Reconfigurable Architectures for Component-based Augmented Reality Systems

Thomas Reicher
Department of Computer Science
Technische Universität München
Arcisstr. 21, 80290 Munich, Germany
reicher@in.tum.de

Research Area

Component-based software engineering for augmented reality systems.

Technical Problem

For my work I am driven by the requirements of Augmented Reality (AR) systems on wearable computers and ubiquitous computing. In this thesis I am combining concepts from both fields with concepts from component-based software engineering to develop a homogenous component model where the borders of a wearable computer system is extended into its environment.

AR applications provide users with a new way to interact with their environment. By tracking the user’s position and viewing direction the system can overlay artificial artifacts on real world objects. The most prominent example is superimposing computer graphics on real world objects presented in a head mounted display (HMD) [4, 5]. Application areas of AR can be found in service and production of complex systems such as powerplants service [6], the installation of cable trees in planes [3], or in car assembly [8].

For the correct overlay of virtual objects tracking of the user’s position and orientation must be very precisely and in realtime. This is an field of active research (see [1]). The devices for tracking are usually connected to the user equipped with a wearable computer, for example a small video camera. One newer approach is to integrate tracking information from devices in the user’s environment [7].

The needed ingredients or parts for an AR system can be identified but up to now augmented reality systems are handcrafted systems that are quite monolithic and standalone. My goal is to develop a distributed, component-based peer-to-peer architecture where the components on the wearable computer and the components in the environment can be combined homogenously and ad hoc to build an application system (see Figure 1). An application system can reconfigure
at runtime by exchanging components. Each component is an active component that has type, runtime state and knowledge about itself.

![Distributed design of AR system](image)

**Figure 1. Distributed design of AR system**

Following this idea I found the following technical problems:

- The target devices are relatively small with restricted resources.
- The development of applications with low-level techniques such as socket programming is too complicated and expensive.
- Existing middleware solutions are too resource demanding and inflexible.
- Connection attributes between components are usually not an issue of the architecture on desktop applications and LAN based distributed applications. The standard modelling techniques do not support them properly.
- In a distributed world with ad hoc connections between components the attributes of a connection at design and runtime are important.

**Justification**

In existing middleware solutions such as CORBA, RMI or Microsoft COM the developer has no influence on how the components communicate because the communication is hard-coded into the middleware implementation. There should be a way where the components can specify how they want to communicate with other components.

Another disadvantage in CORBA or COM is that components get information about each other by interface specifications written down in an IDL. These interface specifications do not describe the quality of service that the interface provides, for example, information about a tracker component’s frame rate so that a renderer can decide whether it wants to use the tracker’s service. Active components can give more runtime information about themselves than pure software components.

Modelling techniques such as the UML do not have concepts to model heterogenous connection types between components in a homogenous way. I think it is very important to treat connections on the same level as the components and model the attributes of them.
Hypothesis

My hypothesis is that the introduction of a meta-layer in each component and Connector objects can solve some of the mentioned problems. Specifically I use the following concepts:

- Separation of the system initialization phase from the actual application specific communication.
- Each component is active and can give information about its current state and connections.
- Interfaces to manage components at runtime.
- Introduction of Connectors as first class objects at design and runtime.
- Modelling knowledge can be used by applications and components at runtime.

I introduce a set of interfaces for the components that are independent from the functionality of the component. So I can separate all interfaces in functional interfaces of the components and management interfaces that control the components and the functional interfaces. Namely I provide interfaces for the establishment of a connection to the desired component, to manage them and to query about the provided functional and needed functional interfaces (Needs and Abilities). Needs and Abilities will be described in an XML dialect. Components and connectors are typed. So an instance of a component can declare its own type, on which other types of components it relays on, and which type the Connector should be.

Connections between components are first class objects both in the application models as well as during runtime. Connector objects represent a connection between components on either side.

With management interfaces and Connector objects I can distinguish two phases, a setup and initialization phase and an application communication phase. In the first phase the components communicate on the management level to offer own services and to find the needed partner components. For this I use a Service Manager. A Service Manager is a superset of a location manager as known from Jini’s Lookup Service, the CORBA Trading Service or the Service Location Protocol. Additionally to providing a database with registered services and making a pattern matching based on attributes the Service Manager also starts available but not yet started services or stops them afterwards and establishes Connectors that provide the infrastructure for communicating components such as a shared-memory space, network sockets or a CORBA event channel. In the second phase, after the Connectors were established the components exchange data. How they do it was negotiated in the first phase.

With the clear separation of component and connector I am able to provide a homogenous view on the system while preserving the needed flexibility in data exchange between components.

Contributions

With my research I want to do a first step towards using distributed component-based software concepts for limited devices where existing middleware concepts are to inflexible. Especially AR systems on wearable computers should be supported. The developed DW ARF prototype is a base for component-based AR systems.

The same concepts could be applied to applications in a ubiquitous computing environment. There dozens or hundreds of services from devices in the user’s environment are coming and
leaving dynamically when the user walks. Applications can be used when the needed services
could be found and a system can be constructed dynamically.

The developed prototypes will give some insight about software engineering with distributed
active components. The border of the system is very fuzzy and very dynamic because con-
nections between the components come and go while the user is moving in his environment.
If for an application a certain constellation of components can connect to each other then the
application can be used. In a change of the connectable components the system can reconfigure
dynamically.

Methods

As a first step I studied existing AR systems to find out about the functional and non-
functional requirements and current solutions. In the existing solutions the needed subsystems
and their functional interfaces could be identified.

Based on the identified components the concept for a flexible configuration of an AR system
and an architecture instance was designed. As a technical implementation platform CORBA
for the communication with the Service Manager was chosen. This does not contradict my
statement about existing middleware because we separated the system initialization phase from
the application communication phase.

With this design a prototype as proof of concept was implemented with a couple of master
theses, called DW ARF. This prototype successfully demonstrated that AR systems could be de-
ployed on a distributed system. Successful means that the development of the demonstration
application was quickly as component-based software engineering promises and the commu-
nication overhead for the middleware could be neglected. The concepts were presented at the
International Symposium on Augmented Reality (ISAR 2001)[2]. Meanwhile this architecture
is used for other AR prototypes.

With the knowledge gathered with the first prototype I will enhance the actual concept to-
wards a formal meta-model for components and their interfaces. With such a meta-model the
design of new systems with tools will be easier and it allows to develop tools to manage such
a distributed system, for example show the involved components. Such a model of a system
architecture can be used at runtime to let the application reason about its state. For example
whether the needed components are available, how they must be configured or control the state
of the components.

References

MacInyre. Recent advances in augmented reality. IEEE Computer Graphics and Applica-

Riß, Christian Sandor, and Martin Wagner. Design of a Component-Based Augmented

an ar application work in the airplane factory. In Reinhold Behringer, Gudrun Klinker, and


