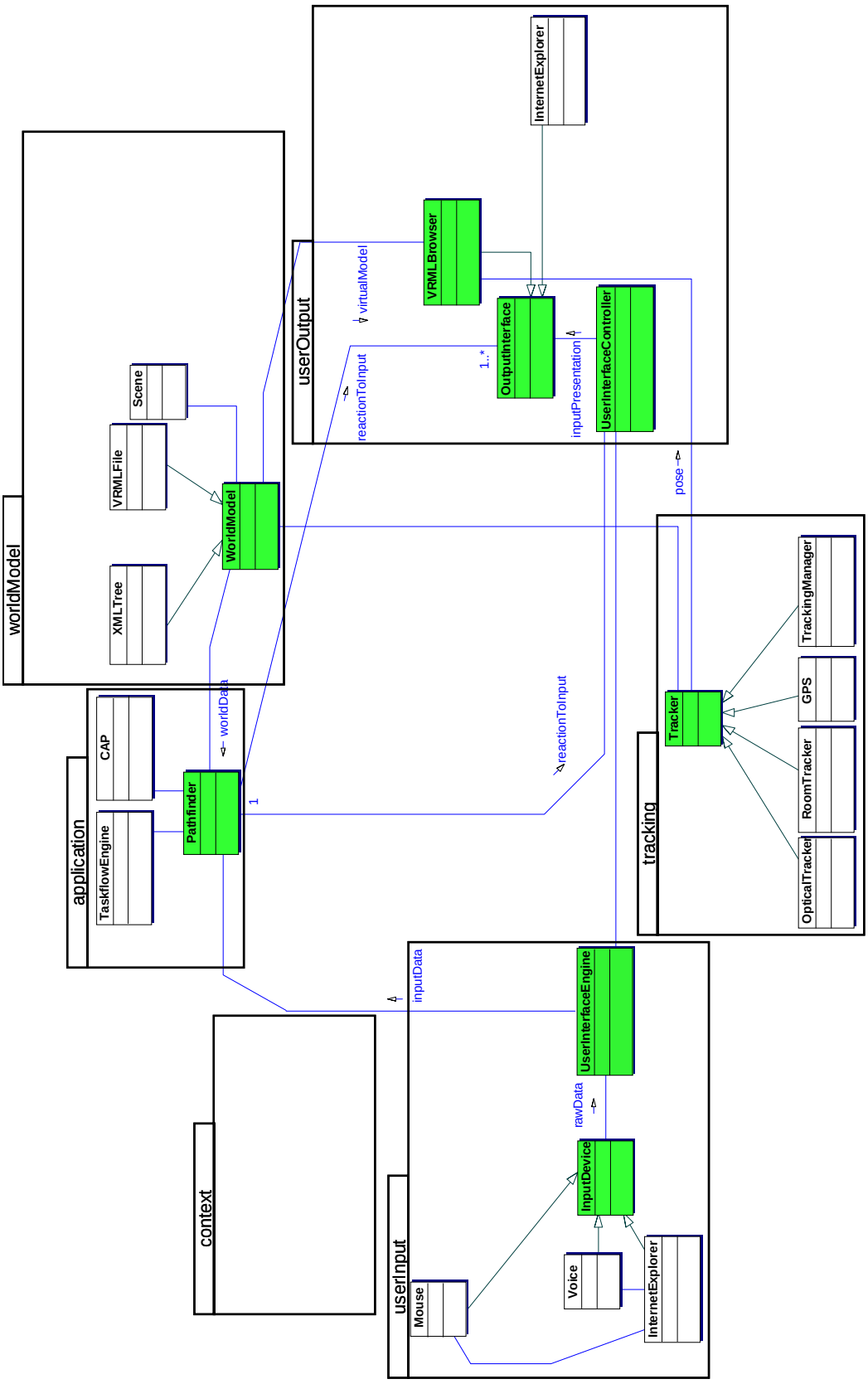


Figure 4.14.: DWARF architecture mapped onto reference architecture

Most of the application could be modeled using high-level services such as the world model (for geographical information) and the taskflow engine (for step-by step navigation instructions). Still, some of the application's logic has to be programmed separately.

One key responsibility of the application in DWARF is bootstrapping. Here, the application has to ensure that the taskflow, world model and user interface descriptions for navigation were correctly sent to the high-level DWARF services.

Once the system is started, the application provides glue logic. This involves, for example, translating the position data from the tracking subsystem into more high-level events that were significant for the taskflow, such as "entering room 3175". To accomplish this, the application uses the representation of the rooms in the world model.

Deployment. The first prototype uses standard PC laptops mounted on a fixed frame backpack. Successive prototypes have additionally used palm-sized devices worn on the arm; future miniaturization is planned.

DWARF runs on a variety of platforms, including C++ and Java on Linux for Intel, PowerPC and StrongARM, Windows NT, 2000 and XP, and Mac OS X. The first demonstration system was deployed on two laptops running Windows, connected with standard ethernet cables.

Architectural Analysis.

The DWARF components can be classified into the four layers.

Application level. Applications such as Pathfinder or TRAMP.

Application domain level. Navigation, CAP.

Solution domain level. Reusable components for AR systems are WorldModel, TrackingManager, OpticalTracker, GPSTracker, RoomTracker, UserInterfaceEngine, TaskflowEngine.

Architectural style level. DWARF uses several styles depending on the components. The underlying overall low-level structure is that of a distributed ad hoc peer-to-peer system. Each component uses other peer components and provides services to other components. Once the connection between components is established each component plays a role in that connection. So components can communicate, for example, in a client/server style, as connected pipes, via events or using shared memory.

4.4.3. MR Platform

For the description of the MR Platform we refer to [UTS+02] (courtesy of Hiroyuki Yamamoto, Canon MR Labs). The MR Platform is an environment for research and development of mixed and Augmented Reality and is based on the results of the Japanese Mixed Reality project. It provides a video see-through head-mounted display and a software development kit which includes libraries for registration and video mixture. Functionality such as user interaction, inclusion of context knowledge and the visualization of virtual objects are out of the scope of this toolkit. Additionally the toolkit provides tools for sensor and camera calibration.

According to our reference AR subsystem decomposition, the MR Platform only provides the tracking, part of the userOutput and the worldModel subsystems.

The userOutput subsystem in the SDK only contains the rendering of the real world image for the video see-through head-mounted display. For this visualization the SDK uses low-level

OpenGL drawing instructions. The visualization of virtual objects is not part of the platform. There are many computer graphics libraries such as OpenSG, OpenInventor or OpenGL that can be connected with the MR Platform.

Figure 4.15 illustrates the design of the MR Platform.

The classes with the *Mrp* prefix are real classes from the MR Platform kit. The classes of the lower part of the diagram without the *Mrp* prefix are abstract classes chosen by us to show the reference functionality of the corresponding MR Platform class.

Figure 4.16 shows the data flow within the toolkit.

The MR Platform SDK is implemented as C++ class library on top of the Linux operating system and the Video4Linux image capturing library. Figure 4.17 illustrates the position of the MR Platform library in the context of the layer classification of MR Platform applications [UTS+02].

The tracking devices, magnetic tracker and video camera are connected through local method calls. The update of the sensor and the video image is done by calling an update method. Usually this is done within a loop. Within each cycle, the user's position is calculated and the new image for the HMD is rendered. The processing of the current position is left to the application developer. For example, in the RV-Border Guards [Osh99] multiplayer game the user position is transmitted to the common states server that manages the common state of the game and the position of all players.

Due to the fact that the tracking-update-render loop must be called from within the application, the application developer remains responsible for the control flow.

To embed the MR Platform into custom applications, a developer must manipulate the OpenGL output window, where the MR Platform renderer displays the images of the video cameras for video see-through. There are no further restrictions.

4.4.4. STAR

STAR consists of many different tools for the production of AR content, but for this survey, we only consider the run-time AR application. The STAR Augmented Reality system is a light-weight but complete Augmented Reality system for the visualization of a sequence of maintenance tasks. Except for context, each subsystem of our augmented reality reference architecture can be found.

The STAR runtime system uses a thin-client concept. Figure 4.18 gives an overview on the system components of a STAR system.

On the client, a video camera captures the image, the image is encoded, packed and transferred to the server. The server unpacks the image, performs image processing, calculates the camera position, and augments the image with virtual objects. The augmented image is again encoded, packed, and transferred to the client. The client unpacks, decodes and draws the image. This process is illustrated in Figure 4.19

Contrary to ARVIKA, where tracking is done on the client system, STAR uses an approach where the complete tracking process is executed on the server. This enables thin clients with very low computing demands but pays with latency because of the video transfer.

4.4.5. UbiCom

The UbiCom project identified three important constraints for AR for mobile users in a ubiquitous communication system: **low power** on the wearable system, a **system approach**

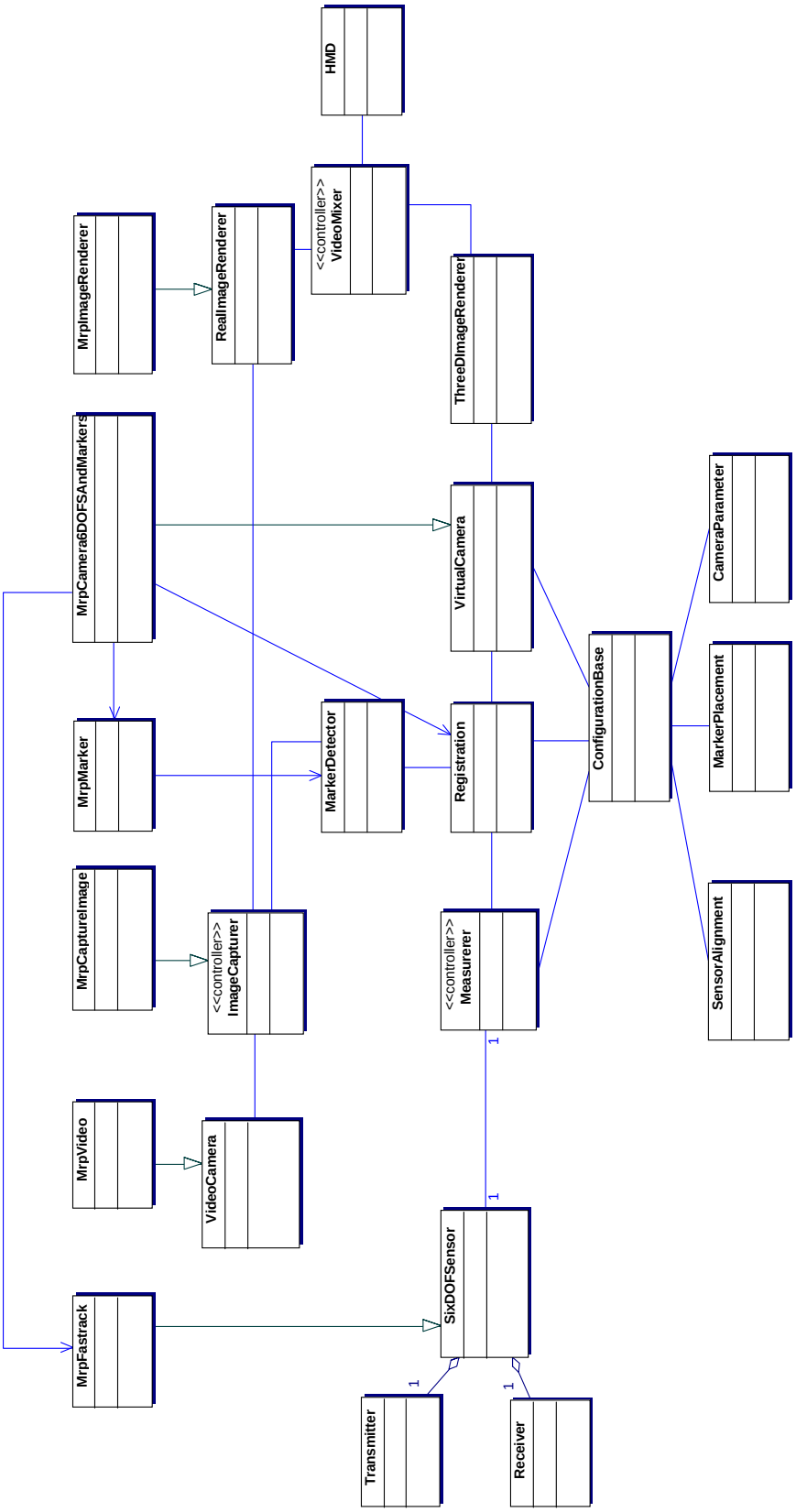


Figure 4.15.: Class diagram of the MR Platform Tracking subsystem

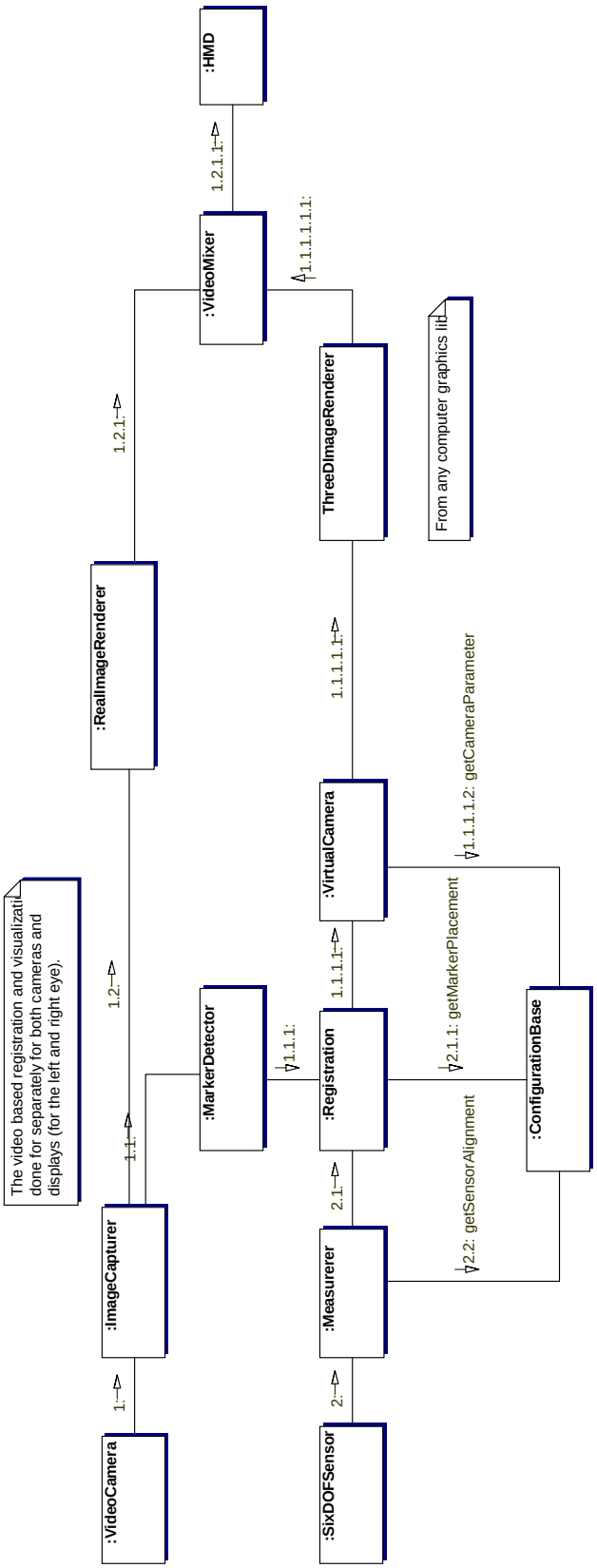


Figure 4.16.: Collaboration within the MR Platform Tracking subsystem

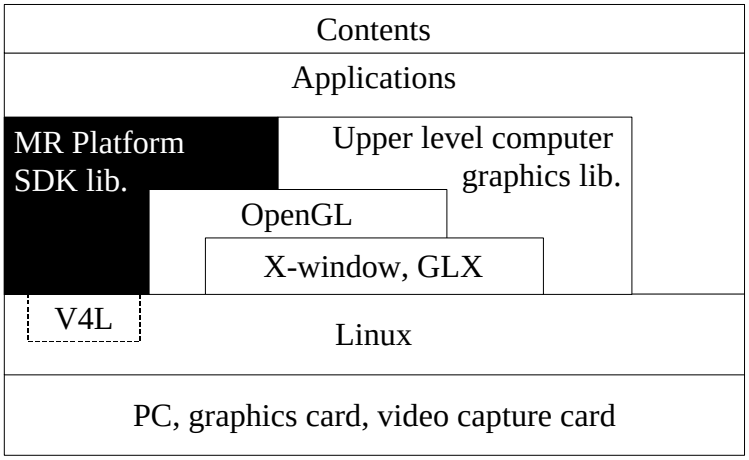


Figure 4.17.: AR library layers seen by the MR Platform

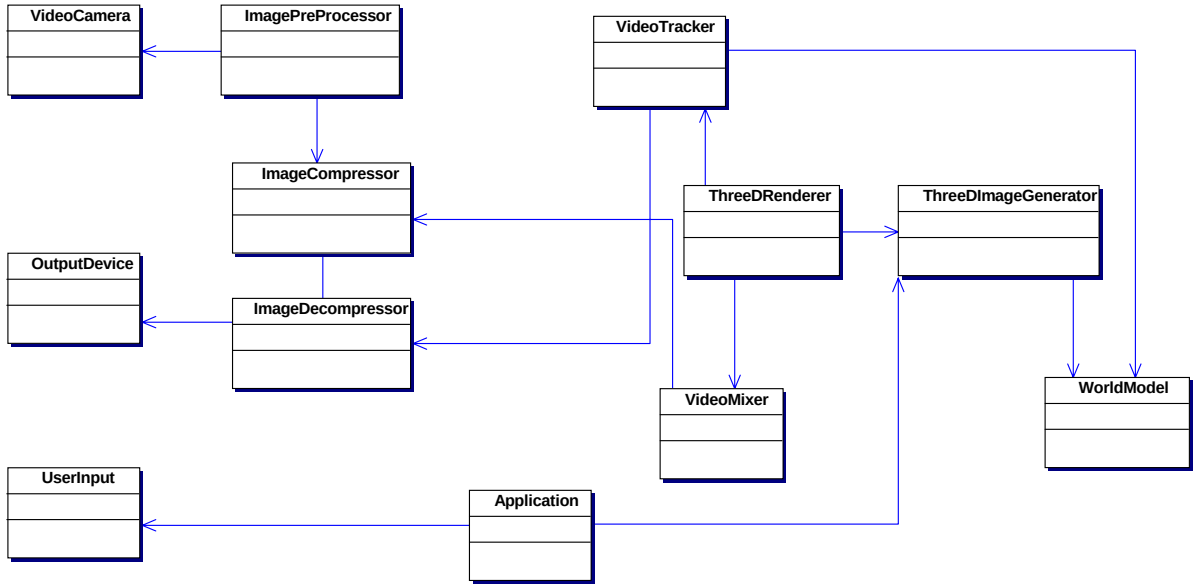


Figure 4.18.: STAR system architecture

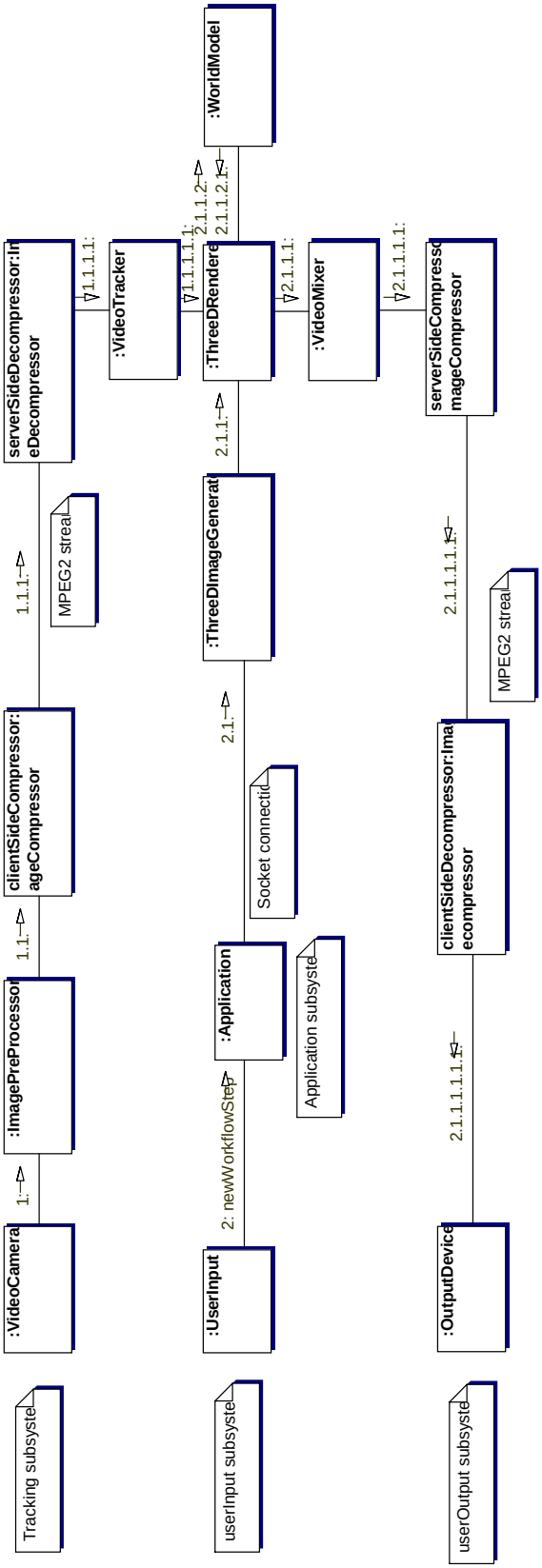


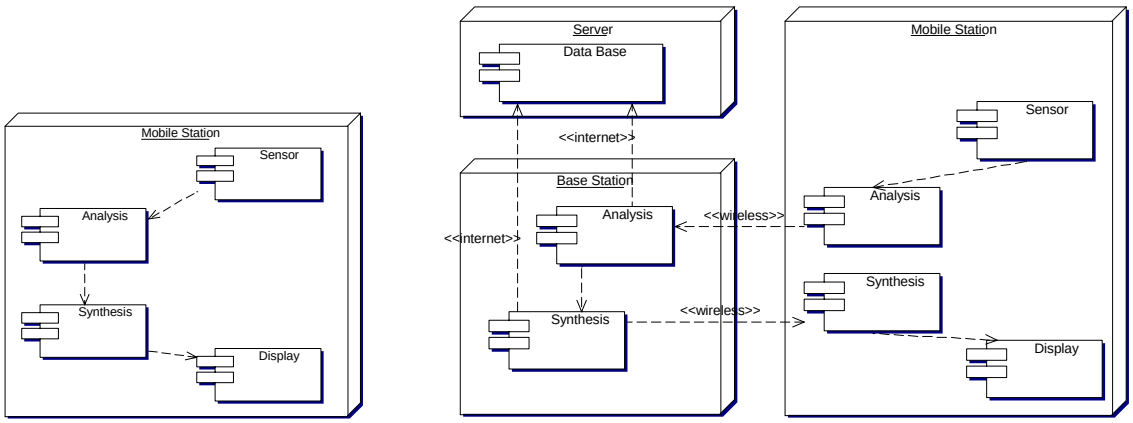
Figure 4.19.: STAR data flow

(carefully interfaced components), and **negotiated quality of service** in throughput, delay, and power consumption. To satisfy these constraints, the Ubiquitous Communications program has to address the scientific challenge of specifying an overall system that is flexible, adaptive, and scalable at all levels.

From an architectural point of view, UbiCom systems have several properties in common. They are data driven with occasional feedback control, and they allow for local quality of service negotiations. These common properties are used to restrict the class of possible architectures, i.e. to restrict the design space for the UbiCom wearable system and the overall UbiCom architecture (including the ubiquitous computing resources in the backbone). The architecture template recognizes three types of resources: communication, storage, and processing. The following list summarizes the most important architecture features:

- I/O centric rather than CPU centric,
- support for high-speed asynchronous data flow (switched or dedicated point-to-point),
- low-speed control bus,
- democratic processing (no single global control),
- asymmetric processing nodes, that is, a heterogeneous system with application specific hardware,
- power-down modes.

These features resulted to an approach where the general loop of video gathering – image analysis – video synthesis – display is distributed in parts done by the mobile client and parts done by the background environment. Figure 4.20 gives an overview of a) the general processing loop and b) the distributed UbiCom approach [Lag00].



(a) General approach

(b) UbiCom approach

Figure 4.20.: General and UbiCom approach for the AR processing chain

Figure 4.21 shows a UML deployment diagram that illustrates that the mobile client platform employs a distributed peer-to-peer approach. The interesting point is that the software components are distributed on a set of hardware modules.

The overall client system consists of five hardware modules. Each of them has a SA 1100 computing unit to execute the Linux operating system and function specific code. The particular functions sorted by modules are *Positioning*, *Rendering and Display*, *Video and Application*, *Wireless Connection*, and *Interconnect*. Additionally to the SA 1100 unit each modules has function specific hardware components. The Positioning module has a GPS receiver, a GPS antenna and Gyroscope for obtaining the user's position and orientation, the Video and Application module has a Camera MTV-5366 module to connect a video camera and a MPEG 2 encoder to stream video to the backbone unit (for video-based tracking). Analogously is the construction of the Rendering and Display module and the Wireless Connection module. The Interconnect module serves as the data exchange backbone for the connected modules.

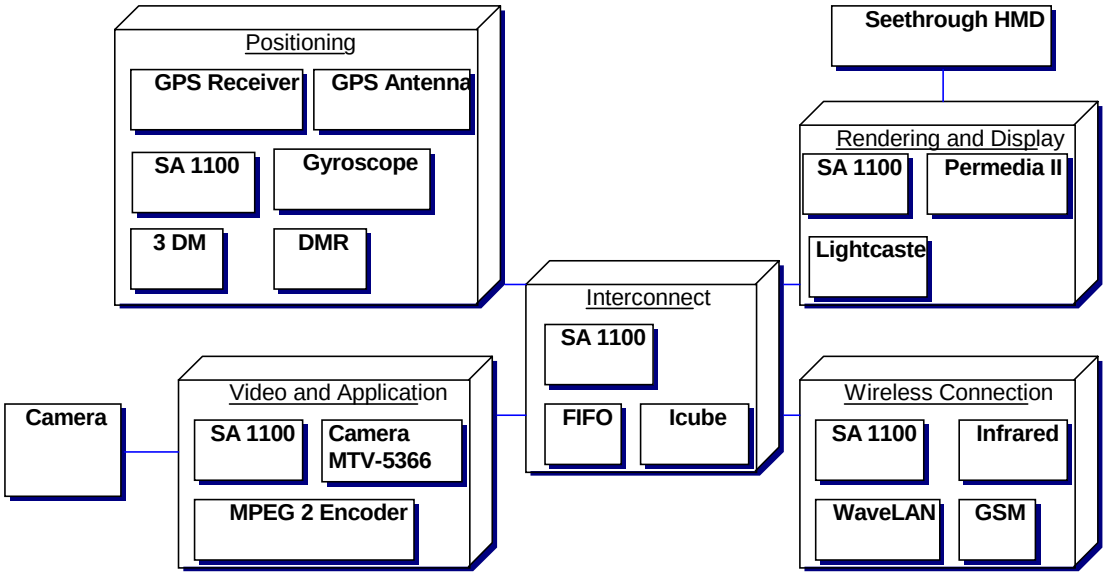


Figure 4.21.: Deployment diagram of the UbiCom client prototype

UbiCom implements an approach where part of the tracking is done on the server, and part on the client. For video-based tracking the images are transferred to the server, additionally the client uses an inertial tracker for calculating the pose and GPS for the position. Video-based rendering is only needed to correct the drift of the inertial tracker after some milliseconds.

5. Comparison of Approaches

In the last chapter, we showed how different architectures share similarities on a broad level. On a subsystem level, however, the developers used quite different approaches to solve the different problems of AR, e.g. tracking or rendering.

Some of these approaches recur within different systems—sometimes explicitly, such as when two systems use a common library, and sometimes implicitly, when different developers apply the same basic techniques.

In this chapter, we will build on the last chapter's subsystem decomposition to compare the approaches to building key AR components.

5.1. Methods

Our methods in this chapter are heavily based on the idea of *patterns* in software architecture. Patterns are structured descriptions of successfully applied problem-solving knowledge. We are not so bold as to call the approaches we identified "patterns", as we would need more data for that—but in the future, we hope to be able to set up a system of "AR patterns."

Each approach is described by name, goal, motivation, a description, usability, consequences, and project usage. This follows the scheme of describing architectural or design patterns, e.g. as used in [GHJV95, SSRB00].

5.2. Application

The application subsystem is the place where the application developer can add the application-specific logic. There are various solutions possible with different advantages and disadvantages.

5.2.1. Main Executable

Goal: Keep the flow of control.

Motivation: The main parts of an application is independent of Augmented Reality. Indeed AR is only one part among others and is only used to visualize some content.

Description: Write the application in a high-level programming language, explicitly describing what happens when.

Usability: Use this approach if it is necessary to keep the control flow, for example to guarantee real-time constraints for non-AR subsystems, e.g. reacting to external events. The disadvantage is that it is up to the application developer to implement the continuous update of registration and the rendering.

Consequences: The modifiability of the application is low.

Known use: MR Platform

5.2.2. Scripting

Goal: Quickly develop new applications.

Motivation: The real-time constraints of a user application are often not very strong, so that it is possible to quickly develop new applications in a scripting language supported by a powerful environment.

Description: For the development of an application, there is a scripting wrapper around all components that have performance constraints. These components are written in compiled languages such as C++ and offer scripting interfaces.

Usability: The development of scripted applications allows rapid prototyping but demands powerful components that implement important functionality. The disadvantage is that the scripting approach is not suited for very complex applications.

Consequences: A script interpreter is needed, as well as (possibly) a special scripting language for AR.

Known use: ImageTclAR, Karma, Coterie, MARS, EMMIE

5.2.3. Node in Scene Graph

Goal: Embed application in world model.

Motivation: In Augmented Reality, user interaction is connected with the physical environment. Consequently applications are often linked to places in the real world. With this approach, the application is seamlessly embedded in the environment.

Description: A scene graph models the world around a user as a tree of nodes. Each node can be any type object, usually graphical ones. But there are also non-graphical objects that include control code.

Usability: Together with a scene graph-based rendering approach.

Consequences: The scene graph-based approach for an application handles the control flow to the underlying scene graph platform, e.g. Open Inventor. On the other side this approach offers a relatively easy possibility for the implementation of shared applications for locally nearby users. One 3D interface can be shared among several users but displayed for each user from a different view.

Known use: Studierstube

5.2.4. Part of Event Loop

Goal: Let an AR library do the tracking and rendering and call the application within the tracking-rendering loop.

Motivation: The tracking and rendering must be done in a regular loop that updates the user's view based on her motion. Embed the application into this loop.

Description: To alleviate the development of AR applications some libraries provide the needed low-level functionality to update the user's view regularly. The application's task is to provide hooks that can be called within the update loop and that might react on changes in the view.

Usability: With a library for tracking and rendering.

Consequences: The control flow is managed by the update loop of the tracking-rendering system.

Known use: ARToolKit

5.2.5. Web Service

Goal: Treat AR as one type of media among others.

Motivation: For content-based applications the web-based approach has been proven to be a reasonable approach. AR scenes and world model information can be seen as an AR document. A scene such as an arrow that points to a particular button in front of the user is then described in document that is loaded from a web server.

Description: The control flow is situated on a web server and implemented within a web service. This web service is published under a particular web address and the answer of the service is rendered on a web client. If the answer contains Augmented Reality content then the AR component is activated to display the given AR content.

Usability: This approach can be used where the focus is on displaying various types of content and load them dynamically from a server.

Consequences: The client and the server must be connected. If a connection cannot be guaranteed then there must be a proxy available locally that emulates the server. Alternatively, a smaller instance of the server component may be deployed on the client machine. This approach should be combined with a scene-based rendering component, e.g. a VRML or custom AR browser.

Known use: ARVIKA

5.2.6. Multimedia Flow Description

Goal: Use high-level description language to describe AR scenes.

Motivation: For the development of multimedia content, there are several formats that simplify the creation of new content by providing high-level concepts such as timers. Examples for such languages are SMIL and Macromedia Flash. Additionally to multi-purpose languages for multimedia content, there are domain specific languages for particular fields—for example, description languages for Workflows or for technical manuals (IETMs).

Description: A high-level markup language provides domain specific components and concepts that help quickly create new content. For example, to support a training scenario for unskilled workers, the AR system should visualize a sequence of AR scenes and other documents. To describe such a scenario, the content creator has to combine workflow steps and add content to each step. An execution engine for workflows reads such a description and controls the presentation of the current working step.

Usability: This approach can be used for applications with a meta-model for the description of documents and their relationship and dependencies combined with a component that reads and executes such descriptions.

Consequences: The complexity of this approach is higher for simple applications. This approach should be combined with a scene-based rendering component, e.g. a VRML or custom AR browser.

Known use: STAR, ARVIKA, DWARF

5.3. User Input

Augmented Reality systems tend to concentrate more on output than on input; however, the interest in new user input techniques and architectures is growing. In this section, we concentrate on architectural approaches of combining user input, not on the input devices

themselves.

5.3.1. Handle in Application

Goal: Keep the system architecture simple; provide high-fidelity interfaces.
Motivation: The simplest way to handle user input for a specific application is to hard-code it into the application logic itself. Also, this allows custom-tailored input styles for each application.
Description: Include input handling code in the application code, with explicit references to the types of input devices.
Usability: Within a main executable application.
Consequences: Potential for high-fidelity interfaces; reduced modifiability.
Known use:

5.3.2. Use Browser Input Functions

Goal: Take advantage of existing functionality.
Motivation: When using a browser for rendering, you can take advantage of its input features.
Description: VRML browsers can send events out through the EAI interface when the user clicks on on-screen objects with the mouse or when the gaze direction coincides with certain objects. Other broswers provide similar functionality.
Usability: Together with a browser-based output subsystem.
Consequences: Separating input from output modalities and integration into multimodal systems can be difficult. Since wearable systems do not generally have a mouse, the mouse movement must be simulated with another input modality.
Known use: ARVIKA

5.3.3. Networked Input Devices

Goal: Combine the data of various input devices to form multi-modal user interfaces.
Motivation: Multi-modal interfaces require many simultaneous input devices. Modeling all possible combinations is exponentially complex.
Description: Provide an abstraction layer for input devices and a description of how the user input can be combined; interpret this description using a controller component. Use middleware to find new input devices dynamically.
Usability: Good when combined with multiple viewers for output.
Consequences: Allows integration of new input devices at run time or when building new systems.
Known use: DWARF

5.4. Tracking

Without tracking, Augmented Reality is impossible. In this section, we concentrate on architectural approaches of gathering tracking data, not on the tracking devices or algorithms themselves.

5.4.1. Tracking Server

Goal: Use a centralized tracking server to reduce hardware requirements on the client system.

Motivation: There are many different tracking technologies available: magnetic, video based, inertial, or combinations thereof. Calculate the user's pose from raw data may require significant computing power. One strategy to avoid this is to offload the computing to a server in the user's environment and only transfer the result to the client system.

Description: A tracking server in the user's environment coll

Usability: Use when integrating a commercial tracking system.

Consequences: Availability of a connection to the server.

Known use: STAR, UbiCom.

5.4.2. Networked Trackers

Goal: Combine the data of various trackers in the environment without knowing the physical location.

Motivation: The best results for tracking can be achieved by combining various tracking methods. Usually, the possibilities to connect tracking devices physically to the client device or a tracking server are restricted. As a solution, wrap every tracker with a software interface that masks the location of the tracker by providing a network-accessible interface. To gather the data of all trackers, each tracker is accessed remotely. If supported by middleware, the lookup can be done by name, not by network address.

Description: For each tracking device, provide a wrapper that uses middleware concepts such as CORBA. The wrapper provides an interface to the tracker and registers itself in the network. Components that need a tracker (consumer) look for them through middleware services and connect to them. The components search for the trackers by name, not by address. Once connected, the tracker and the consumer communicate transparently.

Usability: This approach can be used when the client system can connect to tracking devices over the network.

Consequences: The advantage is that virtually any number of tracking devices of different types can be combined. The disadvantage is the overhead of network communication.

Known use: ARVIKA, Studierstube, DWARF.

5.4.3. Operating System Resources

Goal: Directly connect tracking devices to client system.

Motivation: If no network connection is available then only devices that are physically connected to the client device can be used. Particularly inertial trackers and video cameras are small and can be used.

Description: The tracking devices are accessed through drivers for the operating system.

Usability: The client device must powerful enough to execute the tracking algorithms.

Consequences: The advantage is that no network connection is needed but only devices that can be connected physically to the client device can be used. And the computing must be done completely on the client device.

Known use: MR Platform, ARToolKit

5.5. World Model

The World Model is used to describe the world around the user, particularly the virtual objects and their position. Besides that, the world model must also store information about the marker positions or any other features required for tracking.

5.5.1. OpenGL Code

Goal: Describe virtual objects in OpenGL source code and load objects at runtime from library.

Motivation: To be able to render objects in an AR system, it is enough to write some OpenGL code, compile it and feed the code to the renderer. For testing purposes, this will suffice.

Description: The developer creates OpenGL code and calls the OpenGL rendering engine to display it. For correct registration with the user's pose, the position and angle of the virtual camera that looks at the scene can be changed. This is usually done in rendering-tracking-update loop.

Usability: When only small scenes need to be presented, simple OpenGL code can easily be written and tested, or existing code for scenes may be reused.

Consequences: The advantages are that it is easy to test an AR system. However, this approach is not very scalable as OpenGL is a fairly low-level library for 3D.

Known use: ARToolKit

5.5.2. Scene Graph Format

Goal: Use a standard format with authoring tools to generate content and rendering engines to display them.

Motivation: Scene graphs are the standard component to display virtual environments. Each scene graph can read scene graph descriptions in several formats, where the most well-known is the VRML format.

Description: With an authoring tool a content developer creates the model of a virtual scene. In industrial context scenes created with CAD tools can be simplified and reused. The scene description is saved in the file system and given the AR system for processing. Scene graphs are usually stored on the file system.

Usability: Can be used where a scene graph is used for rendering, the scene does not change too often, and the scenes are discrete without interconnection with each other.

Consequences: Simplifies the creation of new content as authoring tools can be reused and fits perfectly to the scene graph component.

Known use: ARVIKA, DWARF, MR Platform.

5.5.3. Object Stream

Goal: Serialize world model in main memory to disk.

Motivation: Serialization is a well-known technique to save runtime objects. This can be reused for objects in a scene graph at runtime.

Description: The runtime environment allows to serializing objects to disk. The next time the application is started the objects are recreated by deserializing them. Recursively, a whole scene graph can be loaded from disk.

Usability: Usable where the world model is created by the system user and the system only has to read own formats.

Consequences: An object stream can be read only by systems that knows the classes of the objects. Thus, an object stream is a proprietary format.

Known use: Tinmith

5.5.4. Configuration File for Marker Positions

Goal: Read configuration data for the trackers from a flat file.

Motivation: The easiest way to tell a tracker what to look for and how to interpret it is to write it to a file and read it every time the system starts.

Description: At system startup or any time the system comes into a new environment the trackers have to know what to look for and how to interpret it. So the tracking component is told the file that describes the markers or any natural features it has to look for. It loads the file at startup time or upon request at run time.

Usability: Used if only a small number of markers are needed. If many markers are needed the same problems as with file-based scene graphs will happen.

Consequences: Does not scale well for data more complex than marker positions or for dynamically changing data.

Known use: ARVIKA, MR Platform, ARToolkit

5.5.5. Database

Goal: Use abstract description of world model from a database.

Motivation: For larger descriptions of the user's environment a collection of file-based scene descriptions may not be enough. For example, finding a particular scene by filename requires a name schema that could become quite complex depending on the number of scenes and the complexity of them. Especially in mobile environments, this can happen quickly.

Description: Instead of loading a particular scene from a file, the system has access to a database system. This system contains information about the environment, e.g. in a geographical schema. Part of the information are graphical information and marker information. The system queries for the graphical information that belongs to a discrete database object and passes it on to the rendering component. The same is true for marker information (to the tracking component) and real world objects (e.g. for occlusion).

Usability: World models in a database can be combined with graphical models that are also saved in the database. This decouples the application dependent model, e.g. the model for a machine to be repaired, from the graphical information needed to render objects that are part of the model.

Consequences: A database server is a heavy-weight component compared to file system but offers more possibilities to model larger environments. Especially useable for mobile applications a database model based on geographical information system (GIS) concepts combined with graphical models for concrete objects.

Known use: ARVIKA, ArcheoGuide, MARS.

5.6. Context

The context information must be gathered, processed and distributed to interested components.

5.6.1. Blackboard

Goal: Gather and process context information.

Motivation: Blackboards are a well-known pattern to gather information from various sources, apply rules and create new information. This is particularly suited where a lot of raw data with low-level information must be refined in several steps to higher abstract information. Consumers and producers of information are decoupled.

Description: Information producers write information to the Blackboard, a central component. Information consumers read data from the Blackboard, process them and may write new, higher abstract information to the Blackboard.

Usability: Good if information from many sources must be analyzed and filtered.

Consequences: Blackboards are a central component which might become a bottleneck. As an advantage, the participating components do not need to know each other.

Known use: MIThril, ARVIKA.

5.6.2. Repository

Goal: Provide a central means for information exchange.

Motivation: Many components need context information, other produce context information. A Repository component is a central component that is well-known by all components. It acts as a information storage.

Description: Components that produce context information write to the repository. Components that are interested into context information read from the repository. The repository uses an addressing schema to manage the information. Each kind of data is written and read by providing its address.

Usability: Good if data from many sources must be stored and read.

Consequences: Similar to Blackboards, Repositories can be a bottleneck. But again the participating components do not need to know each other.

5.6.3. Publisher/Subscriber

Goal: Provide a means to distribute context information.

Motivation: When context data does not need to be saved for a certain period of time, it can be distributed at once without need for a temporary storage.

Description: Context providers connect as publishers to a central messaging service, context consumers as subscribers. The context providers write the new context information to a particular channel which distributes it to the connected subscribers.

Usability: May be used where there is no need to store context information. Could be used in combination with a context repository. Some data may be saved, some delivered at once.

Consequences: Again might be a bottleneck and publishers and subscribers do not need to know each other.

Known use: DWARF, ARVIKA

5.6.4. Ad hoc

Goal: Connect components that need context information directly with source.

Motivation: For simple cases a centralized service for data exchange may not be needed. For example, if only location context is needed a direct link to the tracking subsystem may be enough.

Description: An interested component directly queries the context producer component or it registers itself as subscriber. The subscriber list is managed by each component privately.

Usability: Suited for reduced context sensitivity.

Consequences: If the amount of context sources and context consumers increases ad hoc hardwired direct connections will lead to unmanageable code. The coupling between consumer and producers is very tight, which makes them very interdependent.

5.7. User Output and Rendering

The User Output subsystem in AR deals mainly with the presentation of three-dimensional information; thus, most of the approaches here are geared specifically to rendering.

5.7.1. VRML Browser

Goal: Use a rendering component that can display simple virtual scenes.

Motivation: The usage of a VRML browser is a simple way to display virtual scenes. The standardized VRML format, a markup language for the description of virtual worlds, allows to use a lot of tools for authoring virtual worlds and to reuse components that can render descriptions of virtual worlds.

Description: Use a third-party VRML browser, often designed as a web browser plugin, to display 3D information. Use the External Authoring Interface (EAI) that is part of the VRML standard to modify the scene and set the viewpoint based on tracking data.

Usability: A VRML browser component can be used if the complexity of the scenes is relatively low and the browser is only used as a rendering engine.

Consequences: The advantages of using a VRML browser are the standardized format and the reuse of tools for authoring and the reuse of existing components. This allows rapid prototyping of Augmented Reality system based on VRML scenes. The disadvantages are that the EAI is restricted to relatively simple operations and that tying the VRML browser to the rest of the system may be tedious. Also, the rendering performance of VRML browsers is not as high as that of native OpenGL.

Known use: STAR, DWARF

5.7.2. OpenGL

Goal: Use a standardized library to render 3D objects and keep maximum flexibility and control.

Motivation: OpenGL is the standard low-level library for 3D graphics. While higher-level approaches, particularly scene graphs, provide a more powerful interface for 3D worlds, OpenGL provides the most flexibility to the application programmer. Scene graphs use their own control path user applications have to comply with. By using OpenGL, the developer can implement his own control flow.

Description: OpenGL provides low-level 3D constructs. The application developer creates new objects and tells the render to display them [Ope02]. With the information from the trackers the scene can be rendered with the correct viewing direction and distance.

Usability: Usable on nearly all systems.
Consequences: Modifiability of system is low; changes to the scene lead to a recompilation.
Known use: Older ARToolKit versions.

5.7.3. Scene Graph

Goal: Use a rendering component that allows more complex and dynamic scenes.
Motivation: For the representation of 3D environments, scene graphs have shown to be a reasonable choice. The level of abstraction is higher than for OpenGL, but they are much more powerful and flexible than VRML browsers with a limited application programming interface. Most scene graph components can read VRML based descriptions of scenes.
Description: Examples are (Open) Inventor, OpenSG, Open Scene Graph.
Usability: Use a scene graph if you don't need the low-level graphics access that OpenGL provides but want to render more complex scenes and need more dynamic access that a VRML browser offers.
Consequences: Can restrict the possibilities for modeling the application.
Known use: ARVIKA, Studierstube

5.7.4. Proprietary Scene Graph

Goal: Use a rendering component that can render scenes and provides a customized programming interface.
Motivation: There may be reasons to develop an own scene graph for rendering, e.g. the provided interfaces are not satisfying or the control over the data flow, that is often controlled by the scene graph, should be kept.
Description: The Tinmith system uses an own scene graph for graphics rendering on top of OpenGL combined with an own concept for object access through an own addressing schema. Each node of the scene graph has the same abilities to serialize and address them as the other objects in the system.
Usability: An own scene graph may be useful if the developer wants to control the implementation and the behaviour of the scene graph. For example, in the Tinmith system each object of the system can be made persistent and it can be reached over an address schema. The nodes of the scene graph are objects of the same type and there have the same attributes.
Consequences: The scene graph has to be developed from the ground up. For example, to make scenes persistent, either a parser for a standard format like VRML has to be developed or a proprietary format which cannot reuse scenes developed by standard tools such as VRML editors.
Known use: Tinmith

5.7.5. Video Transfer

Goal: Offload the video rendering to a server, transfer it to the client and present it there.
Motivation: To reduce hardware requirements of the client system, use a rendering server in the environment and transfer the completely rendered images to the client. Particularly suited for video see-through AR.
Description: The client gathers videos through one or two head-mounted cameras, encodes them (e.g. MPEG 2), compresses them, and transfers them to the server. The server

uncompresses the video images, processes them (calculates the camera position and orientation), augments, encodes and compresses the images. The images are sent to the client, decompressed and shown on the HMD.

Usability: This can be used with a good network connection and a strong rendering server.

Consequences: The advantage is that the hardware requirements for the clients are very low. Even PDA-class devices can be used. The main disadvantage is increased latency due to the network transfer.

Known use: STAR, MR Platform, AR-PDA.

5.7.6. Multiple Viewer Classes

Goal: Use different media types for different types of information.

Motivation: Although the central requirement of AR is displaying three-dimensional information, other types of output devices such as speech synthesis or wrist-worn displays are also useful.

Description: Provide an abstraction layer for different types of viewers (AR, speech, text etc.) that can handle certain document types. Then provide the viewers with the appropriate documents.

Usability: This approach can be used whenever multiple output media are desired, unless the rendering complexity is so high that the 3D viewer cannot be parameterized.

Consequences: Advantages are a modular system design and the possibility of integrating additional output devices at run time or in new systems. Disadvantages are the extra complexity of a viewer abstraction layer.

Known use: ARVIKA, DWARF

6. Conclusion

In this chapter, we summarize our main results, point out difficulties and show opportunities for future work.

6.1. Results

In this study, we have assembled descriptions of many different AR projects worldwide, their research and development goals, their application requirements and their system architectures. We have shown which quality attributes are significant requirements for all AR systems and proposed links between certain application domains and specialized requirements. We have developed an abstract reference AR architecture as a superset of existing architectures and have mapped several existing architectures onto this in order to make comparison of AR architectures possible. We have identified common approaches used in different AR systems to achieve the same goals, and have shown how the use of these approaches affects the systems' quality attributes.

We hope that this data will help system architects make informed decisions on their own architectures, and also promote the discussion of architectural approaches within the AR field.

6.2. Lessons Learned

Gathering descriptions of software architectures from a young field is hard; comparing them is even harder. We have sorely recognized the need for a common language in the description of software architectures—this will certainly still take years to develop.

In order to make more valid statements about specific systems, we now realize that we would have to iterate the process of interviewing architects and analyzing the results. Given the results of our first questionnaire, the second would probably structure the questions in a different fashion.

Building AR architectures is not an easy task; there is no one-size-fits-all approach. Instead, the architects must carefully choose the components best suited to their particular application. Systems that are close to commercial production are furthest here, especially in the area of industrial service and maintenance.

6.3. Future Work

In this study, we have laid the foundation for further research in AR software architectures. We see four main directions in which this work could be built upon by ourselves or by others.

First would be to pursue a more detailed mapping of application domains and project types onto required quality attributes, thus identifying clusters of systems that would profit from a similar architecture.

Second would be to further develop the generic AR architecture, validating it against more systems and extending it in the process.

Third would be to develop the list of architectural approaches in AR into an AR pattern language, describing the links between different approaches and the detailed consequences of choosing one approach. The combination of these three steps would make a guidebook for the choice of software architectures in AR.

Fourth and last is to use the knowledge we have developed so far to lake a closer look at one particular architecture, trying to identify its specific strengths and weaknesses and capitalize on working solutions to problems in AR that are used in other systems.

A. Web References

- ARVIKA – Augmented Reality for development, production and service
  http://www.arvika.de/
- The Augmented Reality Homepage:
  http://www.augmented-reality.org/
- Jim Vallino's Augmented Reality page:
  http://www.cs.rit.edu/~jrv/research/ar/
- Shin'ichi Konomi Ubiquitous Computing links:
  http://homepage1.nifty.com/konomi/shinichi/ubicomp.html
- Sony CLS lab Augmented Reality link collection:
  http://www.csl.sony.co.jp/projects/ar/ref.html
- Augmented Reality list by University of Washington HIT lab:
  http://www.hitl.washington.edu/projects/knowledge_base/ar.html

B. Questionnaire

This appendix contains the questions from the questionnaire we distributed, in condensed form. We will gladly supply the full questionnaire upon request.

B.1. Project context

First, please give us some information on the history and context of your development project. Please use additional questionnaires for separate projects.
What is the name of your project?
When and how did the development project start?
How many people are working on the project?
How many person-years have gone into the project up to now?
Is the project focus on research or building commercial products?
When do you expect the first commercial products to become available?
What would the target market(s) be?
Which area of business are any commercial partners from that are involved in development?
Which area of business are any commercial partners from that are potential customers?
Is the project focus on building entire AR systems or individual components (e.g. for human-computer interaction, tracking, etc.)?
If it is on building components, which components?

B.2. Supported and planned scenarios

Next, we would like to find out which scenarios the system developed is planned for. If you need more space, please use an extra sheet of paper!
Please briefly describe the main scenario(s) the system is planned for (e.g. the system overlays instructions on how to remove the paper cassette of the laser printer).
Which application domains does (or will) the system support?

- ☐ Industrial construction, e.g. of power plants, aircraft or automobiles
- ☐ Industrial maintenance
- ☐ Location-based services, e.g. navigation and traveller information
- ☐ Pervasive computing and intelligent rooms or buildings
- ☐ Telecommunications (e.g. teleconferencing)
- ☐ Medical AR (e.g. surgical planning and assistance)
- ☐ Military
- ☐ Automotive (e.g. driver support)
- ☐ Entertainment industry (e.g. games)
- ☐ Infotainment industry (e.g. museum guide, AR supported learning)
- ☐ Building industry (e.g. architecture or exterior construction)
- ☐ Support for the handicapped
- ☐ Other
- ☐ Other

In which part of the business life cycle would your system be used?

- ☐ Design (e.g collaborative design and data exploration)
- ☐ Prototyping

- ☐ Construction
- ☐ Testing
- ☐ Maintenance
- ☐ Refurbishment
- ☐ Product marketing and visualization (e.g. product design and visualization of new car models or buildings)

How is the system integrated into the business process?

B.3. Quality Attributes

In this section, we would like to quantify the requirements for the system. These arise from the desired scenario and the business context, both described previously. Not all quality attributes are of high priority for every system.
Please check the quality attributes or nonfunctional requirements that are of high priority for your system. If possible, please include a short description.

B.3.1. General Quality Attributes Visible at System Runtime

Performance

- ☐ Latency of 3D tracking and rendering
- ☐ Complexity of scene that can be rendered
- ☐ Other

Reliability

- ☐ Repeatability, i.e. same tracking data at the same position
- ☐ Accuracy of tracking
- ☐ Range of tracking
- ☐ Quality of rendering
- ☐ Fault tolerance, e.g. of tracker failing
- ☐ Robustness
- ☐ Other

Availability

- ☐ Amount of time the system is up and running
- ☐ Ability to tolerate network failures and disconnectivity
- ☐ Other

Security

- ☐ Ability of the system to support multiple users with different access rights
- ☐ Ability of the system to withstand malicious attacks
- ☐ Information privacy, e.g. of the user's position and identity
- ☐ Other

Usability

- ☐ Hardware that the user is required to wear on the body
- ☐ Restriction of the user's freedom of movement
- ☐ Restriction of the user's field of view
- ☐ Necessity to calibrate the system at runtime
- ☐ Richness of information presented to the user
- ☐ Ease with which the user can reconfigure the system at runtime
- ☐ Safety
- ☐ Use in harsh environments, e.g. on the factory floor
- ☐ Support for different input modalities
- ☐ Support for different simultaneous input modalities
- ☐ Other

B.3.2. AR–Specific Functionality

Tracking

- ☐ Ability to track user's head
- ☐ Ability to track multiple users
- ☐ Ability to track objects in the environment
- ☐ Ability to track user's hands
- ☐ Ability to use different tracking devices simultaneously
- ☐ Tracking on the user's body ("inside–out tracking")
- ☐ Tracking from the environment ("outside–in tracking")
- ☐ Other

Context

- ☐ Adapt to user's preferences
- ☐ Adapt to user's abilities
- ☐ Adapt to environmental context
- ☐ Other

Rendering

- ☐ Stereo rendering
- ☐ Support for multiple simultaneous viewers
- ☐ Projection onto non-worn surfaces
- ☐ Other

User Interaction

- ☐ Speech recognition
- ☐ Speech synthesis
- ☐ Gesture recognition
- ☐ Tangible interaction
- ☐ Interaction between users
- ☐ Other

Mobility

- ☐ Wireless operation
- ☐ Indoor operation
- ☐ Outdoor operation
- ☐ Network–disconnected operation
- ☐ Other

B.3.3. General Quality Attributes not Visible at System Runtime

Modifiability

- ☐ Ease with which the development team can adapt the system to new requirements
- ☐ Ability to reuse components of the system in new systems
- ☐ Ability to extend the system by adding new components
- ☐ Other

Portability

- ☐ Number of hardware and software platforms supported by the system
- ☐ Ease with which the system can be ported to new platforms
- ☐ Other

Integrability

- ☐ Ease with which individual components of the system can be integrated during the development process
- ☐ Ability to integrate legacy or third-party components

- ☐ Ability to integrate with components of other AR architectures
- ☐ Other

Testability

- ☐ Ease of testing individual components of the system
- ☐ Ability to find faults in the system if they are present
- ☐ Proven correctness of the system behaviour
- ☐ Other

B.4. System Architecture

In this section, we would like to discover as much as possible about the system architecture. Feel free to include separate diagrams if you like; if you need more space, please use extra sheets of paper! Don't hesitate if you are unfamiliar with some terms. Please give the information that you think is most appropriate. Any kind of information is better than none.
What are the main software components of the system architecture? Please describe the main subsystems to give an overall picture.
Please describe how these components interact. If possible, use the main scenario described previously as a guideline.
What kind of computers, with which operating system, are the software components deployed on?
Which I/O devices do the components use, and which computers are they attached to?
How is tracking achieved with these components?
How are user output and visualization achieved with these components?
How is user input (if any) achieved with these components?
How is application logic modelled with these components?
How is enterprise application integration (EAI) achieved? How does the data get into and out of the backend systems?

B.5. Architectural Approaches

Architectural approaches, also called architectural styles, describe the types of components that may be used to build a system, and what types of interactions are possible. This allows us to identify general (overlapping) families of architectures. Generally, a system uses several approaches at the same time.

B.5.1. General Architectural Approaches

Which general architectural approaches does your architecture use? Please explain!
Data flow approaches, dominated by motion of data through the system, with no control by the recipient:

- ☐ Batch sequential
- ☐ Data flow network
- ☐ Pipes and filters
- ☐ Closed-loop control
- ☐ Other

Call-and-return approaches, dominated by order of computation, usually with single thread of control:

- ☐ Main program/subroutines
- ☐ Abstract data types
- ☐ Objects
- ☐ Call-based client/server
- ☐ Layered
- ☐ Other

Independent component approaches, dominated by communication patterns among independent, usually concurrent, processes:

- ☐ Event systems

- ☐ Communicating client/server processes
- ☐ Communicating peer-to-peer processes
- ☐ Communicating threads
- ☐ Broadcast communication
- ☐ Other

Web-based approaches, dominated by choice of communication protocols, server and browser operating environments:

- ☐ Fat client
- ☐ Thin client
- ☐ Browser plug-in
- ☐ Other

Data-centered approaches, dominated by a complex central data store, manipulated by independent computations:

- ☐ Repository
- ☐ Blackboard
- ☐ Other

B.5.2. AR-Specific Architectural Approaches

Which AR-specific architectural approaches does your architecture use? Please explain!
Scene graph approaches, driven by the modelling of three-dimensional data:

- ☐ Scene graph based user interfaces
- ☐ Distributed scene graphs
- ☐ Other

Tracking approaches, driven by need for correct pose estimation:

- ☐ Distributed tracking
- ☐ Hybrid tracking
- ☐ Other

User interface models, driven by structure of user interaction:

- ☐ Scene-based user interfaces
- ☐ Generated user interfaces
- ☐ Multimodal interfaces
- ☐ Taskflow models of user interfaces
- ☐ Other

B.6. Effect of Architectural Approaches on Quality Attributes

In this section, we would like to find out how the chosen architectural approaches affect the desired quality attributes. In a sense, this is a test of the chosen architecture.
How did the chosen architectural approaches help to achieve the high-priority quality attributes?
How did the chosen architectural approaches adversely affect high-priority quality attributes?
What trade-offs had to be made between quality attributes?

B.7. Future Scenarios

In this section, we would like to explore future possibilities for AR systems and architectures.
How would the chosen architecture support these changes to system requirements or available technology?
Supporting other application domains
Supporting more users
Real-time tracking and rendering
More complex data to be visualized

Porting to different hardware
Porting to different operating systems
Better network performance
Worse network performance
More processing power
Less processing power
Greater user mobility
What do you think will be the greatest challenges for future AR systems?

Bibliography

[aqu00] Aquagauntlet. `http://www.mr-system.co.jp/project/aquagauntlet/index.html`, 2000.

[arv01] ARVIKA - Augmented Reality for development, production and service. ARVIKA flyer, `http://www.arvika.de/www/e/topic5/flyer.htm`, 2001.

[aur02] Project Aura - Website. `http://www-2.cs.cmu.edu/~aura/`, 2002.

[BCK98] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesely, Reading, MA, 1998.

[CKK02] Paul Clements, Rick Kazman, and Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison Wesley, Boston, MA, 2002.

[con02] STAR consortium. Star website. `http://www.realviz.com/STAR/`, 2002.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[HNS00] Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Object Technology Series. Addison-Wesley Publishing Company, 2000.

[Ioa02] Nikos Ioannidis. Archeoguide. `http://archeoguide.intranet.gr/project.htm`, 2002.

[Lag00] R. L. Lagendijk. Ubiquitous communications (ubicom) - updated technical annex 2000. Technical report, Ubiquitouc Communications Program TU-Delft, Jan. 2000.

[mrl02] MR Platform - Website. `http://www.mr-system.co.jp/mr-platform/`, 2002.

[Ope02] OpenGL.org. Opengl - high performance 2d/3d graphics. `http://www.opengl.org/`, 2002.

[Osh99] T. Oshima. Rv-border guards: A multiplayer entertainment in mixed reality space. In *Poster session of IEEE Internation Workshop on Augmented Reality*, San Francisco, USA, 1999.

[scr02] Siemens - SCR - Website. `http://www.scr.siemens.com/2a.html`, 2002.

[SSRB00] Douglas Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture, Vol. 2: Patterns for Concurrent and Networked Objects*. Wiley, New York, NY, 2000.

[tin02] Tinmith - Website. `http://www.tinmith.net`, 2002.

[UTS+02] Shinji Uchiyama, Kazuki Takemoto, Kiyohide Satoh, Hiroyuki Yamamoto, and Hideyuki Tamura. Mr platform: A basic body on which mixed reality applications are built. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, Darmstadt, Germany, 2002.