DWARF
Distributed Wearable Augmented Reality Framework

Technische Universität München, Chair for Applied Software Engineering
Prof. Bernd Bruegge Ph.D., Prof. Gudrun Klinker, Ph.D.

The Chair for Applied Software Engineering at the Technische Universität München is doing research on the field of Augmented Reality systems. Augmented reality (AR) is a technology by which a user's view of the real world is augmented with additional information from a computer model. Users can work with and examine real world objects while receiving additional information about those objects or the task at hand. Rather than pulling the user into the computer's virtual world, AR brings information into the user's real world, thereby building upon people's visual and spatial skills. AR constitutes a very promising new user interface concept for many applications, e.g., in medicine, exterior construction, interior design, the assembly, maintenance, and repair of complex technical objects, and games (as seen in figure 1).

From a software architecture point of view AR systems can be divided into a set of components that each contribute a dedicated functionality to the whole system.

Figure 1. Setup of a Game Application

The task was to find a solution where rapid prototyping with several researchers on different computers should be possible. This way we wanted to accommodate the heterogeneous infrastructure of the chair consisting of several Intel-based PCs and laptops with Windows and Linux, Apple workstations and laptops, and Compaq iPAQ PDAs. Additionally it should be possible to integrate third party components such as external trackers into the system. The solution to these requirements a framework for component-based peer-to-peer systems, called DWARF. We model Augmented Reality applications as distributed systems which form spontaneously from mobile and stationary components. Such a system can reconfigure itself at runtime by exchanging components and changing component configurations. As communication infrastructure we chose CORBA.

Architecture. The DWARF framework is based on the concept of collaborating distributed *services*. The services are interdependent and expose their requirements, called *Needs*, and offers, called *Abilities*, with the help of *service managers*. On each network node of a DWARF system, there is one service manager; there is no central component. The service manager controls its local services and maintains descriptions of them. Each service manager cooperates with the others in the network to set up connections between services. Each service has an XML description of its *Abilities*, *Needs* and *connectors* (communication protocols). See figure 2. This concept extends the approach of the CORBA Component Model (CCM) of required and offered interfaces respectively event sinks and sources towards more flexibility in the inter-service communication and is part of the research efforts.

Each Ability has a set of attributes describing quality-of-service parameters of that service. Likewise, each Need specifies a predicate about the quality of service it expects. This predicate is used by the service manager to select abilities that can provide a sufficient quality of service to satisfy a given need. This predicate can also be used at runtime to ensure that the desired quality of service is still provided.

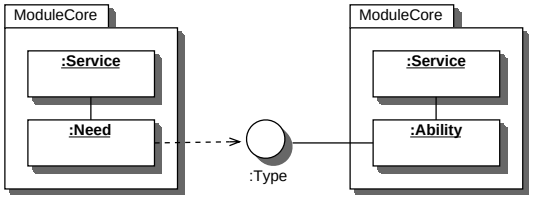

Figure 2. Services get connected by Needs and Abilities

Implementation. The stationary computers are connected together using standard 100 megabit ethernet; the mobile computers (laptops and palmtop) are connected using IEEE 802.11b wireless ethernet.

The DWARF services are realized as separate processes and threads within single processes. Distributed middleware, consisting of CORBA and several DWARF service managers, connect the services together.

Upon startup, each service registers itself, via CORBA, with its service manager running on the local machine, which listens for connections on a well-known TCP port. The middleware is lightweight enough to run on the Linux-based palmtop as well as the full-sized machines.

The service managers running on the different machines communicate with one another using SLP and CORBA and set up connections between the services. The services then use CORBA method calls or CORBA notification service events to communicate.

The format of communication between the services is specified in CORBA IDL. Some services have CORBA interfaces with specified methods that are called by other services; most, however, communicate using CORBA structured events.

The implementation of the DWARF service manager uses the open-source OmniORB CORBA implementation [1], which is also used for the services written in C++. Additionally, it uses OmniNotify, a CORBA Notification Service implementation based on OmniORB. Both OmniORB and OmniNotify were cross-compiled to Linux StrongARM so that the middleware could run on the palmtop. Services written in Java use OpenORB, an open-source Java CORBA implementation [2], and JavaORB, its predecessor, which was necessary for the Java 1.1 virtual machine needed to control the Cortona VRML browser under Windows.

To find services managed by other service managers, the service managers use OpenSLP [3], an implementation of the Service Location Protocol SLP. This allows services to be found based on type and named attributes. OpenSLP can evaluate predicates based on boolean expressions, including wild cards, which is used to connect the matching needs and abilities based on their predicates and attributes.

Tool support. As a general-purpose monitoring and debugging tool, we implemented a tool that presents a graphical view of the network of interconnected services that dynamically changes when the system configuration changes. It also allows developers to view arbitrary event streams in the system, which proved invaluable for debugging (see figure 3).

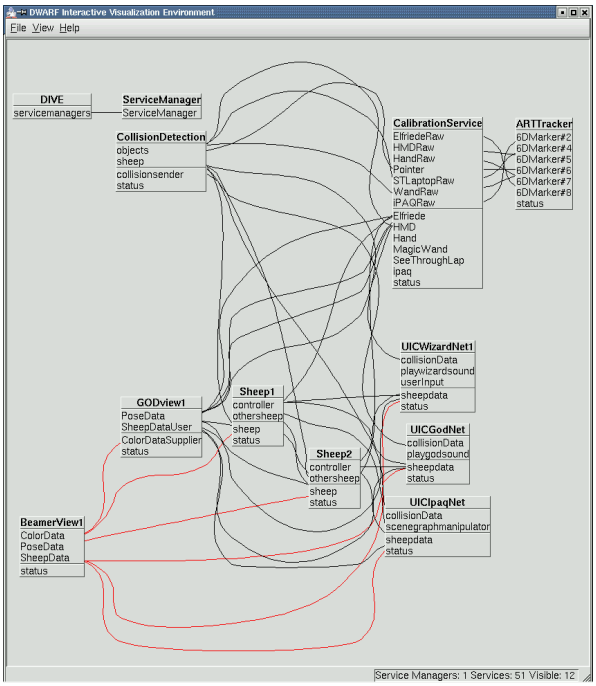

Figure 3. Monitoring and debugging tool for distributed services

Using DWARF, we have previously built Pathfinder, an experimental system for campus navigation; FATAMORGANA for the visualization of prototype automobile designs; TRAMP for automobile repair; FIXIT for machine maintenance; and PAARTI, prototype vehicle construction. We also built SHEEP, a multiplayer collaborative game (figure 1) with tangible user interfaces. Finally, new projects based on DWARF are underway. At the time of writing, these include ARCHIE, a collaborative architectural design system, HEART, a system for cardiac surgery, and PONG, a set of simple Augmented Reality games authored with a Python-based scripting interface.

[1] `http://omniorb.sourceforge.net`
[2] `http://openorb.sourceforge.net`
[3] `http://www.openslp.org`

Benefits

Our project is the first that uses CORBA to realize an Augmented Reality system. Since that first prototype in 2000 several others have been built based on the described concept. To use CORBA as the underlying middleware had several advantages over alternatives such as socket communication, Java RMI, SOAP, DCOM, or proprietary solutions.

Platform and language independence. CORBA allows us to develop our services in a broad range of programming languages. In particular, we use C++, Java, and Python. With CORBA we are free to chose the development language from case to case. Usually we chose programmin language and operating system because we need to build against a third-party library to implement a service. Currently we have a mixture of services deployed on workstations and laptops running Apple Mac OS X, Linux, and Microsoft Windows. We even have services running on a Compaq iPAQ with Linux.

Rapid prototyping. As we used the concepts of the DWARF framework, all developers were forced to use components running as separate processes. This lead to a clear and lean definition of the communication between different components in CORBA IDL files. In addition, the relationship between components of a running system could be displayed dynamically. This allowed us the very late integration of several student developers in the overall process without the need for lengthy documentation of the current system state, a simple look at the debugging tool sufficed. In fact, several people unfamiliar with the overall system were given and completed successfully self-contained tasks such as adding a collision detection component two weeks before the final deadline of the overall system.

Use available hardware. CORBA is running on every hardware and operating system that is already available at our chair. So we can concentrate on developing the service functionality instead of thinking about the best hardware/operating system combination for a new system.

Integration of third-party hard- and software. The implementation of Augmented Reality systems is very laborious if every component should be built from scratch. On the other side, there are several components available from other research groups that could be reused. Quite often these components require a fixed runtime environment, porting would be to costly. CORBA helps us to develop wrappers around these components in the required programming language and operating system.

Remote monitoring. As a general-purpose monitoring and debugging tool, we implemented a tool called the DWARF Interactive Visualization Environment. In analogy to visual programming interfaces in systems such as AVS, this tool presents a graphical view of the network of interconnected services that dynamically changes when the system configuration changes. It also allows developers to view arbitrary event streams in the system, which proved invaluable for debugging. In addition, the tool let us explain the running system to technically interested spectators and illustrate the distributed nature of DWARF.