Herding Sheep:
Live System Development for Distributed Augmented Reality

Asa MacWilliams, Christian Sandor, Martin Wagner, Martin Bauer,
Gudrun Klinker and Bernd Bruegge
Technische Universität München, Fakultät für Informatik
Boltzmannstraße 3, Garching bei München, Germany
(macwilli, sandor, wagnerm, bauerma, klinker, bruegge)@in.tum.de

Abstract

In the past, architectures of Augmented Reality systems have been widely different and taylored to specific tasks. In this paper, we use the example of the SHEEP *game to show how the structural flexibility of* DWARF*, our component-based Distributed Wearable Augmented Reality Framework, facilitates a rapid prototyping and online development process for building, debugging and altering a complex, distributed, highly interactive AR system.*

The SHEEP *system was designed to test and demonstrate the potential of tangible user interfaces which dynamically visualize, manipulate and control complex operations of many inter-dependent processes.* SHEEP *allows the users more freedom of action and forms of interaction and collaboration, following the tool metaphor that bundles software with hardware in units that are easily understandable to the user. We describe how we developed* SHEEP*, showing the combined evolution of framework and application, as well as the progress from rapid prototype to final demonstration system. The dynamic aspects of* DWARF *facilitated testing and allowed us to rapidly evaluate new technologies.* SHEEP *has been shown successfully at various occasions. We describe our experiences with these demos.*

1. Introduction

In the past, architectures of Augmented Reality (AR) systems have been widely different and taylored to specific tasks. For example, AR systems in task-centered, well-structured activities such as aircraft maintenance are traditionally differently organized than AR systems for ubiquitous computing environments. In [2], we presented the design of DWARF, the Distributed Wearable Augmented Reality Framework, claiming that a component-based framework had key advantages for users and developers.

It is time to verify and update these claims. We have used and tested DWARF to build a series of different application platforms — the most recent one being SHEEP, the Shared Environment Entertainment Pasture [21]. In this paper, we show at the example of SHEEP how the inherent structural flexibility of DWARF was crucial to a rapid prototyping and online development process for building and simultaneously debugging and altering a complex distributed, highly interactive AR system. The dynamic nature of DWARF allows developers and users to collaboratively test and further develop a running system. The result is a continuously running testbed to rapidly build prototypes and extend them to full systems.

We begin by presenting our motivation for building a multimodal, multiplayer game with DWARF, relating our approach to previous work. Next, we present the architecture of the SHEEP system. We show how existing and new DWARF components were used in the areas of tracking, sheep simulation, visualization, interaction and middleware. Afterwards, we describe the process of developing SHEEP and present required tools for continuous system development, integration and testing. We conclude by discussing the lessons we learned from developing and presenting the system and suggest options for future improvements to our framework.

1.1. Background

The DWARF project started in 2000 as a research platform for mobile and distributed AR systems. The framework contains *services* for position tracking, three-dimensional rendering, multimodal input and output and task modeling, which are used to build complete augmented reality applications. Distributed *middleware* manages the communication between services. The applications are designed to be very flexible and allow the user a great degree of freedom; they run in many different environments and hardware configurations.

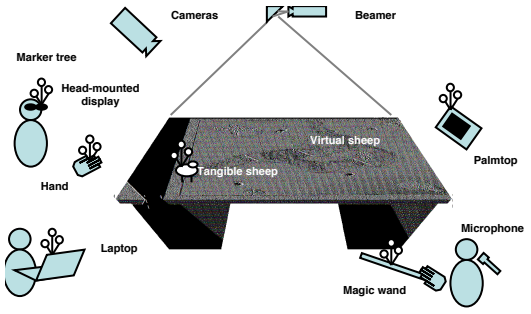

Figure 1. Game setup

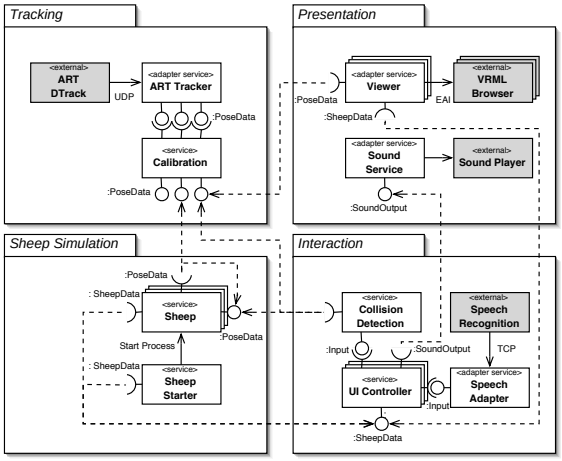

Figure 2. System architecture: services arranged by subsystems. Services communicate via their abilities (circles) and needs (semicircles). Third-party components are shown in gray.

Previous Systems Using DWARF, we have previously built experimental systems for campus navigation [2], vizualization of prototype automobile designs [11], machine maintenance [7] and prototype vehicle construction [8].

In each of these systems, a group of 5 to 50 students, often new to the field of AR, used the existing DWARF services to create an initial prototype within a timeframe of up to three months. In two projects that were later continued with industry partners [8, 11], the DWARF middleware and individual services (e.g. for rendering) were replaced with proprietary components that were already in use in the industry partners' computing environment.

Why SHEEP? The SHEEP system was designed to test and demonstrate the potential of tangible user interfaces which dynamically visualize, manipulate and control complex operations consisting of many inter-related processes and dependencies. On the basis of this project, we were able to evaluate and refine the architectural claims of DWARF and consolidate the system services, as well as distributed tracking, online calibration and multimodal interaction technologies. The concepts were presented at last year's ISMAR in Darmstadt.

Each of the previous DWARF systems was designed for a specific, often industrial application. In SHEEP, we wanted a less directed application that would allow the users more freedom of action, letting us experiment with various forms of interaction and collaboration. This lead to the idea of a multiplayer shepherding game centered around a table with tangible and virtual sheep in a pastoral landscape.

The Game The game is centered around a pastoral landscape which is projected onto a table from above (figure 1). On the table, a herd of virtual sheep roams around the pasture. Each virtual sheep is aware of the others' positions. It tries to stay close to the rest of the herd while avoiding collisions with other sheep. When a player puts a tangible, 'real' sheep on the table, the virtual sheep recognize it as a member of the herd and move towards it.

In order to add a new sheep to the game, the user points at the table with a magic wand and says 'insert.' A new sheep appears at the point where the wand touched the table. Sheep can be removed from the game, too, with a touch of the magic wand and the command 'die.'

With a small palmtop computer, a player can pick up a sheep. It then disappears from the table and appears on the palmtop's screen. The player can put the sheep down somewhere else on the pasture.

Using a see-through head-mounted display, the user can view the landscape three-dimensionally. He can pick up a sheep with his hand and inspect it. He can color the sheep by moving it into virtual color bars. Finally, he can put the sheep back on the table where it then rejoins the herd.

Spectators can see a three-dimensional view of the landscape on the screen of a laptop that can be freely moved about. The sheep appear in their correct three-dimensional position, even when picked up by another player.

1.2. Related Work

The idea for the SHEEP system was heavily inspired by the cow-painting demo from *Studierstube* [17]. Other examples for AR games include the well known ARQuake [25] and AquaGauntlet [14].

Distribution of our system is handled using a CORBA based middleware, which was first proposed in the context

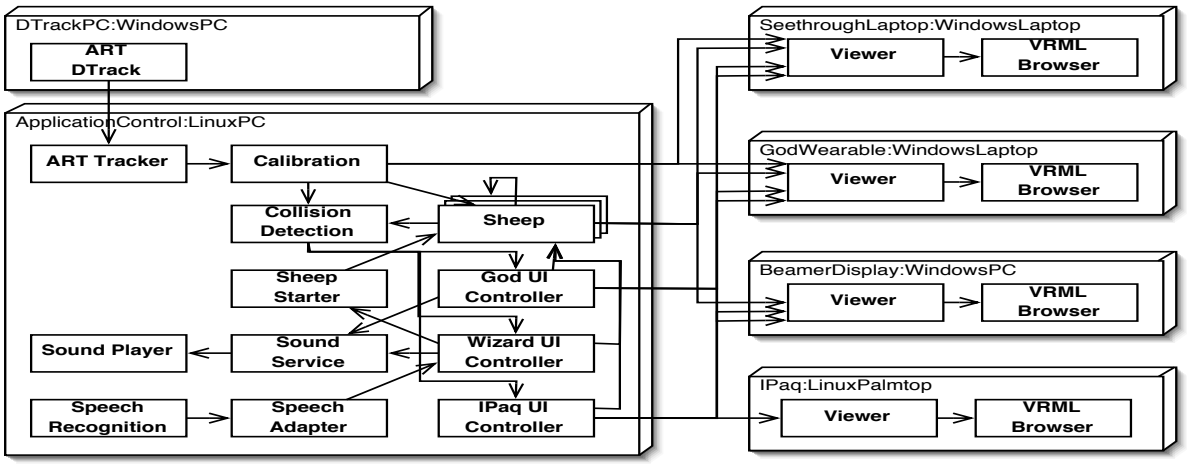

Figure 3. System architecture by deployment. The boxes are services and external components; The arrows indicate data flow. The system consists of three stationary and three mobile computers.

of VR applications by Kim et al. in the COVRA project [9].

There is quite a lot of effort going in the direction of architectures and frameworks for developing AR applications in general. Typically, these systems work towards abstracting from the idiosyncrasies of specific input and output devices by generating more general interaction facilities [12, 15, 16, 22, 26]. Most of them require configuration parameters, such as a specification of the input and output devices in use or the network adresses of different components. These parameters must be known at least at compile time, sometimes even before.

There is some effort towards facilitating the configuration of an AR system. The Studierstube project [23] has a modular design based on the Open Inventor architecture allowing rapid prototyping by scripting the system's functionality. The AMIRE project [6] aims at an architecture that allows easy authoring of AR content. In contrast to these approaches, our concept allows a dynamic reconfiguration or substitution of components at runtime without the necessity to restart or even recompile other parts of the systems. However, most ideas from the projects just mentioned may easily be added to our system. In addition, these projects may incorporate some of our components, e.g. for interaction control, using the DWARF open source distribution.

Recently, systems have started evolving towards allowing a static configuration of tracking devices before starting the system [18]. Our design approach goes one step further and focuses on the possibility to have a system that gets configured dynamically at runtime as new devices appear or new services are started.

2. System Architecture

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

2.1. Overview

The architecture of the SHEEP demonstration system is shown in figure 2.

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

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

The services are distributed on several machines, as shown in figure 3. In DWARF, we follow the tool metaphor [3], bundling software with hardware in units that are easily understandable to the user. For example, the palmtop system performs the complete 3D rendering locally

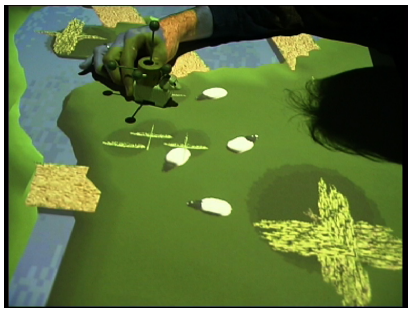

Figure 4. Interaction via a tangible sheep

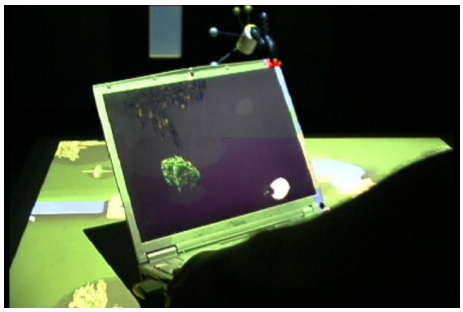

Figure 5. Two views of the same landscape: beamer and see-through laptop

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

2.2. Tracking

The tracking subsystem serves as the primary connection between the real and virtual world. Currently, we use the optical *DTrack* system from ART GmbH — a stationary outside-in optical tracking system which is able to track up to 10 marked users, objects and devices within a $4m \times 4m$ area at submillimeter precision.

The tracking package of figure 2 shows the simple pipe and filter architecture of the SHEEP tracking subsystem. The *DTrack* system is abstracted using a service that transforms its UDP stream to several DWARF pose data event streams, containing position and orientation. These are processed by a calibration service yielding positional information that fits the needs of the remaining SHEEP application.

The calibration procedure was very simple: in a first step, we manually calibrated a magic wand such that its tip's position could be used as a pointing device. The calibration of the projected image on the table was achieved by defining the origin of the ART tracker's coordinate system be in the lower left corner of the table image and by aligning the z-axis with the left border of the image. This allowed us to implement a simple and convenient one-click process for calibrating arbitrary objects with a rigidly fixed ART marker. The user only has to align the object with the left border of the projection table, point to the object's desired center using the magic wand and click on a button in the GUI of the calibration service. Although this calibration procedure is rather imprecise, it proved to be sufficient for our purposes. As with most components of the SHEEP system, the focus was on ease of use rather than high precision.

The dynamic, testbed-like character of DWARF encourages and supports this kind of 'just good enough' rapid prototyping — providing the option to start conceiving a more precise calibration routine and adding it to the SHEEP system whenever deemed necessary.

2.3. Sheep Simulation

The behavior of the virtual sheep is based on a distributed variant of a simulated flock of birds [19].

Each sheep is a single process that is connected to the other sheep and receives their position information. It then tries to stay close to the center of the herd while at the same time avoiding collisions with other sheep.

In addition to the virtual sheep we also have a toy plastic sheep (figure 4). It is tracked and can be moved freely by the user. For the virtual sheep, the tracked position information of the toy sheep is indistinguishable from the virtual sheep. Thus, users can guide the herd by moving the plastic sheep — the virtual herd will center around it and follow it.

The sheep simulation at this point is not highly advanced. For example when introducing a second plastic sheep, the virtual herd will center in the middle between the two plastic sheep. This component could be easily replaced by a better simulation — and, in fact, several variants of the simulations were used and dynamically exchanged in early stages of the system. For some time we had the sheep only running around in circles until a more stable simulation code was ready for deployment.

2.4. Visualization

The SHEEP scene is shown in several views: a beamer-view displayed on the table, several mobile 3D-views on tracked laptops, see-through views on head-mounted displays and simplified sheep presentations on palmtop computers (Compaq iPAQ) (figures 4, 5, 6, 7). The views are generated by different instances of 3D viewers running under x86 Windows or StrongARM Linux.

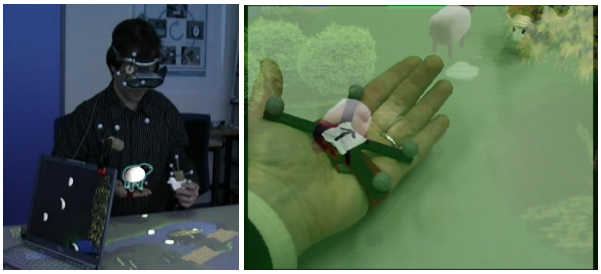

Figure 6. View through head-mounted display

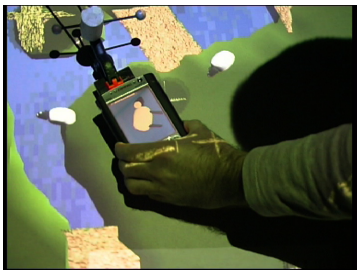

Figure 7. Display of 3D content on the iPAQ

Scene descriptions are provided in VRML. A small adapter service written in Java communicates via the External Authoring Interface (EAI) with a VRML browser, such as Cortona from Parallelgraphics[1] on Windows platforms, and FreeWRL[2] on the palmtops. The viewers interface to the *User Interface Controller* component (see section 2.5) which maintains the state of the user interface. According to the current state, viewers add or remove objects, such as new sheep, from the scene and display them as specified by their actual properties. The viewers also interface to tracking components to set their viewpoints and to position tracked objects and virtual sheep.

We encountered various problems in implementing this architecture of the viewing component, such as system incompatibilities and performance discrepancies across platforms that made us provide several versions of VRML objects for different views. To get acceptable performance on the iPAQ, we had to compile the EAI libraries of FreeWRL, which are written in Java, into native code with `gcj`[3], yielding a big performance boost from 0.01 to 0.3 frames/sec for a scene containing 162 polygons in a 320 times 200 resolution (see figure 7) — which is still too slow, but is mainly due to the lack of hardware acceleration and even worse the lack of a floating point unit on the iPAQ.

Furthermore, the EAI is not set up to support the required communication traffic (several 100 position updates per second) and the dynamic nature of the scene descriptions when sheep are added and removed at the user's will. We encountered limitations in the use of the VRML `EXTERNPROTO` mechanism: Because of the EAI requirement that every accessable object has to have a predefined, unique name field, all sheep that are potentially created during the course of a game have to be planned for in advance when the scene is described in the VRML code. The result is a rigid object structure, too large for small demonstrations, and potentially too small during a game with many users.

[1] http://www.parallelgraphics.com
[2] http://freewrl.sourceforge.net/
[3] http://gcc.gnu.org/java/

Because of these problems, we are currently reimplementing our viewer based on the open-source Open Inventor implementation Coin3D[4]. Due to the flexible architecture of DWARF new experimental viewing components can be brought into the system arbitrarily. They can co-exist and mix with the current viewing components.

2.5. Interaction

Interactions in SHEEP were implemented using the *User Interface Controller* DWARF service. The User Interface Controller combines the functionalities of dialog control and discrete integration. The software adapters for the input devices emit tokens that are received by the User Interface Controller. In the case of SHEEP these are: speech recognition and collision detection for tangible interaction. According to the state of the user interface (which is kept in the User Interface Controller) and the tokens that are received, actions are triggered. These actions are dispatched to services in the presentation layer and cause a change, addition or removal of display elements. The rule-based evaluation of the user input is encapsulated into guards that check whether a transition is legal.

We use Petri Nets to model multimodal interactions — as is common practice in the area of workflow systems [1]. Figure 8 shows the flow of events within the Petri Net while a user points the magic wand at a location on the table and utters the word 'insert' in order to add a new sheep to the pasture. The User Interface Controller receives two tokens from the input drivers, as shown in the lower row of figure 8: one that represents the collision wand/table and one for the 'insert' speech command. These tokens are placed onto places in the Petri Net. After all places on incoming arcs of a transition are full, the transition is triggered, resulting in the creation of a new sheep.

The User Interface Controller is based on the Petri Net framework Jfern[5] which provides large parts of the functionality needed for this component. The Petri Nets that

[4] http://www.coin3d.org
[5] http://sourceforge.net/projects/jfern

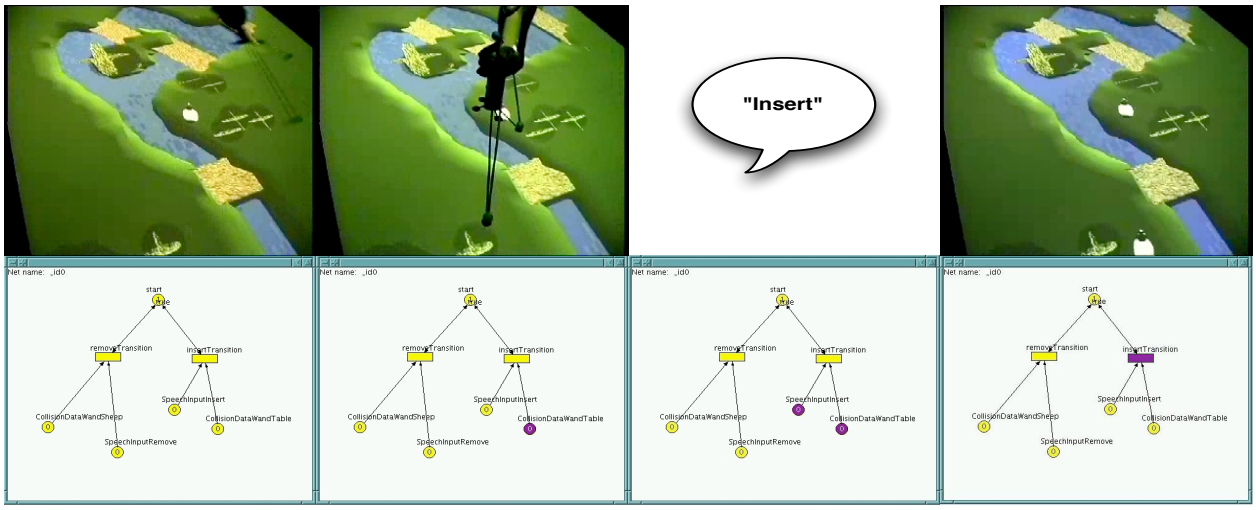

Figure 8. Realization of a multimodal point-and-speak user interface with a Petri Net within the SHEEP game. The second row shows the tokens inserted in the Petri Net by the interactions in the first row.

model the multimodal interactions for a user are written in XML. From these descriptions Java classes are generated. Jfern also allows the graphical display of the Petri Net and its current state. These graphical representations are very useful for debugging.

Future work on this service will include the collection of interaction patterns to form an libary of interaction Petri Nets. Because of the common token format, it is possible to reuse a Petri Net within different applications and even with different input devices, as long as they deliver the needed tokens to the User Interface Controller.

2.6. Middleware

DWARF is based on distributed *services*. The services are managed by distributed CORBA-based middleware. 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.

The stationary computers are connected together using standard 100 megabit ethernet; the 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. Since the current implementation of the service manager does not run on Windows, the Windows machines must connect to a service manager on a specified remote host. 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 (interface definition language). Some services have CORBA interfaces with specified methods that are called by other services; most, however, communicate using CORBA structured events. Events sent over the network via the CORBA notification service incurred a latency of approximately 3 ms in our setup.

The implementation of the DWARF service manager uses the open-source OmniORB CORBA implementation [6], 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 [7], and JavaORB, its predecessor, which was necessary for the Java 1.1 virtual machine needed to control

[6] http://omniorb.sourceforge.net
[7] http://openorb.sourceforge.net

the Cortona VRML browser under Windows.

To find services managed by other service managers, the service managers use OpenSLP [8], an implementation of the Service Location Protocol SLP. This allows services to be found based on type and boolean predicates over named attributes, which proved to be completely sufficient.

The API of the middleware is specified in CORBA IDL and was slightly extended from previous versions of DWARF. In particular, the concept of sessions was introduced, allowing services to retrieve information about their communication partners at run time. For example, the VRML display adds a new sheep to the scene graph when a new session from a sheep service is established.

For future versions of the framework, the middleware will be ported to additional platforms (notably Windows; Mac OS X support has already been implemented). Also, the speed with which services find each other needs to be improved; currently, this can take up to several seconds.

3. Development

In this section, we describe how we developed SHEEP. It shows the combined evolution of framework and application, as well as the progress from rapid prototype to final demonstration system. We also present several tools we developed to help us test and integrate the final system.

3.1. Development Process

The development of SHEEP took place from April to September 2002 with the goal to demonstrate a running system at ISMAR'02 on September 30. Aside from the authors, participating developers included five students. Before the final "hot phase" of development in the last month, developers participated on a part-time basis.

Iterative approach The development of SHEEP did not proceed in a linear fashion from specification to implementation. Rather, we used an iterative approach, regularly reassessing the progress made in service implementation and system integration. We then modified our target accordingly; it was more important for us to have a working system in time for the demo presentation than to have a fully developed game.

Initial concept The initial concept of a multiplayer game was conceived in April. After several discussion rounds in our augmented reality seminar, we decided on the shepherding scenario. This scenario included several features that were not implemented in time for the demonstration system, due to the extra hardware and development time they would have required: a virtual wolf to chase the sheep, manipulations to the landscape such as moving trees, sheep correctly crossing bridges, a "paint-the-sheep" user interface on the palmtop, and an installation CD-ROM for spectators who bring their own laptops and wish to participate in the game.

Mapping onto services As a first step, we mapped the functionality of the scenario onto existing and new DWARF services. This was mostly straightforward. For example, we obviously needed a viewer service to render the three-dimensional scene, and we obviously needed a tracking service for the ART tracking hardware we planned to use. Also, we decided to model each sheep as a separate service (rather than one single 'herd' service) to further experiment with the distributed nature of DWARF and to demonstrate its ad hoc, peer-to-peer connectivity in a heterogeneous environment — potentially allowing spectators at ISMAR to add and control sheep from their own laptops.

In other cases, it was not quite so clear; for example, the collision detection service was a matter of discussion for several weeks. Should collisions be detected by a separate service (the final solution) or within another existing service, such as a viewer or tracker? In a similar manner, we debated whether to use a central world model service or not; finally, we settled on a simple decentral exchange of `SheepData` events when a sheep died or changed color.

Design during implementation In these cases where the mapping was difficult, we deferred the design decisions until later in the project, and started to implement those services whose design was already clear. This allowed us to rapidly evaluate the new technologies and base our design decisions on what was actually technically possible.

Service implementation For SHEEP, we needed to refine existing services, implement new services based on old ones, and implement completely new services. For example, the ART tracking service was refined to allow a flexible per-marker configuration, and the VRML viewer was refined to allow more simultaneous manipulations to the scene. The sheep simulation services were implemented from scratch, and the UI controller was re-implemented to use Petri Nets rather than finite state machines.

Integration tests During development, the dynamic aspect of the middleware facilitated testing; different versions of services can be started in any order, and they are connected together by the middleware.

We used this to perform integration tests as early as possible during development. The first tests were two-service tests. For example, we tested an early version of the sheep simulation with an early version of the VRML viewer, and

[8] http://www.openslp.org

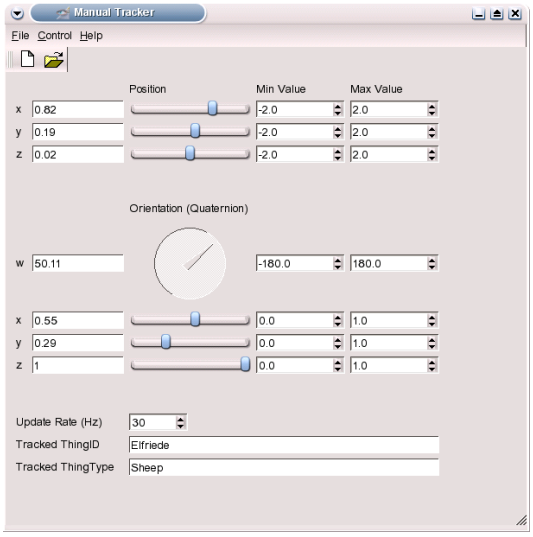

Figure 9. Simulation of arbitrary pose data

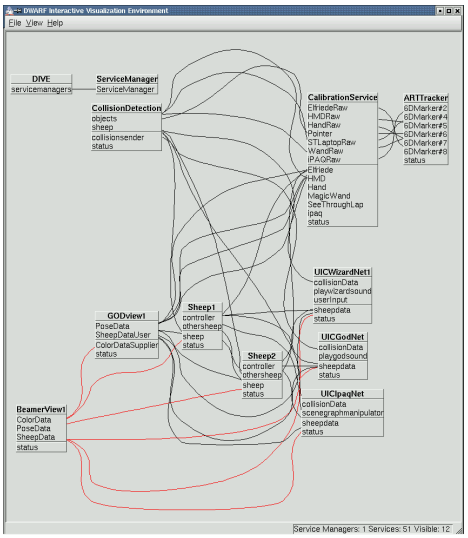

Figure 10. Monitoring and debugging tool for distributed services

saw a single sheep flying in circles around the pasture. This way, we ensured that the interfaces between services were developed very early in the project, and that we would not run into too many unpleasant surprises during integration later on. The early versions of simple services were useful to other developers — in this particular case, the circling sheep could be used to debug the VRML viewer.

Jam sessions — development at run time Towards the end of the project, we introduced 'jam sessions'. All participating developers met in our lab and programmed and tested together. Inspired by ideas from extreme programming [4], we reviewed code of individual services in pairs to ensure the code's correctness.

In the jam sessions, we performed larger integration tests of three to twenty services, in order to test the entire system's functionality. The decentral nature of the middleware meant that it was rarely necessary to shut down and restart the entire system during testing; individual parts were always kept running. This meant that when a fault was found in a service during an integration test, the service's developer took that service off-line, repaired, recompiled and restarted it. Since the rest of the system was still running, the other developers could continue testing, and finally, the improved service rejoined the rest of the system.

3.2. Development Tools

Test services Single DWARF services are not very useful on their own. In fact, they are difficult to develop on their own without additional services for testing and debugging. For this, we implemented several test services, e.g. the manual tracker, which can send arbitrary pose data to the rest of the system (figure 9). Additionally, we used early versions of services, such as the circling sheep, to test other services.

Monitoring tool As a general-purpose monitoring and debugging tool, we used the DWARF Interactive Visualization Environment (figure 10), which was developed in parallel to the SHEEP system. In analogy to visual programming interfaces in systems such as AVS [10], 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.

4. Results

We completed the first version of the SHEEP system in September 2002. Since then, it has been shown off at various occasions. This section sums up our experiences with these demonstrations.

4.1. Demos

SHEEP was shown at ISMAR 2002's demo session for the first time in public. After some minor changes that enhanced the speed and stability of the system, we installed

SHEEP as a permanent exhibition in our lab and demonstrated it to a mainly non-expert audience at various occasions. Although we did not conduct any real studies on the usability of the system, we got some anecdotical experience regarding the feasibility of our approach for augmented reality based games.

4.2. User Feedback

During a TUM open lab day, about 100 non-experts tried out SHEEP. We got valuable feedback especially from children. In general, the tool metaphors used in SHEEP were very clear, the interactions involving the iPAQ, the see-through laptop, the wizard with the magic wand and the tangible sheep were understood almost without any explanations. This observation underlines Svanaes' and Verplank's claim [24] of good usability of magical metaphors. In contrast to the good results of interaction metaphors differing from classical AR approaches, our implementation of 3D overlays using a HMD had several problems. Direct interaction with floating menus being fixed in the user's field of view is problematic, see also [5].

Most people tried to keep their hand's position in the middle of their viewing frustum and were therefore unable to complete the required interaction of touching colored bars at the top of this frustum (see figure 11). In addition, this interaction metaphor depends on the user's physiognomy. Children's arms were in general too short to reach the bars which were placed at a distance that is within easy reach of an adult.

4.3. Developer Feedback

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 (see figure 10). 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 SHEEP 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.

5. Conclusion and Future Work

The SHEEP project was successful in fulfilling our goals. First, we managed to set up a well-equipped testing and development lab for experimentation with tangible multimodal interactions. Second, several essential framework services, such as the ART tracking service, the calibration service and the user interface controller were consolidated from previous versions and integrated into the framework. The problems we had with the VRML Viewer showed us the need for a redesign of this service.

Figure 11. The view of the god player through the HMD. Sheep can be colored by moving them into one of the three color bars that are displayed head-fixed.

The work presented in this paper can be carried on in three major directions: improvement of SHEEP, improvement of DWARF and future DWARF projects.

Several extensions are possible to the SHEEP game itself: a virtual wolf that chases the sheep, improved sheep simulation so that the sheep correctly cross bridges and avoid rivers, user manipulations to the landscape such as moving trees, or a "paint-the-sheep" user interface on the palmtop. With the infrastructure we have now in our lab, we can start to conduct usability studies and thus refine the interactions even more.

The framework itself is constantly evolving, and will continue to do so. Extensions that are already underway include a distributed three-dimensional world model, a more powerful and efficient Inventor-based viewer implementation, more accurate and yet simple calibration mechanisms, and other trackers. In addition, work is underway to let the user switch between several simultaneously running applications on the distributed system.

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.

Acknowledgements

Special thanks go to our students Daniel Pustka, Franz Strasser, Gerrit Hillebrand, Marco Feuerstein, Ming-Ju Lee and Otmar Hilliges for developing many useful components and tools, without these this work would not have been possible. The tracking system used for SHEEP was partially on loan from BMW (TI-360). This work was partially supported by the High-Tech-Offensive of the Bayerische Staatskanzlei.

References

[1] W. AALST, *The Application of Petri Nets to Workflow Management*, The Journal of Circuits, Systems and Computers, 8 (1998), pp. 21–66.

[2] M. BAUER, B. BRUEGGE, G. KLINKER, A. MACWILLIAMS, T. REICHER, S. RISS, C. SANDOR, and M. WAGNER, *Design of a Component–Based Augmented Reality Framework*, in Proceedings of the 2nd International Symposium on Augmented Reality (ISAR 2001), New York, USA.

[3] M. BAUER, B. BRUEGGE, G. KLINKER, A. MACWILLIAMS, T. REICHER, C. SANDOR, and M. WAGNER, *An Architecture Concept for Ubiquitous Computing Aware Wearable Computers*, in Proceedings of 2nd International Workshop on Smart Appliances and Wearable Computing 2002.

[4] K. BECK, *eXtreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[5] D. A. BOWMAN and C. A. WINGRAVE, *Design and Evaluation of Menu Systems for Immersive Virtual Environments*, in VR 2001, pp. 149–156.

[6] R. DÖRNER, C. GEIGER, M. HALLER, and V. PAELKE, *Authoring Mixed Reality. A Component and Framework-Based Approach*, in Proceedings of First International Workshop on Entertainment Computing, Makuhari, Chiba, Japan.

[7] F. ECHTLER, H. NAJAFI, and G. KLINKER, *FixIt*, in Demonstration at the International Symposium on Augmented and Mixed Reality (ISMAR 2002), Darmstadt, Germany.

[8] F. ECHTLER, F. STURM, K. KINDERMANN, G. KLINKER, J. STILLA, J. TRILK, and H. NAJAFI, *The Intelligent Welding Gun: Augmented Reality for Experimental Vehicle Construction*. Submitted to the International Journal of Automation in Manufacturing Technology, 2003.

[9] S. KIM, U. YANG, N. KIM, and G. J. KIM, *COVRA-CAD: A CORBA based Virtual Reality Architecture for CAD*, in Proceedings of International Conference on Virtual Systems and Multimedia.

[10] G. KLINKER, *An Environment for Telecollaborative Data Exploration*, in Visualization '93, IEEE Computer Society Press, pp. 110–117.

[11] G. KLINKER, A. DUTOIT, M. BAUER, J. BAYER, V. NOVAK, and D. MATZKE, *Fata Morgana – A Presentation System for Product Design*, in Proceedings of ISMAR 2002, Darmstadt, Germany.

[12] B. MACINTYRE, *Exploratory Programming of Distributed Augmented Environments*, PhD thesis, Columbia University, 1999.

[13] T. OSHIMA, *RV-Border Guards: A multiplayer entertainment in mixed reality space*, in Poster session of IEEE International Workshop on Augmented Reality (IWAR 1999), San Francisco, USA.

[14] W. PIEKARSKI and B. H. THOMAS, *Tinmith-evo5 - An Architecture for Supporting Mobile Augmented Reality Environments*, in Proceedings of ISAR 2001.

[15] H. REGENBRECHT and M. WAGNER, *Interaction in a Collaborative Augmented Reality Environment*, in Proceedings of CHI 2002, Minneapolis, USA.

[16] G. REITMAYR and D. SCHMALSTIEG, *Mobile Collaborative Augmented Reality*, in Proceedings of ISAR 2001, New York, USA.

[17] G. REITMAYR and D. SCHMALSTIEG, *OpenTracker–An Open Software Architecture for Reconfigurable Tracking Based on XML*, in Proceedings of VR, pp. 285–286.

[18] C. W. REYNOLDS, *Flocks, Herds, and Schools: A Distributed Behavioral Model*, in Computer Graphics, SIGGRAPH '87 Conference Proceedings, pp. 25–34.

[19] C. SANDOR, A. MACWILLIAMS, M. WAGNER, M. BAUER, and G. KLINKER, *SHEEP: The Shared Environment Entertainment Pasture*, in Demonstration at ISMAR 2002, Darmstadt, Germany.

[20] D. SCHMALSTIEG, A. FUHRMANN, and G. HESINA, *Bridging multiple user interface dimensions with augmented reality*, in Proceedings of the 1st International Symposium on Augmented Reality (ISAR 2000), Munich, Germany.

[21] D. SCHMALSTIEG, A. FUHRMANN, G. HESINA, Z. SZALAVARI, L. M. ENCARNAÇÃO, M. GERVAUTZ, and W. PURGATHOFER, *The Studierstube Augmented Reality Project*, Presence, 11 (2002).

[22] D. SVANAES and W. VERPLANK, *In Search of Metaphors for Tangible User Interfaces*, in Proceedings of Designing Augmented Reality Environments (DARE 2000).

[23] B. THOMAS, B. CLOSE, J. DONOGHUE, J. SQUIRES, P. D. BONDI, M. MORRIS, and W. PIEKARSKI, *ARQuake: An Outdoor/Indoor Augmented Reality First Person Application*, in Proceedings of International Symposium on Wearable Computers (ISWC 2000), pp. 139–146.

[24] S. UCHIYAMA, K. TAKEMOTO, K. SATOH, H. YAMAMOTO, and H. TAMURA, *MR Platform: A Basic Body on Which Mixed Reality Applications are Built*, in Proceedings of ISMAR 2002, Darmstadt, Germany.