Integrating Studierstube and DWARF

Martin Bauer, Otmar Hilliges, Asa MacWilliams
Christian Sandor, Martin Wagner, Gudrun Klinker

Technische Universität München
Institut für Informatik

Joe Newman, Gerhard Reitmayr, Tamer Fahmy
Thomas Pintaric, Dieter Schmalstieg

Technische Universität Wien
Interactive Media Systems Group

***Abstract*— Studierstube and DWARF are modular Augmented Reality frameworks, each with distinct advantages in different application areas. Both can easily be extended by adding new components. In this case study we show how new components can act as a bridge between the frameworks. By facilitating the exchange of basic data types such as pose and user input, the frameworks become interoperable. This allows us to build new applications leveraging the advantages of both frameworks, while fostering cooperation between research groups.**

I. Introduction

A. Motivation

Almost all Augmented Reality (AR) systems developed over the last few years are either aimed at improving certain aspects of AR such as tracking and user interaction, or attempting to take AR into new application domains, ranging from aircraft wireboard assembly [7] to interactive theaters [6]. However, very little research has focussed on applying software engineering principles to the creation of reusable AR architectures.

This paper describes how component-based design of AR frameworks allows the easy integration of results obtained from different research projects, even if the primary goals of these projects are distinct. We show how the component-oriented design of both the Studierstube [25] and the DWARF [3] frameworks for AR systems allow us to encapsulate the particular strengths of each framework in components and then combine these components to create a system that is more than just the sum of its parts.

First we examine the related work and describe in detail the key features of DWARF and Studierstube and how they are used in typical scenarios. Technical details are then provided as to how the systems are capable of interacting seamlessly on the component level. Finally, a scenario is described illustrating some of the new possibilities for AR applications which can be developed using this approach.

B. Related Work

Research and commercial activities in Software Engineering provide strong evidence that componentizing software makes it reusable. The two most important comercial component-based frameworks are Microsoft .NET [19] and SUN ONE [28]. Both define a component model and communication mechanisms. These frameworks are often used to build client-server systems such as web applications. Several experimental frameworks support context-aware computing, e.g. GaiaOS [23], [12], Ninja [10], Aura [9], Oxygen [20], PIMA [2], Cooltown [14] or the Sentient Computing project [1]. Augmented Reality can be regarded as a subset of context-aware mobile computing, however, none of these projects (with the possible exception of the Sentient Computing Project) specifically addressed the requirements of AR such as the provision of low latency tracking data.

The Sentient Computing project used a wide-area ultrasonic location system, and other sensors to update a model representing environmental state. This model was built from components based on persistent CORBA objects, each corresponding to an associated real-world object. This component-based approach not only achieved the Software Engineering goals of code reusability and maintainability but also provided a natural interface which made it appear to users that their perceptions of the environment were shared directly by the system. A QoS scheme was responsible for sharing system resources between AR [21] services (demanding high bandwidth low latency data) and less needy context-sensitive applications.

A number of other AR frameworks have been developed, notably COTERIE [16], AMIRE [8] and VRJuggler [5]. For the most part both COTERIE and VRJuggler operate at the level of the scene graph in a similar fashion to Studierstube, whereas AMIRE is more closely related to DWARF in its formal distribution of components as “gems” and “MR components”. Other libraries such as ARToolKit [13] and the MR platform [29] greatly facilitate the development of new AR systems. However, the use of such libraries generally results in monolithic systems. The DWARF and Studierstube projects arguably provide the strongest component-based approach to implementing an AR framework.

In this paper, we describe the integration of these two frameworks. To our knowledge, no previous effort has been made to systematically combine two existing AR systems, besides the inclusion of libraries such as ARToolKit.

II. Key Features of the Frameworks

In this section, we describe the basic components of the two frameworks, and the features they support.

Studierstube, its OpenTracker library [22], and DWARF each consist of modular components, although typically these are of different granularity. Extensions to Studierstube and DWARF typically consist of the addition of new components.

Several basic data types exchanged by the frameworks' components are similar, such as 6-DOF pose, and input device data. Thus, we can create adapter components that are part of both frameworks to facilitate interoperation.

Since both frameworks were designed to make the addition of new components simple, developing the adapter components was straightforward. In fact, we were able to develop one such component, which allows OpenTracker to send pose data to DWARF, in a single afternoon of joint programming.

Below, we describe the basic components of DWARF and Studierstube in order to explain the adapter components.

A. DWARF

The basic units of the DWARF framework are distributed services. A *service* is a piece of software running on a stationary or mobile computer that provides a certain piece of functionality such as optical tracking. DWARF contains services for position tracking, 3D rendering, multimodal input and output, and modeling of user tasks. The framework can easily be extended by adding new services or improving existing ones.

Following the tool metaphor [4], the services and their associated support software is bundled with hardware in units that are easily understandable to the user, such as an HMD with a 3D rendering laptop, or a palmtop with a menu interface. The services are realized as individual processes or threads, run on different mobile and stationary computers and connect dynamically using wired or wireless networks.

This dynamic connection and loose coupling supports the building of flexible Augmented Reality applications in Ubiquitous Computing environments. Systems we have built so far [3], [24], [15] consist of between 10 and 50 services. The DWARF services themselves are designed to support runtime reconfiguration in order to provide high flexibility to the user. Examples include multimodal interaction services using dynamically chosen interaction devices, and a simple run-time marker recalibration service.

Distributed CORBA-based middleware manages the services. Each DWARF system network node has one *service manager*; there is no central component. Each service manager controls the node's local services and maintains descriptions of them. The service managers cooperate with each other to set up connections between services.

To model what a service can offer to other services and what it needs from other services, we use concept of *needs* and *abilities*. A match of one service's need to another service's ability leads to a connection between the services; this is set up by the distributed service managers.

Abilities describe the functionality a service provides, such as position data for optical markers. A service can have several abilities, such as an optical tracker that can track several markers simultaneously. Abilities are typed; an example is *PoseData* for 6D pose.

Needs describe the functionality required of other services. For example, an optical tracker needs a video sequence and descriptions of the markers it should detect, and a 3D renderer needs the position and orientation of the viewpoint it should render the scene from. Needs are also typed, and only abilities of the same type can satisfy a need.

The communication protocols currently supported in DWARF are CORBA notification service events, shared memory, and remote method calls. Most of the communication in typical DWARF systems uses events, with predefined structures for types such as *PoseData*. In Section III, we describe how these events can be used in Studierstube's OpenTracker data flow graphs and OpenInventor scene graphs.

B. Studierstube

Studierstube's software development environment is realized as a collection of C++ classes built on top of the Open Inventor (OIV) toolkit [27]. The rich graphical environment of Open Inventor allows rapid prototyping of new applications. The file format of Open Inventor enables convenient scripting, overcoming many of the shortcomings of compiled languages without compromising performance. At the core of Open Inventor is an object-oriented scene graph storing both geometric information and active interaction objects. Our implementation approach has been to extend Open Inventor as needed, while staying within Open Inventor's strong design philosophy [30].

This has led to the development of two intertwined components: a toolkit of extensions of the Open Inventor class hierarchy – mostly interaction widgets capable of responding to 3D events – and a runtime framework that provides the necessary environment for Studierstube applications to execute. Together, these components form a well-defined API, which extends the Open Inventor API and also offers a convenient programming model to the application programmer.

Applications are written and compiled as separate shared objects and dynamically loaded into the runtime framework. The shared objects are treated as singletons so that only one instance of each applications code is loaded into the system at any time. Besides decoupling application development from system development, dynamic loading of objects also simplifies distribution, as application components can be loaded by each host whenever needed. All these features are not unique to Studierstube, but they are rarely found in Virtual Environment software.

Studierstube also supports distributed execution of applications by providing a shared scene graph library called Distributed Open Inventor [11] (DIV) implemented using a reliable multicast protocol. In contrast to the DWARF model, DIV provides a closely coupled distribution service where each host is running a fully replicated instance of a distributed application. Because applications are implemented as scene graph nodes, they can be replicated as well. This is used at startup to stream a copy of an application to all members of a session, to update late joining hosts of the current status or to implement ubiquitous applications that move between hosts [26].

To accommodate tracking devices Studierstube relies on OpenTracker, a dedicated library that implements a pipe- filter architecture to operate on tracking data. The main concept

behind OpenTracker is to break up the required transformation of data manipulation into individual, smaller steps and build a data flow network of these generic operations. Such operations include reading data from devices, transforming it to fit the requirements of the application and sending over network connections to other hosts. Such data flow network is configured with an XML based input file which is read at startup.

OpenTracker already provides a large number of individual nodes implementing different steps. Source nodes provide device drivers for magnetic, ultrasonic, and inertial trackers, GPS receivers, or virtual devices such as keyboard and mouse events, files or network connections. Filter nodes reside in within the framework and provide geometric transformations, smoothing and merging of data, selecting between different sources. Sink nodes finally provide the transformed data to applications, send it over network connections or provide debugging output to files and consoles.

III. Generic Adapters

To leverage both Studierstube and DWARF, we identified two possible bridging points. First we show how an OpenTracker network can be used as a DWARF service or conversely DWARF can be used to connect different OpenTracker networks together dynamically.

Then we describe how an arbitrary DWARF service that sends or receives events can be embedded in an Open Inventor scene graph such that it can be used in a Studierstube application.

A. OpenTracker ↔ DWARF

The first step is the integration of the low-level tracking data framework OpenTracker into the DWARF system. The OpenTracker library is extensible with modules that encapsulate interfaces to devices, other software frameworks or algorithms. These modules drive nodes in the tracking graph that can generate new events, consume events or transform them. We designed a new DWARF module for OpenTracker that implements two types of nodes, a *DwarfSink* and a *DwarfSource* node. The module is responsible for the DWARF specific setup of the service and the needs and abilities corresponding to these nodes (see fig. 1).

A *DwarfSink* node is mapped to a set of abilities of a DWARF service serving as output of a given tracking graph, while a *DwarfSource* node is mapped to a set of needs of a DWARF service and inputs data into the graph. The module implements a full DWARF service that dynamically exposes its descriptions of needs and abilities based on the OpenTracker configuration. Therefore a single standalone OpenTracker program can act as a complete DWARF service. Depending on the application, the complexity of the OpenTracker network inside such a DWARF service can vary from large networks for a specific application to small networks containing just a few nodes for rather dynamic scenarios, as seen in figure 2.

OpenTracker defines a simple data type for events consisting of pose data using 3 floats for position, 4 floats for orientation

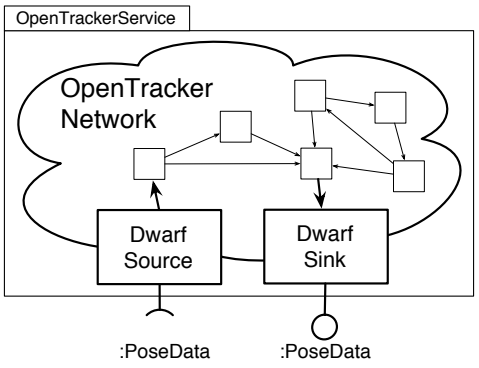

Fig. 1
A DWARF Service with Sources and Sinks in an OpenTracker Network

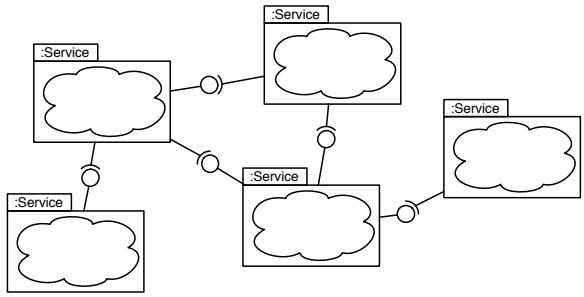

Fig. 2
A DWARF Network consisting of several OpenTracker Services

in quaternion format, a 16 bit value to store the state of 16 buttons, a time stamp and a confidence value in the range 0 to 1. This data type is mapped on a fixed set of DWARF types using one instance of *PoseData* to store the position and orientation information and a set of up to 16 *InputDataBool* instances to store button values. A corresponding set of abilities is defined for each *DwarfSink* and a set of needs for each *DwarfSource*. Figure 1 shows an encapsulation of an OpenTracker network into a DWARF service with the corresponding needs and abilities of type *PoseData*.

B. Open Inventor ↔ DWARF

To integrate DWARF with the Studierstube framework in general, we defined a more powerful mapping between the Open Inventor framework and DWARF. Open Inventor supports the notion of nodes in a scene graph that contain fields of a predefined set of types such as single or multiple values of floats, doubles, integers, vectors, rotations and matrices. Using a set of custom nodes we can model a DWARF service that sends or receives events directly.

A *DwarfService* node represents a single DWARF service. It provides fields to configure the general parameters of a service such as the service manager to use. Moreover it contains lists of subnodes that describe its needs and abilities. These are modelled by a *DwarfNeed* and *DwarfAbility* node respectively.

```
struct PoseData {
    ThingID id;     // a string
    ThingType type; // a string
    double position[ 3 ];
    double orientation[ 4 ];
    double positionAccuracy;
    double orientationAccuracy;
    Time timestamp;
};
```

$\Longleftrightarrow$

```
PoseData {
    SoSFString id
    SoSFString type
    SoSFVec3f  position
    SoSFRotation orientation
    SoSFFloat positionAccuracy
    SoSFFloat orientationAccuracy
    SoSFTime timestamp
}
```

Fig. 3

A DWARF EVENT TYPE DEFINED IN IDL AND ITS CORRESPONDING OPEN INVENTOR NODE DEFINITION.

Each node exposes fields to set the attributes and predicates of the DWARF counterparts.

The individual DWARF event types are mapped to different Open Inventor node types. Each node type defines fields for the components of the corresponding event type (see fig. 3). The *DwarfNeed* and *DwarfAbility* nodes then contain subnodes of the type corresponding to their Dwarf event type. A *DwarfNeed* node exposes incoming events by setting the field values of its subnode. A *DwarfAbility* node observes its subnode and fires new events to any connected services for any changes to the subnodes fields.

The description of Open Inventor nodes allows any Studierstube application to describe a DWARF service purely within the scene graph. The exposure of event data as fields leverages the typical Open Inventor mechanisms of field connections, sensors and callbacks to establish the data flow between application code and the DWARF adapter. Therefore DWARF services can be integrated without any additional knowledge on the side of the programmer.

C. Implementation status

The integration of DWARF and OpenTracker as described in section III-A has already been implemented in a single one-day-effort. The adapter to integrate arbitrary DWARF services in any Open Inventor application from section III-B has been exhaustively designed, but is currently being implemented.

IV. SYSTEM INTEGRATION: THE SMART HOSPITAL

In this section we describe a visionary scenario that can be realized with a combination of DWARF, Studierstube and OpenTracker components.

Medical scenarios provide very compelling Augmented Reality applications as clinicians are well trained, and hospitals are already full of sophisticated technology. The usage of PDAs as mobile knowledge bases is currently becoming a standard technique among doctors[1]. Clinical infrastructure, while being designed to handle very diverse tasks and disparate forms of data, would ideally achieve some degree of integration (through wired and wireless networks) to form coherent *sentient spaces* in which the environment becomes responsive to its occupants and particularly sensitive to the needs of medical staff. The scope for developing Augmented Reality applications in such an environment is huge, but different situations will demand very different characteristics from the underlying framework.

One such scenario might involve the provision of virtual presence to remote clinicians in an operating theatre during surgical procedures. Virtual presence would, for example enable a specialist consultant to easily examine the operation from different viewpoints while communicating to the surgeon (through a separate voice channel) with clinicians on the spot and possibly to manipulate diagnostic and other instruments. An extension of such a scenario would allow many students to be "virtually present" during a surgical procedure. Similar "virtual operating theatre" demonstrators already exist but do not operate in real-time and are entirely video-based reducing the scope for remote interaction due to occlusion.

The people present in the operating theatre would be tracked with an accurate commercial tracking system, whilst remote participants may use the same location technology or cheaper, simpler devices (with fewer modes of interaction) allowing them to view real-time synthesised, enhanced, and augmented views of the activity in the theatre.

The way in which sensory data generated by the location system is handled and provided to on-site and remote participants will depend on the nature of the infrastructure (such as trackers, and display end-points) as well as network constraints. Studierstube would be the tool of choice for representing and rendering complex 3D medical data (e.g. real-time bloodflow, or important biological structures) in the form of a scene graph which could be shared across multiple hosts through the DIV mechanism. However, spontaneous behaviour such as new users joining a session (or other users leaving) would be better managed by DWARF services, which would automatically perform the necessary connections and disconnections to the different data sources and sinks based on the availability of resources, and the requirements of the clients. In case the connections cannot be set up completely automatically, the DWARF Selector service [17], can be used to resolve ambiguities. This service has already been deployed on PDA devices, which would be a convenient device for the doctors to use, because they are already using them for access to knowledge bases.

V. CONCLUSION AND FUTURE WORK

An AR application developer can produce more elegant solutions when given a choice between all the components from Studierstube, OpenTracker and DWARF. For example,

[1]see e.g. http://www.neuroguide.com/aan_pda.html for a list of available knowledge sources for PDAs in medicine.

while Studierstube has not explicitly adressed the issue of multimodal interactions the DWARF User Interface Controller[18] can merge input distributed over several modalities into a single DWARF event representing the user's intention. An adapter (see section III-B) can then translate the DWARF event into an Open Inventor field change, which observed by Studierstube interaction objects.

By combining DWARF and OpenTracker we not only reduce the overhead of writing device drivers and filter objects multiple times. Additionally, we obtain a powerful combination of static local setups defined by OpenTracker networks which can be dynamically combined to form large scale DWARF applications in Ubiquitous Computing environments.

To our knowledge this was the first time two AR frameworks were rendered interoperable. We believe that this approach could start a positive trend in the Augmented Reality community. An ongoing discussion of the advantages of interoperability as other frameworks develop and mature can only benefit the field as a whole.

ACKNOWLEDGEMENTS

This work was partly sponsored by the Austrian Science Foundation *FWF* under contracts no. P14470 and Y193, and Vienna University of Technology by *Forschungsinfrastrukturvorhaben TUWP16/2002*. Partial support was also provided by the *High-Tech-Offensive Bayern* of the Bayerische Staatskanzlei.

REFERENCES

[1] M. ADDLESEE, R. CURWEN, S. HODGES, J. NEWMAN, P. STEGGLES, A. WARD, and A. HOPPER, *Implementing a Sentient Computing System*, IEEE Computer, 34(8), 2001, pp. 50–56.

[2] G. BANAVAR, J. BECK, E. GLUZBERG, J. MUNSON, J. SUSSMAN, and D. ZUKOWSKI, *Challenges: an Application Model for Pervasive Computing*, 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000), Aug. 2000.

[3] 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 2nd Intnational Symposium on Augmented Reality (ISAR), New York, Oct. 2001.

[4] M. BAUER, B. BRUEGGE, G. KLINKER, A. MACWILLIAMS, T. REICHER, C. SANDOR, and M. WAGNER, *An Architecture Concept for Ubiqutous Computing Aware Wearable Computers*, in International Workshop on Smart Appliances and Wearable Computing (IWSAWC 2002), 2002.

[5] A. BIERBAUM, C. JUST, P. HARTLING, K. MEINERT, A. BAKER, and C. CRUZ-NEIRA, *VR Juggler: A Virtual Platform for Virtual Reality Application Development*, in Proc. VR 2001, Yokohama, Japan, March 13–17 2001, IEEE, pp. 89–96.

[6] A. D. CHEOK, W. WEIHUA, X. YANG, S. PRINCE, F. S. WAN, M. BILLINGHURST, and H. KATO, *Interactive Theatre Experience in Embodied + Wearable Mixed Reality Space*, in International Symposium on Aumgented and Mixed Reality (ISMAR), Darmstadt, Germany, 2002.

[7] D. CURTIS, D. MIZELL, P. GRUENBAUM, and A. JANIN, *Several Devils in the Details: Making an AR Application Work in the Airplane Factory*, in Augmented Reality - Placing Artificial Objects in Real Scenes, R. Behringer, G. Klinker, and D. W. Mizell, eds., A K Peters, Ltd., 1999.

[8] R. DÖRNER, C. GEIGER, M. HALLER, and V. PAELKE, *Authoring Mixed Reality. A Component and Framework-Based Approach*, in Proc. IWEC 2002, Makuhari, Chiba, Japan, May 14–17 2002.

[9] D. GARLAN, D. SIEWIOREK, A. SMAILAGIC, and P. STEENKISTE, *Project Aura: Toward Distraction-Free Pervasive Computing*, IEEE Pervasive Computing, special Issue on Integrated Computing Environments, 1(2), 2002.

[10] S. D. GRIBBLE, M. WELSH, J. R. VON BEHREN, E. A. BREWER, D. E. CULLER, N. BORISOV, S. E. CZERWINSKI, R. GUMMADI, J. R. HILL, A. D. JOSEPH, R. H. KATZ, Z. M. MAO, S. ROSS, and B. Y. ZHAO, *The Ninja Architecture for Robust Internet-Scale Systems and Services*, Computer Networks, 35(4), 2001, pp. 473–497.

[11] G. HESINA, D. SCHMALSTIEG, and W. PURGATHOFER, *Distributed Open Inventor : A Practical Approach to Distributed 3D Graphics*, in Proc. ACM VRST'99, London, UK, December 1999, pp. 74–81.

[12] C. K. HESS, M. ROMÁN, and R. CAMPBELL, *Building Applications for Ubiquitous Computing Environments*, in International Conference on Pervasive Computing (Pervasive 2002), Zurich, Switzerland, August 2002, pp. 16–29.

[13] H. KATO and M. BILLINGHURST, *Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferenencing System*, in Proc. IWAR'99, San Francisco, CA, USA, October 21–22 1999, IEEE CS, pp. 85–94.

[14] T. KINDBERG, J. BARTON, J. MORGAN, G. BECKER, D. CASWELL, P. DEBATY, G. GOPAL, M. FRID, V. KRISHNAN, H. MORRIS, J. SCHETTINO, and B. SERRA, *people, places, things: web presence for the real world*, tech. rep., Hewlett-Packard Laboratories, 2003.

[15] G. KLINKER, A. DUTOIT, M. BAUER, J. BAYER, V. NOVAK, and D. MATZKE, *Fata Morgana – A Presentation System for Product Design*, in International Symposium on Aumgented and Mixed Reality – ISMAR 2002, 2002.

[16] B. MACINTYRE and S. FEINER, *Language-level support for exploratory programming of distributed virtual environments*, in Proc. ACM UIST 2002, Seattle, WA, November 6–8 1996, pp. 83–94.

[17] A. MACWILLIAMS, T. REICHER, and B. BRÜGGE, *Decentralized Coordination of Distributed Interdependent Services*, in IEEE Distributed Systems Online – Middleware Work in Progress Papers, Rio de Janeiro, Brazil, June 2003.

[18] A. MACWILLIAMS, C. SANDOR, M. WAGNER, M. BAUER, G. KLINKER, and B. BRUEGGE, *Herding Sheep: Live System Development for Distributed Augmented Reality*, in To appear in Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR), Tokyo, Japan, 2003.

[19] MICROSOFT CORPORATION, *.NET Homepage*. http://www.microsoft.com/net, 2003.

[20] MIT LABORATORY FOR COMPUTER SCIENCE, *MIT Project Oxygen: Overview*. http://oxygen.lcs.mit.edu/Overview.html, 2003.

[21] J. NEWMAN, D. INGRAM, and A. HOPPER, *Augmented Reality in a Wide Area Sentient Environment*, in 2nd Intnational Symposium on Augmented Reality (ISAR), New York, Oct. 2001, pp. 77–86.

[22] G. REITMAYR and D. SCHMALSTIEG, *OpenTracker – An Open Software Architecture for Reconfigurable Tracking based on XML*, in Proceedings of the ACM Symposium on Virtual Reality Software & Technology (VRST), Banff, Alberta, Canada, 2001, pp. 47–54.

[23] M. ROMÁN, C. K. HESS, R. CERQUEIRE, A. RANGANATHAN, R. CAMPBELL, and K. NAHRSTEDT, *Gaia: A Middleware Infrastructure to Enable Active Spaces*, IEEE Pervasive Computing, (Oct-Dec), 2002, pp. 74–83.

[24] C. SANDOR, A. MACWILLIAMS, M. WAGNER, M. BAUER, and G. KLINKER, *SHEEP: The Shared Environment Entertainment Pasture*, in Demonstration at the IEEE and ACM International Symposium on Mixed and Augmented Reality – ISMAR 2002, Darmstadt, Germany, 2002.

[25] D. SCHMALSTIEG, A. FUHRMANN, G. HESINA, Z. SZALAVARI, L. M. ENCARNAÇÃO, M. GERVAUTZ, and W. PURGATHOFER, *The Studierstube Augmented Reality Project*, Presence, 11(1), 2002.

[26] D. SCHMALSTIEG, G. REITMAYR, and G. HESINA, *Distributed Applications for Collaborative Three-dimensional Workspaces*, Presence, 12(1), 2003.

[27] P. STRAUSS and R. CAREY, *An Object-Oriented 3D Graphics Toolkit*, in Proceedings of Siggraph'92, 1992, pp. 341–349.

[28] SUN MICROSYSTEMS, *Sun Open Network Environment (ONE) Software Architecture*. http://www.sun.com/products/sunone/wp-arch/, 2001.

[29] 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, 2002.

[30] J. WERNECKE, *The Inventor Toolmaker: Extending OpenInventor, Release 2*, Addison Wesley, 1994.