***A component-oriented approach for Mixed Reality applications**, M. Haller, In Maribel Sanchez-Segura, editor, Developing Future Interactive Systems, Idea Publishing Group, 2004, http://www.idea-group.com/books/details.asp?id=4487*

[Book chapter publication]

A component-oriented approach for Mixed Reality applications

Michael Haller

Upper Austria University of Applied Sciences

Media Technology and Design

A-4232 Hagenberg

Hauptstrasse 117

Austria

Phone: +43 7236 3888 2127

Fax: +43 7236 3888 2199

Email: haller@fh-hagenberg.at

A COMPONENT-ORIENTED APPROACH FOR MIXED REALITY APPLICATIONS

ABSTRACT

This chapter introduces a component-oriented approach for developing Mixed Reality (MR) applications. After a short definition of Mixed Reality, we present two possible solutions for a component-oriented framework. Both solutions have been implemented in two different MR projects (SAVE and AMIRE). The first project, SAVE, is a Safety training system for Virtual Environments, whereas the goal of the AMIRE project is to develop different authoring tools for Mixed Reality applications. A component-oriented solution allows developers to implement better designed MR applications and it fosters the reusability of existing MR software solutions (often called MR gems). Finally, it supports the implementation of adequate visual authoring tools that help end users to develop their own MR applications with no programming skills.

INTRODUCTION

The often underestimated complexity of Mixed Reality applications necessitates efficient application design. Rapid prototyping of Mixed Reality applications is mostly impossible, because of two reasons: firstly, most of the existing frameworks are in many cases too complex to be extended and secondly, it needs a lot of software development skills and interface programming knowledge to develop good designed MR software. In this chapter we want to present a component-oriented approach for developing MR applications. The goal of this approach is to support developers during their development of new MR applications. Having a component-oriented

framework makes the programming life easy, because the developers do not have to reinvent everything from scratch. Based on this approach corresponding authoring tools will support end users to develop their own MR applications without having programming skills. Therefore, we introduce a component-oriented approach for the development of Mixed Reality applications. After a short overview in the taxonomy of Mixed Reality and Virtual Environments (described in section 1) we present the requirements for such applications. Moreover, in section 2 we describe the related work in this field. In section 3, we present a general component oriented approach, followed by two showcases described in section 4, where our approach has successfully been implemented. Both applications are based on the component oriented approach and result in a generic and flexible system. Finally, in section 6, we describe some future trends including the implementation of nice MR authoring tools for end users with no programming skills. We conclude this chapter with a short summary given in section 7.

BACKGROUND

In the following section we will give a short overview of Mixed Reality describing the requirements for the setup of an MR application and presenting the related work in this field.

Mixed Reality, from the Virtual Environment to the Real Environment

Mixed Reality (MR) is a particular superset of Virtual Reality (VR) technology that involves the merging of real and virtual worlds somewhere along the Reality-Virtuality Continuum, which connects completely real to completely virtual environments. The terminology was first introduced by Milgram (Milgram & Kishino, 1994) and is depicted in Figure 1. MR technology has been exploited in the medical, military and entertainment fields (Azuma, 1997); more and more new fields such as industry and training are becoming interested in its possibilities. The use of Mixed Reality enhances users' perception and the interaction with the real world (Azuma et al., 2001). Virtual objects display information that the users cannot directly detect with their senses. In addition, this information conveyed by the virtual objects helps a user perform real-world tasks.

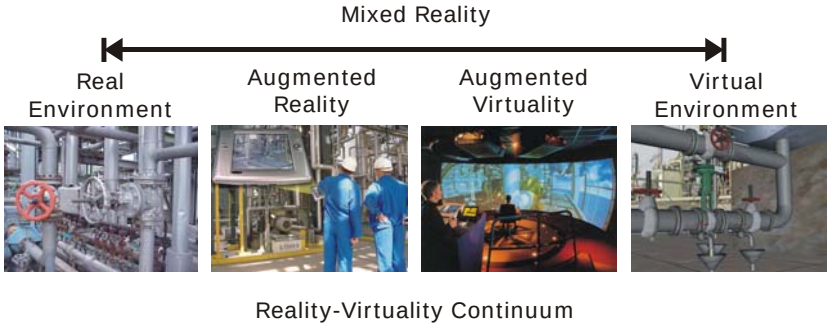


Figure 1: From the Real Environment to the Virtual Environment.

Virtual environments and Virtual Reality based applications can become very complex. Even more complex than present VR systems are the VR tools for modeling these environments (Bimber et al., 2001). Often, VR applications and the corresponding authoring tools are not easily extendible and the authors of VR environments require a lot of programming knowledge for realizing the desired virtual scene (Milgram & Kishino, 1994). On the one hand, developers should have programming skills, on the other hand they should have an exact technical knowledge about the composition of the scene. Due to the different hardware devices (input and output devices), the framework has to be extendable, open for new devices and finally it should be easy to use for programmers. Figure 2 depicts the most important components of a virtual environment: The user is the central part of the system, followed by the input and output devices that present the environment to the user and finally the simulation itself that renders the virtual environment.

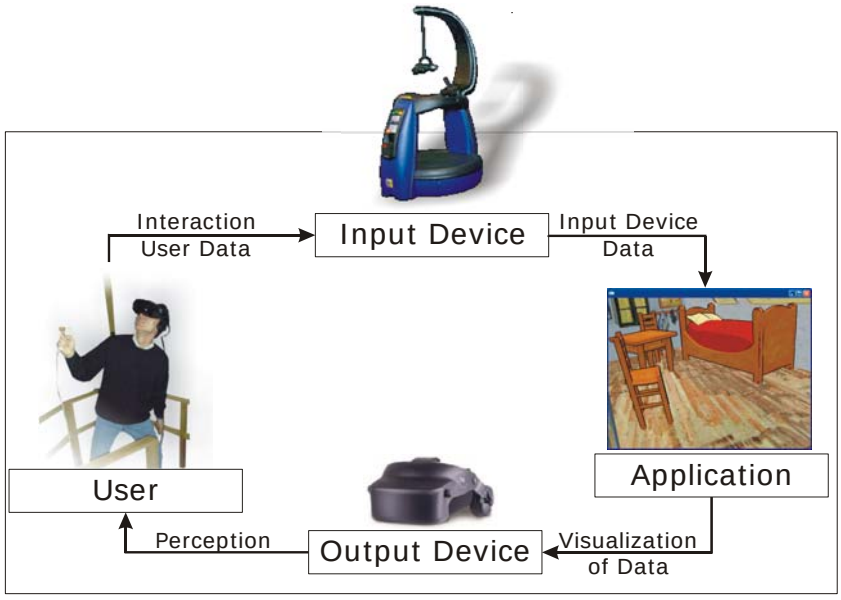


Figure 2: The most important components of a Virtual Environment application.

Normally, Virtual Reality systems work on powerful graphics machines (see Figure 3). The visual representation is mostly displayed to the user through an HMD device that allows an immersive feeling combined with a positional tracking device that reports the user's head and body position to the system. With this input, the simulation can generate a first-person view. Especially in VR simulation, external trainers often supervise the training session. In this case, an external application often communicates with the simulation (e.g. via TCP/IP or UDP).

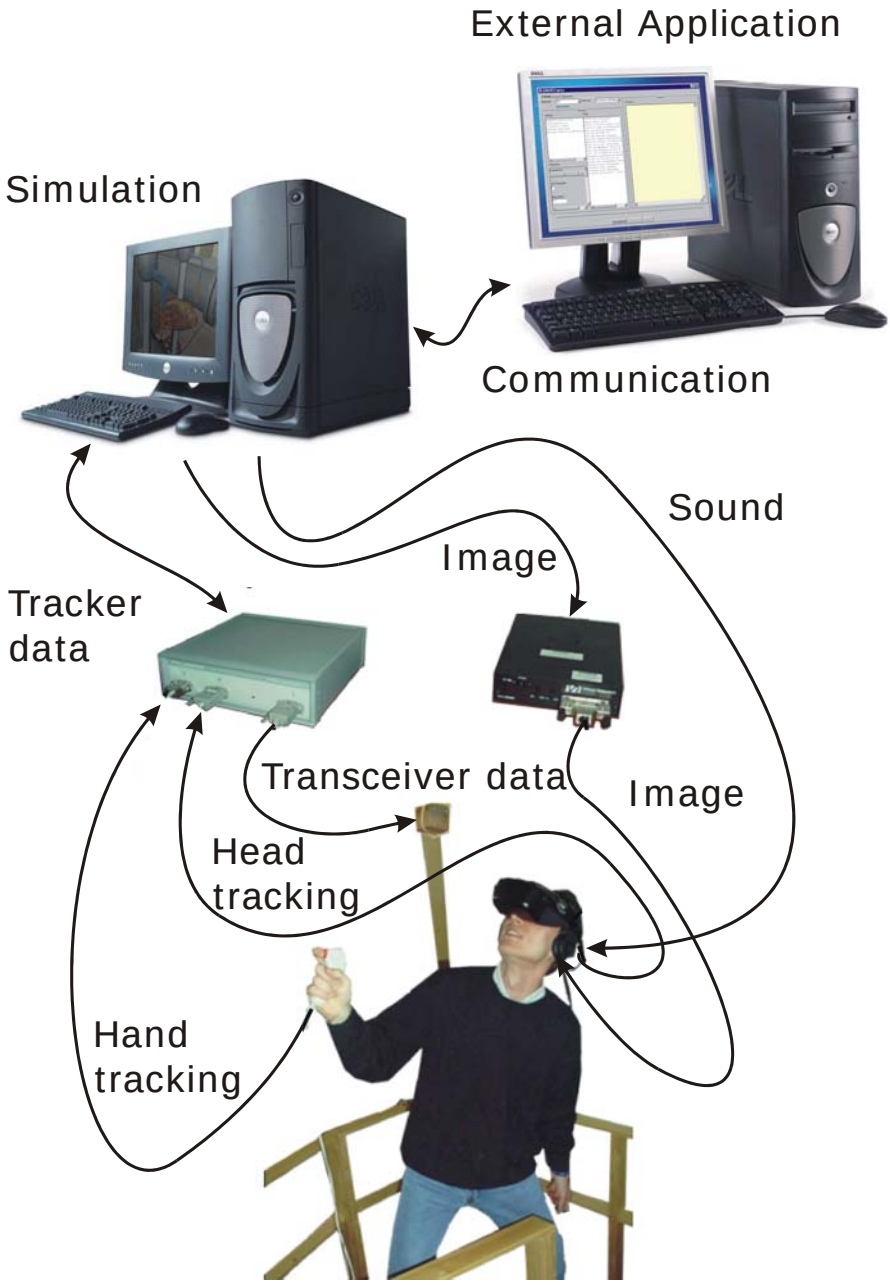


Figure 3: A typical topology of a Virtual Reality application.

Requirements for a Mixed Reality Component Architecture

The component-oriented approach for the implementation of Mixed Reality applications is characterized mainly by the following facts:

- It should allow 3rd party development,
- The framework should hide programming code,
- Components can be implemented by using other components,

- In most cases, the component oriented approach offers authoring tools that allow for an easy development of new components.

As mentioned in (Dachselt, 2001), we have to distinguish between technical and authoring requirements. From the technical point of view, the component-oriented approach should provide:

- **Portability:** Independently from an underlying renderer, each component should be removable and by doing so, it should not influence the system in a negative way.
- **Distribution:** Components should not only work on one system. A distributed application should also be supported.
- **Adaptation:** Each component can be modified by the user's preferences. This should be possible with minimal effort.
- **Performance:** It is one of the key factors for a Mixed Reality application to work in real-time. Long delays between user inputs and the system output cause discomfort or simulator sickness and it negatively affects the user's feeling of presence.

From the authoring point of view, there are the following additional requirements:

- A **clear interface** for the components. This includes the data and the component definition.
- Well written **documentation** and **description** of the components.
- Support of adequate **authoring tools** that allow a rapid prototyping for authoring users who do not have programming backgrounds.
- **Easy configuration** of the properties for a component.
- **Persistence** of components.

- Loading of **additional components** without a modification of the underlying framework.

Dörner and Grimm define in (Dörner & Grimm, 2001) the following component features:

- **Customization**: Each component needs to be able to be modified by an author. Not only programmers should be able to customize components, but also authors-, and experts by using authoring tools.
- **Persistence**: Components and their state should be stored in a way that they can be reloaded. This should even include the distribution of components over a network.
- **Reflection**: Functionality that allows information to be retrieved from a component, such as the events that are sent to another component.
- **Event-based communication**: Components have to communicate with others – in most cases this is an event-driven communication in which e messages are sent from a so called event source to all entities that are registered to listen to the particular events. The communication messages are mostly forwarded by so called slots, or pins.

Related Mixed Reality Frameworks

Many European projects mainly focus on the development of MR applications for a special domain (e.g. technical maintenance), such as ARVIKA (Friedrich, 2000), Studierstube (Schmalstieg et al., 1996), and DWARF (Bauer et al., 2001). Unfortunately, only MR experts are able to develop MR/AR applications. Prototyping of MR based applications becomes a very difficult task, because most research institutes have to develop these applications starting from scratch.

With the use of the ARToolKit library (Kato et al., 1999), the development of MR/AR applications became easier and more popular. Figure 4 shows an AR example based on ARToolKit, in which the users can place 3D sound sources into the real world (Haller et al., 2002), (Dobler et al., 2002). The ARToolKit library is free for non-commercial projects and easy to integrate into an OpenGL environment. In addition, it offers the possibility to recognize objects without the usage of high-end tracking systems. Most of current MR applications are isolated and each institute implements its own framework. The situation is similar to the first steps in game development, where everyone was programming his/her own game-engine. The development of Mixed Reality software is like game development: in general it needs to be very flexible, because things change and new requirements and features need to be added to stay competitive. A source that can easily be adapted to change is a must and worthy goal.

Figure 4: In the ASR (Augmented Sound Reality) the user is able to place virtual 3D sound sources into the real world.

Most of current AR/MR applications are based on a self-made framework or/and at least on a scenegraph library (e.g. Open Inventor, OpenGL Performer, Open SceneGraph, or OpenSG). Of course, most of the AR/MR frameworks like Studierstube (Schmalstieg et al., 1996), DWARF (Bauer et al., 2001), and Tinmith-evo5 (Piekarski & Thomas, 2003) are object-oriented frameworks, but not all of them have a component-oriented approach or there is a lack of corresponding visual authoring tools (e.g. Studierstube (Studierstube, 2003) or DWARF (Dwarf, 2003)). Scripting support as presented in APRIL would help experienced end users, but it seems to be too complex for end users with absolutely no programming skills (April, 2003).

COMPONENT-ORIENTED DESIGN – OVERVIEW

The following section presents a component-oriented design for developing MR applications, describing the main features of the design approach. Moreover, it depicts the component-oriented design workflow including a short overview of the different rules.

The component-oriented design for a MR application

Geiger defines in (Geiger et al., 2000) a component as a separated entity with a specific size. It is characterized by dependencies and the framework permits a dynamic loading of components. Components can be either big or small, but they have to be of a clear structure. In our sense, a component-oriented MR system should be comparable to a set of different small LEGO^{TM} components, which can be connected. In our view, a visible component has its geometry and a property. Moreover, it is characterized by a behavior. Components can be very simple, but they can also be composed of other simple components. They are often called composed components or compound components. Components hide their internal operations and programmers do not need to understand the internal complexity.

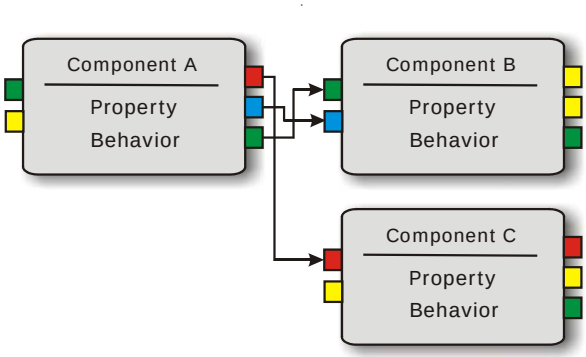


Figure 5: Components can be connected with in-slots and out-slots. These slots have to be of the same type.

Each component is composed of in- and out-slots, which can either send data to another component or receive data from the previous component. Consequently, out-slots can be connected to in-slots. This concept is illustrated in Figure 5. We use typing for these slots to specify the receiver and the emitter data. By comparing the type of in- and out-slots we can decide which slots are compatible for connection. Using slots for inter-object communication is not new in a component-oriented approach. Our communication concept is based on prototypes for components. This means the component developer registers the component via a component manager.

By routing events from out-slots to in-slots of another node, customized functions and dependencies can be implemented, e.g. if a switch has been switched on, a lamp shines, etc. In the SAVE-system, all the components are described by prototype nodes based on VRML 97 with their in-slots, out-slots, and parameters, which allow for a closer description of the object (Haller et al., 2001) (Haller, 2002).

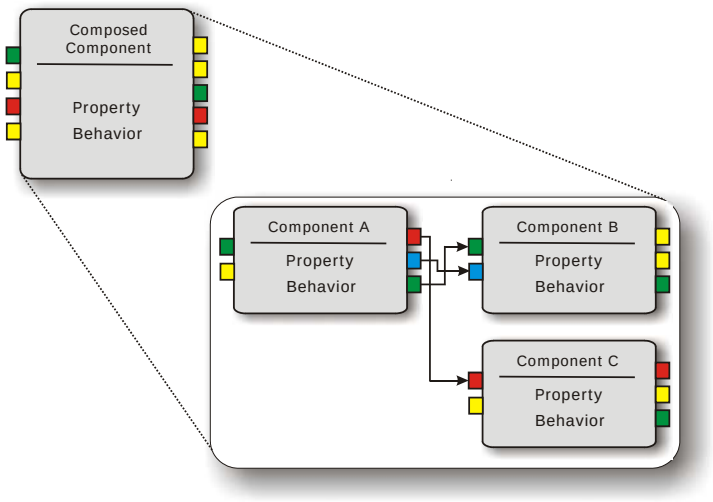


Figure 6: Different components can be grouped into composed components. This approach is important for authoring components. Otherwise, the visualization can be too disorganized and unmanageable.

Of course, components can be composed of more simple components with fewer properties. In Figure 6 we have an example in which not all slots of the components A, B, and C are routed to the outside of the composed component – the composed component has fewer slots. Once a component has generated an event, the event is propagated from out-slot along any route to other nodes. Event notifications are propagated from sources to listeners by the corresponding method invocations on the target listener objects. Data enters via the first component, passes along to the second component, and so on. The architecture has an event-driven design and components only start information processing after receiving an event. After processing, the components can generate new events and the processing can be done in parallel. The system is based on the producer/consumer concept. No component (consumer) starts sending messages and processing information without having received a message from other nodes (producer). The only components, which generate messages, are called active components (e.g. timer, clock). They are triggered by system calls – but they do not start sending messages themselves.

There is no doubt that appropriate authoring tools would greatly assist users. Therefore, visual based authoring tools are closely connected to the component oriented approach. Normally, end users get an authoring tool with a set of pre-defined small components. These components can be modified and end user can tweak their properties. If the authors are more experienced, they can build their own components by modifying the WRL/XML files; indeed, programmers can add more features by programming new components. The structure of the component is described in a WRL/XML file that allows end

users with no programming skills to tweak their components according to their requirements.

The component-oriented workflow

The component-oriented workflow is depicted in Figure 7, which shows the three different categories of persons.

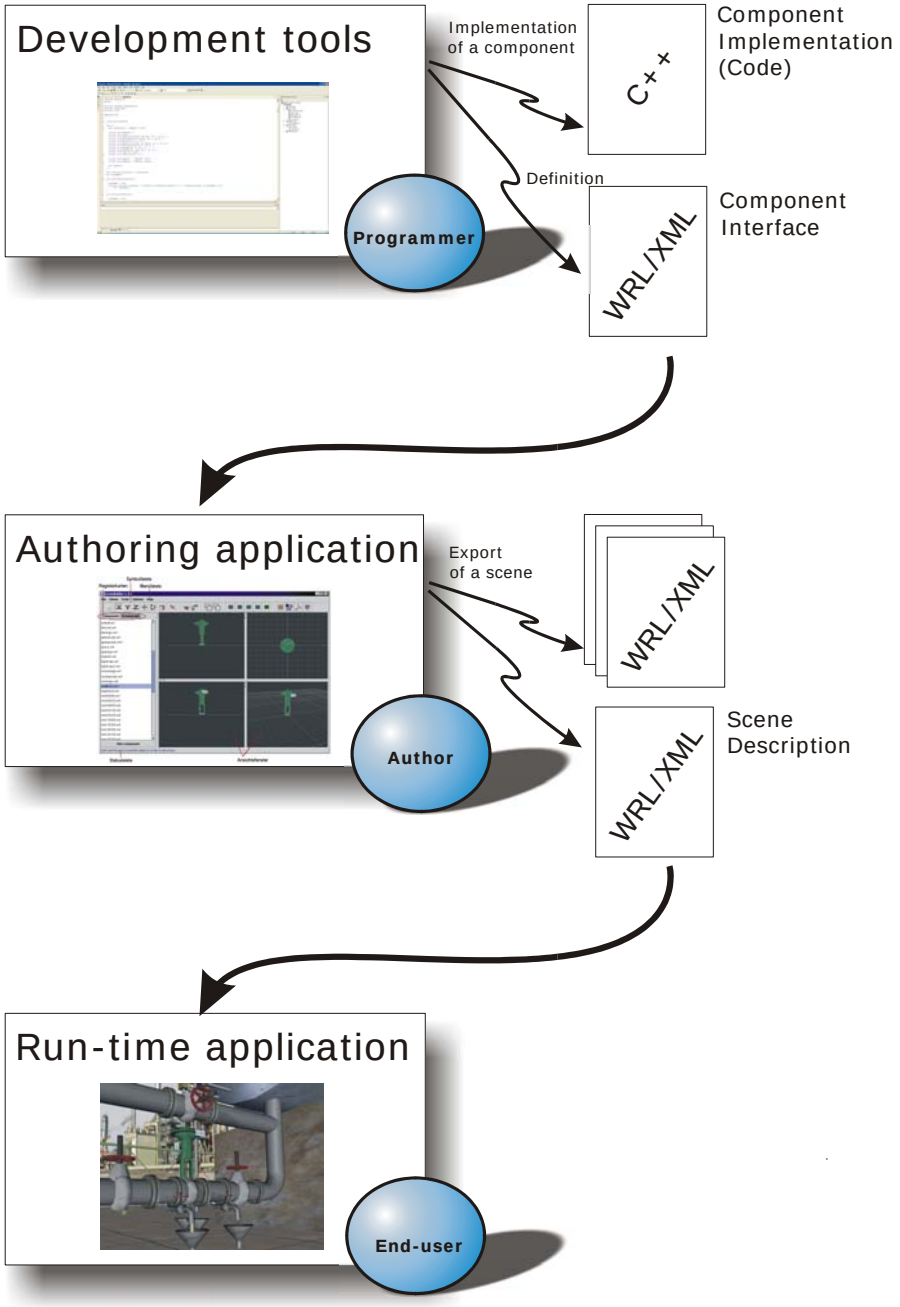


Figure 7: The component-oriented workflow.

The **programmers** are responsible for the creation, implementation and definition of components. In this case the programmers are using a conventional programming environment. The component interface can be described in a XML-form. In SAVE we used so called PROTO's for the interface description. In fact, these PROTO's have been based on the VRML prototypes (Web3D, 2003). XML has become more popular as of late because of two reasons: firstly, there are a number of open source XML-parsers that help developers to parse the XML-files. Secondly, the XML-structure is flexible, extendable, and easy to use.

The **authors** are using so called authoring tools for the implementation of the application. The underlying framework is responsible for the creation of components and the connection between them. Normally, authors are not programming experts – they are experts in their field and normally they know what the end users want to have. But in the normal case they don't know how to implement due to their lack of programming skills. The results of the author are the scenarios. They include the graphical elements, the objects of a Mixed Reality scenario, the property settings, the behavior of the components, and finally the logical connection between the components.

Finally, the **end users** are the persons that are involved in the virtual environment. They are not concerned with programming, modeling, and authoring.

COMPONENT-ORIENTED DESIGN – TWO SHOWCASES

The following two showcases (SAVE and AMIRE) are based on the component-oriented framework. First, we present the SAVE system followed by a closer description of its component-oriented architecture. Second. We demonstrate the AMIRE project including a presentation of the component-oriented approach.

SAVE (Safety Virtual Environment)

In 1997, we started implementing a VR-based training simulation for an oil refinery. The first prototypes of SAVE were quite simple and showed only a small part of the refinery (Save, 2003). In 1999, we had to add new training scenarios and new features and functionality (cf. Figure 8 and Figure 9). What we wanted to have was a framework in which anyone could develop a virtual reality based application with a high degree of complexity. Moreover, we wanted to have an application design with high reusability, for which parts could be reused in different VR training scenarios. We knew that effective reuse requires more than just reusable code and libraries. Our vision was a transition from library-based reuse to kit-based reuse and we wanted to move away from the traditional development of VR environments, for which the users require programming skills. The component-oriented approach with the included authoring tools show how modeling rather than programming can be used to realize virtual environments.

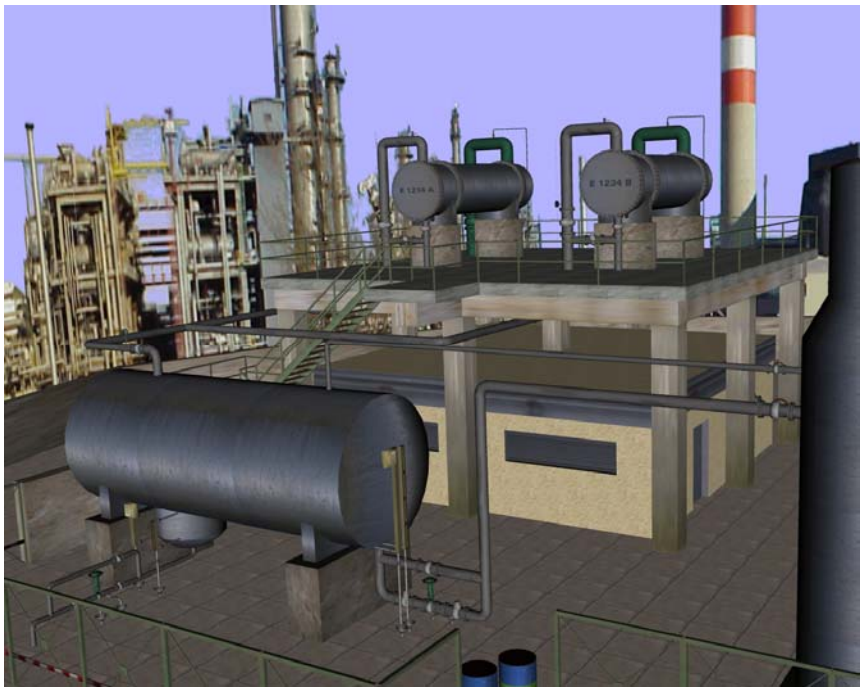


Figure 8: SAVE is a VR-based training program for oil refinery employees.

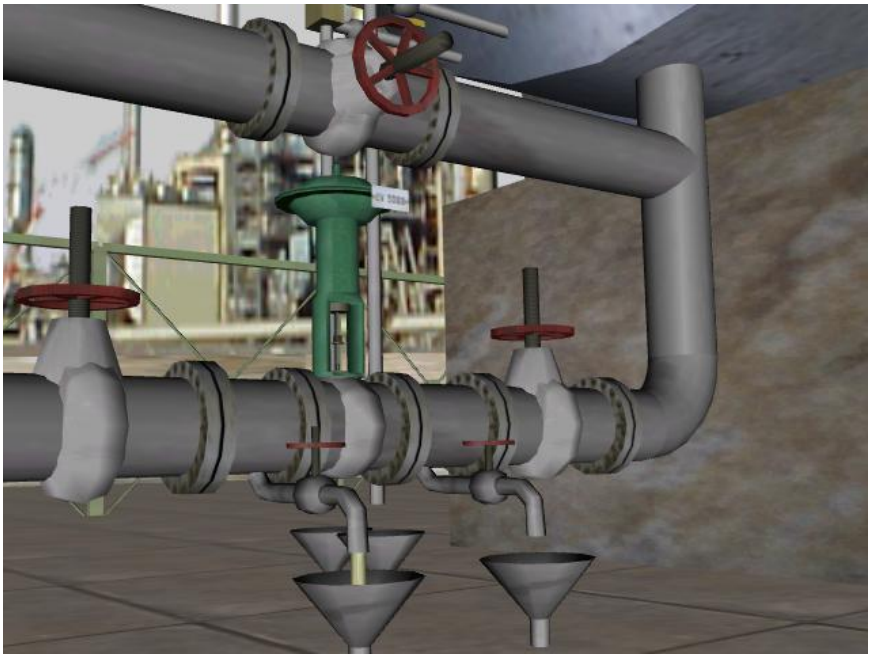


Figure 9: A typical scenario that has to be built-in for the virtual plant simulation. In fact, the dependencies between the components and the animation of the components itself are not so complex.

The component oriented design of SAVE

In SAVE, we wanted to have a component-oriented design that is not comparable to the commercial component models such as JavaBeans or COM. Our system is comparable to a set of different small LEGO™ components, which can be connected. In our view, a visible component has its geometry and its properties. Moreover, it is characterized by a behavior. Our goal was to offer a repository with different simple and even complex objects. All the components are described by prototype nodes based on VRML 97, describing their in-slots, out-slots, and parameters, which provide a more detailed description of the component (cf. Figure 10).

```
PROTO VRExample [
  eventIn  SFBool inputValue
  eventOut SFBool outputValue
  field MFNode children[]
] {}
```

Figure 10: An example of a simple component interface described in VRML 97.

Figure 11 depicts two components A and B and the interface described in VRML 97. Notice that the figure only describes the interface.

```
PROTO A [
  field ...
  eventIn SFBool outSlot
] {}

PROTO B [
  field ...
  eventOut SFBool inSlot
] {}
```

Figure 11: The interface of two components with a boolean slot.

The concrete components with the corresponding data have to be described separately (cf. Figure 12).

```
DEF a A {
  field ...
  ....
  }

DEF b B {
  field ...
  ....
  }

ROUTE a.outSlot TO b.inSlot
```

Figure 12: Two components with the routing statement.

The programmer's view is depicted in Figure 13, where the two components that are connected via slots are implemented. In our example, a b**oolean-**

value will be sent through the communication network. The message-mechanism is implemented in the base class of **A** and **B**, namely **VRNode**, in which the base methods for the communication are implemented. The **main()**-function creates two components. Next, the components **aComp** and **bComp** and their slots (in this case the **boolean**-slot) are connected together by calling the **AddListener()**-method. In our example, the program calls the **Set()**-method of the component **aComp** that results a change of the corresponding **boolean** value of the out-slot. Next, all listeners are notified (by calling the **NotifiyListener()**-method. Consequently, component **bComp** gets the corresponding **boolean-**value in the **ValueChanged()**-method and finally it prints the value on screen.

To be independent of the component **aComp,** an underlying graphic library, we encapsulate direct calls to the actual graphics library in certain graphical classes. Instead of creating the scenegraph using library specific calls, often a so called meta scenegraph will be built.

```
/* The base class VRNode implements all methods for the
 * registration and removal of the slots
 */
class A : public VRNode {
  public:
    int outSlotIdx;

    A() {
      outSlotIdx = AddOutput(&typeid(VRBool),"outSlot");
    }

    void Set(bool b) {
      SetBoolOutputValue(outSlotIdx, b);  // Create 'VRValue'
      NotifyListeners(outSlotIdx);        // Notify listeners about
    }                                     // new value
};

class B : public VRNode {
  public:
    int inSlotIdx;
    B() {
      inSlotIdx = AddInput(&typeid(VRBool), "inSlot");
    }

    // ValueChanged is called whenever the out-slot emits a value to
    // the in-slot.
    void ValueChanged(int in,
                      int inSlotHandle,
                      const VRValue* value) {

       // Which slot?
       if (inSlot == in) {
         VRBool* v = (VRBool*)value;  // Get the Boolean value
```

```
          cout << v->GetBoolValue() << "\n";
       }
    }
};

int main(int argc, char** argv) {
    A* aComp = new A();
    B* bComp = new B();

    // Connect component aComp with component bComp
    aComp->AddListener("outSlot", bComp, "inSlot");

    // Change the value of aComp. Consequently change the value of
    // component bComp
    aComp->Set(false);

    return 0;
}
```

Figure 13: The programmer's view of the SAVE component-oriented approach.

Figure 14 shows the three graphs of a simple example. In this case, a clickableButton consists of two separate geometries: one of the up-state (not pushed) and one for the down-state (pushed). If the user clicks the button, it remains in down-state until the user clicks it again.

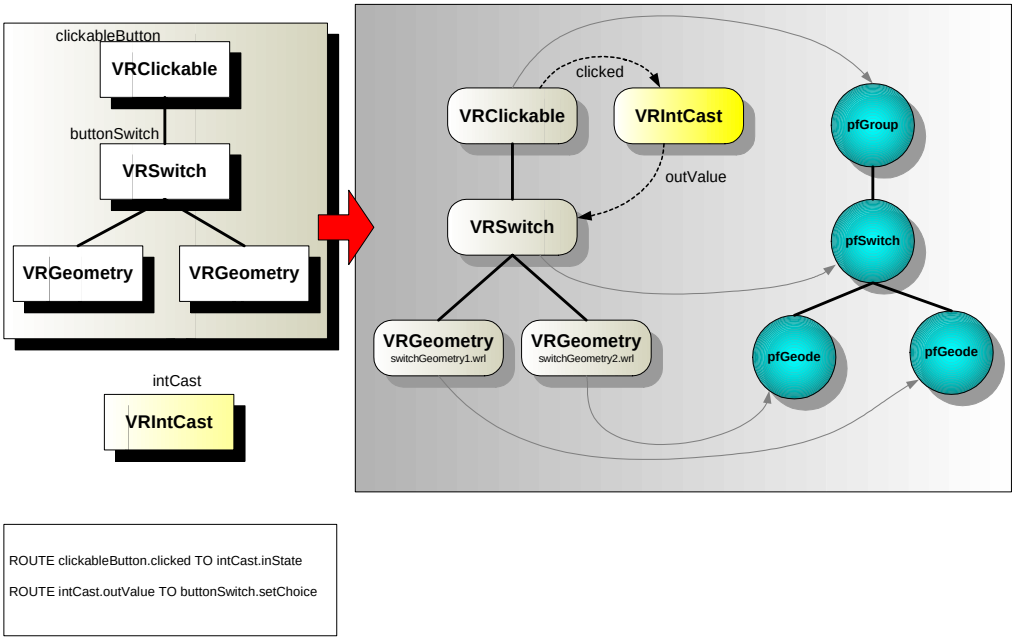


Figure 14: From the description to the scenegraph.

The leftmost graph structure presents the VRML 97 data structure generated by the SAVE parser. Note the thick black lines connecting the nodes. These lines represent parent children relationships. The graph structure in the middle of Figure 14 constitutes the meta scenegraph. Again, the thick black lines represent the parent children relationships. This structure is derived from the VRML 97 scenegraph. The right data structure is the scene graph of the graphics library which contains group-, transformation- and geometry-nodes. Note that the component communication network is not part of this scene graph.

Building the meta scenegraph does not require any knowledge of the underlying graphics library. The graphical classes can easily be replaced by classes which encapsulate calls to an alternative graphics library and no further changes in the source are required to add new graphical objects in the virtual environment. The mapping of the VRML source to the actual scene graph is performed in two steps. Firstly, our parser identifies all VRML nodes and

creates a corresponding VRML node representation. Secondly, the VRML node is traversed recursively and each node is converted to its corresponding meta scenegraph node representation. The result, in fact, is a meta scenegraph. Finally, after a second traversal, the component communication network is established.

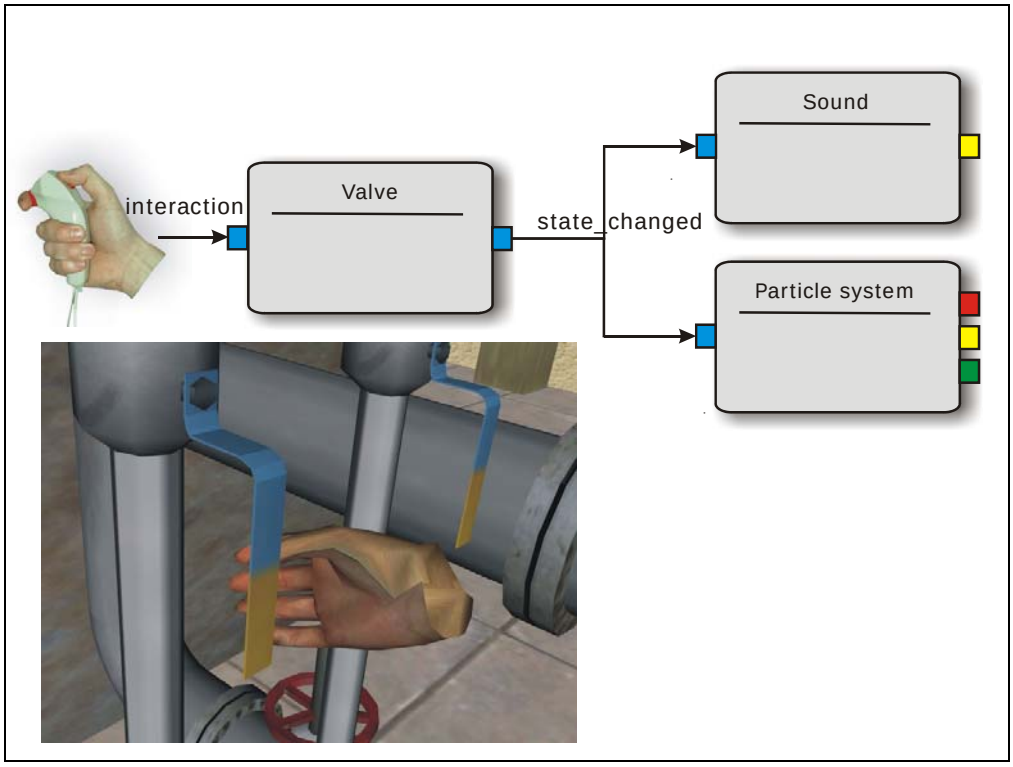


Figure 15: The user interacts with the virtual valve that changes its state and forwards its state_changed messages to the sound and a particle system component.

Figure 15 depicts an example of components that are connected together. The user holds a joystick with an integrated tracking receiver that tracks both, the position and the orientation of the user's hand. Accordingly, the virtual hand can be moved. While focusing the valve and after clicking to the interaction button of the joystick, the message will be propagated through the communication network (see Figure 15). In our example the valve sends its

message to a sound component that plays a sound file; the other listener is a particle system that simulates the outcoming water.

AMIRE (Authoring Mixed Reality)

In the AMIRE project the main goal is to provide Mixed Reality to other professionals than programmers and to facilitate efficient creation and modification of Mixed Reality applications (Amire, 2003). With the Mixed Reality authoring tool, people with lesser programming skills should be able to develop MR applications cost effectively with fewer resources and in reduced time. The AMIRE project is an EU-funded project (IST-2001-34024). The AMIRE authoring tool is based on two main principles: the use of user-centered design approach and open source code. Different authors have been involved all along the development process and they have affected the requirements as well as the user interface design. The tool is developed as open source and offered to all mixed reality developers. The project aims at more widespread use of Mixed Reality in different applications domains. The AMIRE team wants to promote the use of Mixed Reality in new application fields by more heterogeneous developers. With the high reuse of the MR content and easily maintainable, structured component libraries, the development times decrease and rapid prototyping of MR applications is possible.

Figure 16: The main goal of AMIRE is an authoring tool based on a component-oriented approach to achieve MR based applications. One of these applications is a refinery training program, where employees get more detailed information using a Tablet-PC.

The component-oriented design of AMIRE

AMIRE wants to adopt existing solutions and provide efficient means to encapsulate different solutions (called gems) in a uniform way (called components). Similar to existing gems collections (e.g. game programming gems) AMIRE offers an MR gem collection containing efficient solutions to individual Mixed Reality problems. A gem could be an object recognition library, a library for the graphical user interface, a tracking library or simply a 3D model loader. The gem collection is integrated into the AMIRE framework. Typically, MR gems can be reused in many different applications. For example, a “work path animation”-gem that visualizes the workflow of a

special machine in an oil refinery can be reused to explain painting techniques of a famous painting in a museum.

The MR gems in turn can be used to build application specific MR components (e.g. a navigation component for the museum application) as well as an MR framework that defines how MR components can communicate with each other and can be integrated into an application. The MR framework provides the required infrastructure. MR components represent solutions for particular domain specific problems (e.g. MR-based museum application) and they typically combine and extend MR gems towards advanced high-level features of a MR application. MR components feature a unified interface that allows easy configuration.

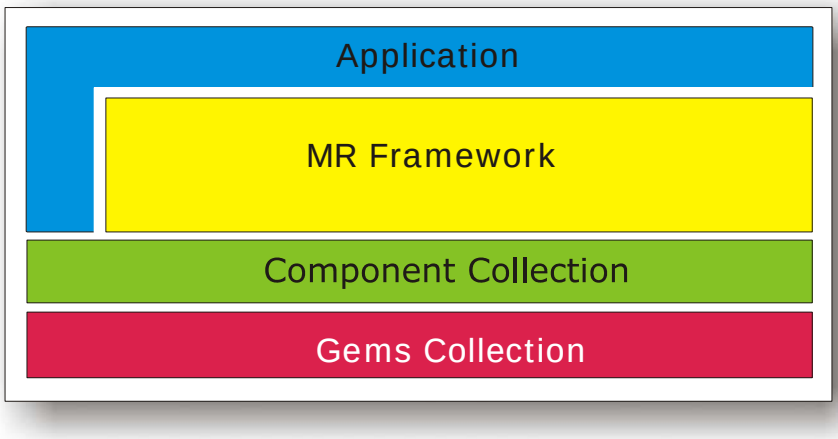


Figure 17: The different layers of AMIRE.

Figure 17 shows the different layers of AMIRE starting from the gem layer including the libraries and C++ solutions. The second layer is the component layer that defines the structure and the interface of the components that are used in the application. The MR framework is the glue of the AMIRE project, because it integrates both the MR gems and the MR components. A detailed

description of the AMIRE framework can be found in (Haller et al., 2003) and (Zauner et al., 2003). Some of the most important features are:

- A generic configuration mechanism of components by so-called properties.
- The communication between components is based on in- and out-slots.
- The framework provides base conventions for 2D and 3D components. Only with these conventions is it possible to build a generic authoring tool which uses MR for the authoring process.
- A prototype-oriented approach is based on two kinds of component prototypes, basic components and composed components.
- The authoring tool supports the dynamic loading of C++ and XML based libraries at runtime.
- An MR application can be saved and reloaded. The file format is XML.
- The integration of an object recognition unit (cf. ARToolKit).

Figure 18 depicts the two process approach of AMIRE, starting from the authoring tool to the run-time application. Both applications are based on the same AMIRE framework and on the same component repository. Of course, the authoring tool includes more components that do not necessarily have to be integrated into the run-time application. Due to the abstraction via components an exchange of the underlying gems can be guaranteed: whenever the technology changes, programmers only have to include the new DLL, but the code for the run-time application doesn’t have to be changed again. This is, of course, a great advantage, especially in a field in which new hardware and new base technologies come out every year.

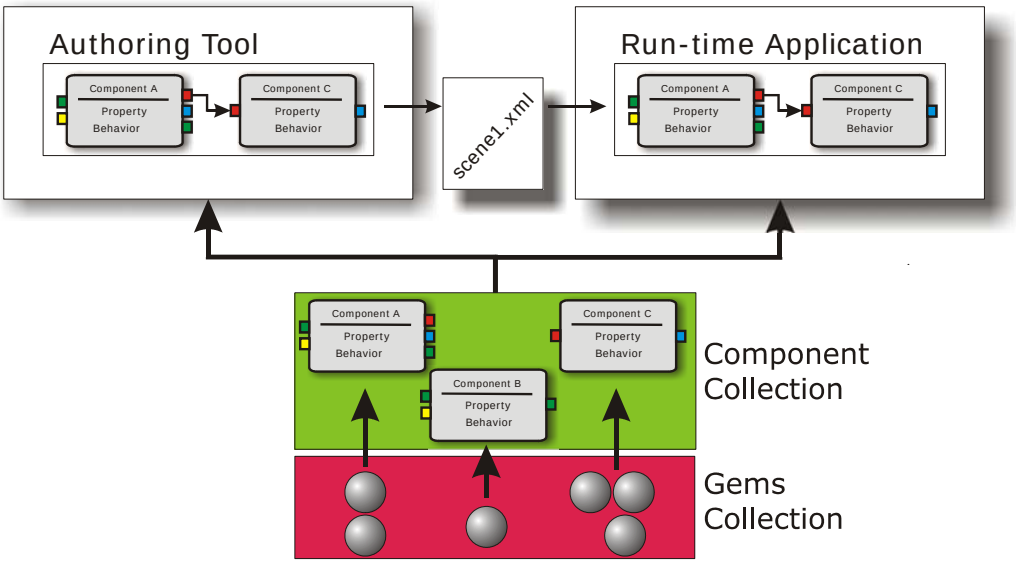
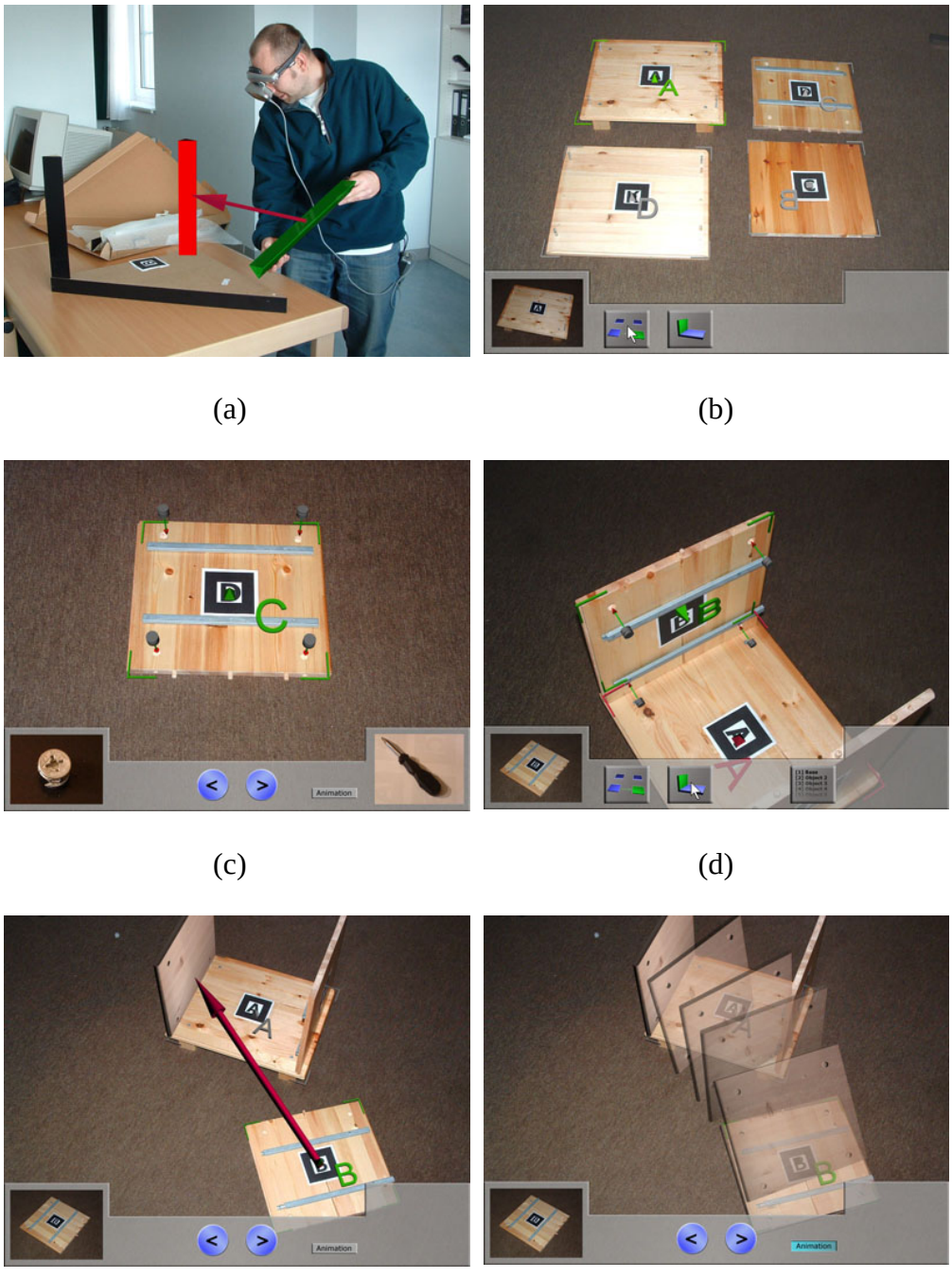


Figure 18: From the authoring tool to the run-time application.

In contrast to SAVE, AMIRE uses XML for the description of the components interface and for the description of a scenario. A prototype-oriented approach is used to create new component instances. This means that AMIRE has prototypes of specific components and an interface to make clones of each prototype. Two kinds of component prototypes are available. The first one is a native kind, completely written in C++ and packed into dynamic libraries such as the DLL format of Microsoft Windows systems or the shared object format of UNIX systems. The second kind is based on the existing set of prototypes and is called a composed component prototype. It consists of a component network, an export list of slots and a configuration export for the components in the network. A composed component prototype is handled like a native prototype. Hence the author does not see any difference between using instances of a native and a composed component prototype. Authoring an application without generating and compiling additional C++ sources requires the dynamic loading of libraries. The AMIRE framework provides an interface to load and to replace a library at runtime. Currently, two kinds of libraries are

supported, namely C++ based libraries and XML based libraries of composed components. As mentioned before, the persistence of an application is supported by an XML based file format. Such an XML file contains a list of library dependencies, the component instances and the connections between them. Furthermore, an XML format for libraries of composed component prototypes is defined.

In addition to the refinery training application, we developed another application, a furniture assembly instructor program (called FaiMR) that is based on the AMIRE framework: printed instruction manuals for furniture assembly often have one disadvantage in common: it takes a lot of time to make sense of their meaning since they show several steps of assembly together in a few pictures. Furthermore, it is hard to find the connections between the instructions printed in 2D and the real parts. The idea of this work is to connect the instruction directly to the parts of a piece of furniture. To do this, Mixed Reality is used, which combines reality (recorded by a webcam) with additional information using common computer graphics in 2D and 3D which are overlayed. A closer description of this application can be found in (Brandl A., 2003) and in (Zauner et al., 2003).



(a) (b)

(c) (d)

(e) (f)

Figure 19: Some snapshots of the FaiMR program, which is based on the AMIRE framework. Figure (a) shows a general overview of the application. Figures (b), (c), and (d) show the view of the furniture expert who can assemble the furniture. The assembly scenario file is stored in an

XML file. The end user has the view as depicted in (e) and (f), where he/she has a new furniture part. The arrow and the animation show how the furniture has to be assembled.

FUTURE TRENDS

Even if the field of Mixed Reality is rather new and even if there are a lot of unresolved problems, such as in tracking and hardware devices, fast development for prototyping of MR is becoming more and more important. Newcomers – even non programmers – should be involved in this fascinating world; because of their artistic and usability knowledge, their constructive input for new ideas would be very fruitful for new MR applications. But actually, especially these persons are hindered to get involved, because of the lack of adequate authoring tools. Performance is of course a very important factor in a real-time application like MR applications.

The component-oriented design itself should be based on different well established solutions (libraries) – this was, for example the main goal of AMIRE. In the optimal case, there are already good solutions for different problems; The Virtual Reality Peripheral Network library VRPN is a good example for tracking (VRPN, 2003). It is a well designed library that supports a lot of different trackers. The same trend can be found in the game industry: Ten years ago, every game company implemented its own game engine, because the projects were smaller and the teams consisted of fewer than 12 people. But now, projects are becoming more and more complex and not everything can be implemented starting from scratch.

Combined with a component-oriented approach is the use of adequate authoring tools for end users who should be able to develop their own MR applications with lesser or no programming skills. In the rare cases end users have programming skills, they often lack scripting or 3D modeling skills. But they are specialists in their field and they know what they would like to have

but they cannot experiment with the new media. Therefore, the usability and the graphical user interface of these tools have to be as simple as possible.

The need for more authoring tools

Nowadays, in the computer game industry every big game is implemented by using corresponding authoring tools (cf. GTk-Radian depicted in Figure 20). There are already different well-defined roles in the production process of a game – from the programmers who develop the engine through the game developers to the game designers and level designers. And the game developers would never touch the engine – they are concerned with the game itself and they build the application with the corresponding authoring tools that are offered together with the game engine. The quality of these tools is so good that even teenagers are able to develop great new levels for their own games.

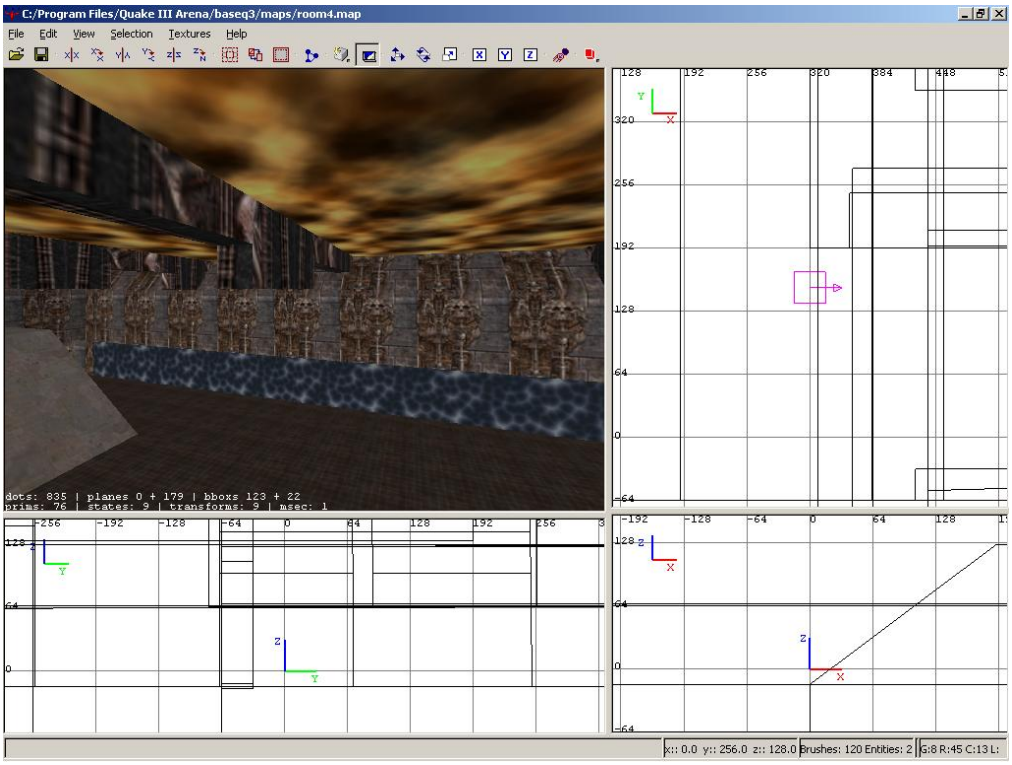


Figure 20: GTk-Radiant is one of the most used level design tools for Quake.

What we need is more effort in the development of good authoring tools for the creation of VR/AR/MR applications. Currently, the assortment of such tools is not very large. Figure 21 and Figure 22 depict two commercial authoring tools designed for VR/MR applications. Both tools are designed for designers and expert users in the sense that they should know about the basics of 3D. Especially Virtools (Virtools, 2003) offers a huge amount of components that can be connected together to build powerful applications that are used in different VR hardware devices (also for CAVE applications).

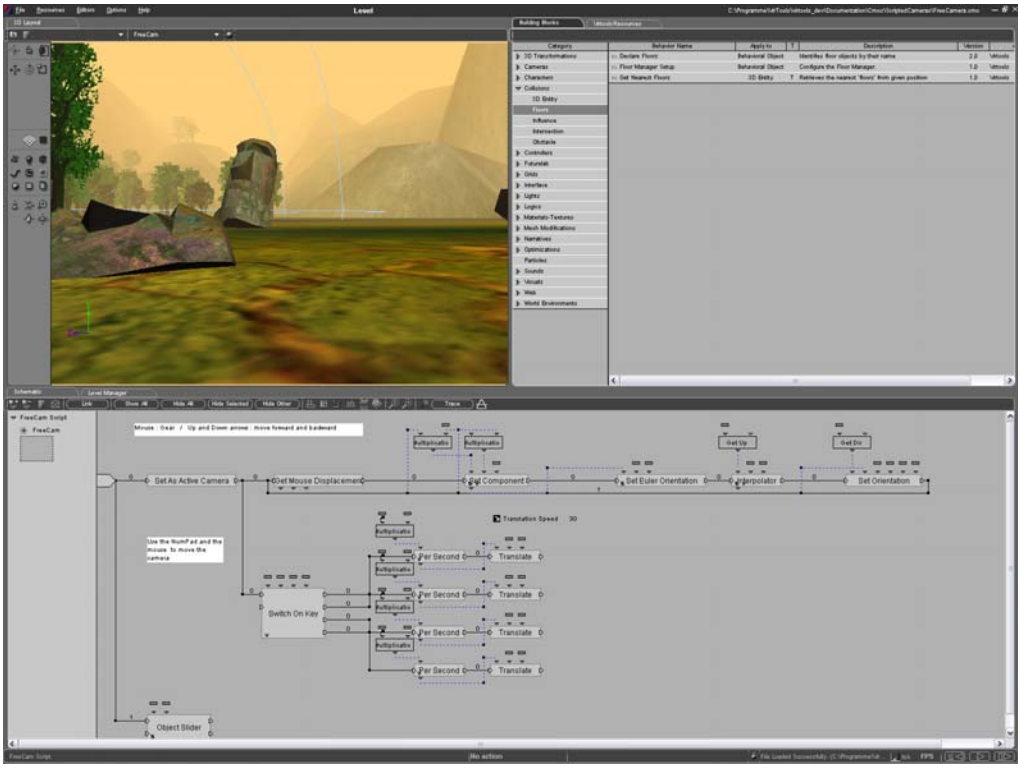


Figure 21: The authoring tool Virtools is frequently used for VR applications (Virtools, 2003).

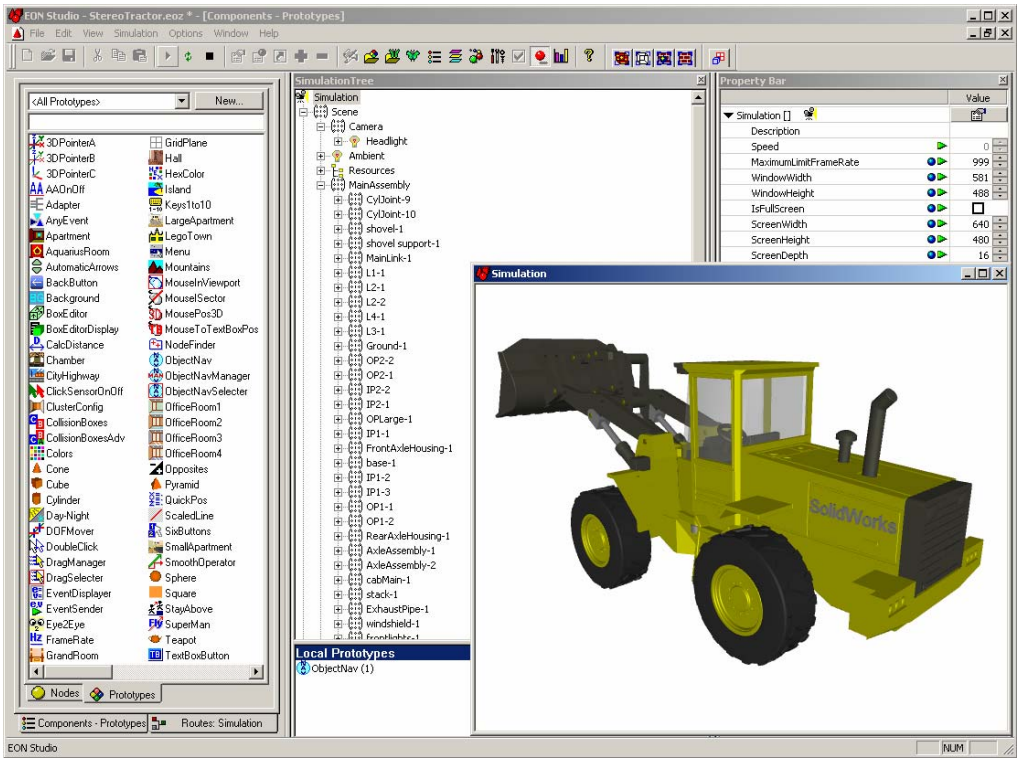


Figure 22: EON Studio uses a component-oriented approach that allows experts to implement their VR applications in a rapid way (EON Reality, 2003).

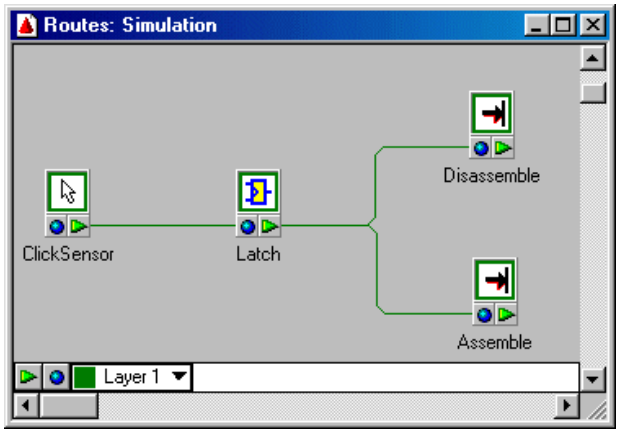


Figure 23: The routing in EON Studio is simple (EON Reality, 2003).

An authoring tool for developing an MR application can be divided into the following sub-authoring tools:

- **Placement tool**: This tool allows the user to place the virtual objects into the virtual world.
- **Configuration tool**: The properties of components are always different from each other and should have to be modified accordingly. A simple user interface with the possibility of a flexible change of these properties should be a key issue of this tool.
- **Connection tool**: Finally, components have to be connected together. With our slot paradigm, the optimal graphical user interface is to drag and drop the slots and a visual line should show the connections. One of the biggest problems in this tool is the visualization method of the connections. In our opinion, all components should be visualized. This means that the graphical user interface can become very, very complex – but it does not hide information from the user. Different abstraction layers would help a lot – because not everyone is interested in all the possibilities.

Currently, in the field of AR, we have no commercial tool available and only prototypes developed in research projects (e.g. in ARVIKA, DART, STAR, AMIRE); The DART project is built as a collection of extensions to the Macromedia Director multimedia-programming environment, and therefore primarily developed for designers who want to develop their own AR application (MacIntyre, 2003). Another approach is postulated by the AMIRE project, in which the authoring tools are based on a component-oriented approach but not on an existing multimedia authoring tool like Director. The AMIRE goal is to take the real environment and to place the objects there. Figure 24 depicts how the authoring in a MR environment could look –

especially the placement of the virtual objects into the world could be realized in a very intuitive way using the real environment. The setting of the properties for the component and the routing of the components is done in 2D (cf. Figure 25).

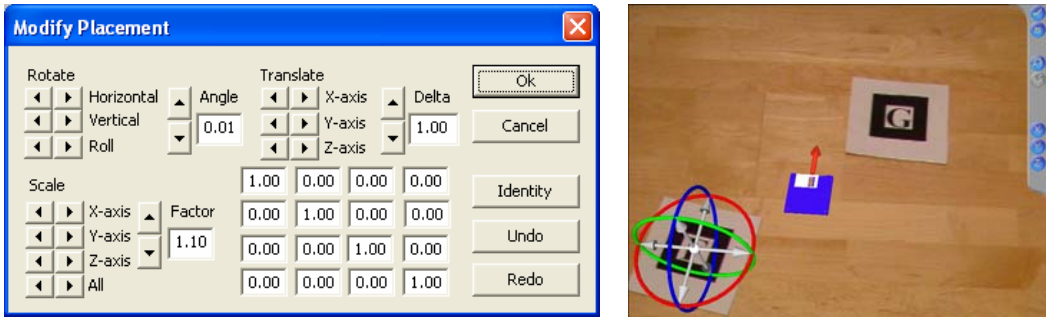


Figure 24: GUI based and AR based placement tool of AMIRE.

We expect more input in this area in the coming years, because current solutions are not the best in regards to usability and GUI interfaces. Figure 25 shows how complex the component dependencies can be – even if the MR application from the logical point of view seems to be easy.

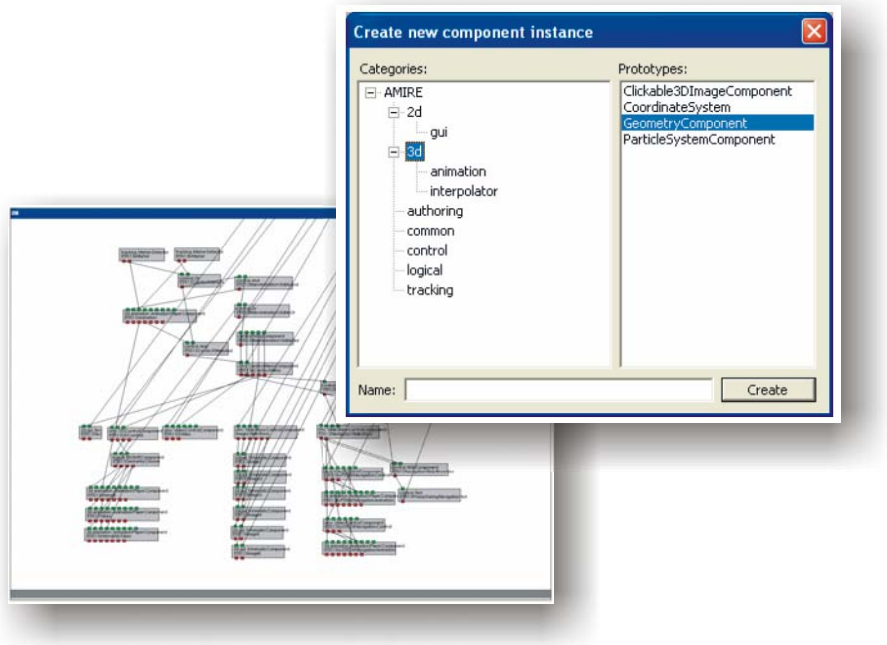


Figure 25: Authoring component schema of AMIRE.

CONCLUSION

The implementation of Mixed Reality applications is a very difficult task, because it includes a lot of different skills (e.g. sound programming, real-time graphics, different hardware in- and output devices, possibly AI techniques etc.). Therefore, we need a good architecture. We described a component-oriented approach for developing Mixed Reality applications. The benefits of this development approach are a fast implementation that allows an easy integration of corresponding authoring tools.

We achieve the following benefits if we use a component-oriented approach:

- **Flexibility:** Each component can be connected to another if it corresponds to the right interface.
- **Reusability**: Components can naturally be reused in the framework.
- **Extensibility:** Different components have to be added to the framework afterwards. A redesign of the framework has to be avoided by adding new components.
- **Easy communication**: The component communication has to be as simple as possible and as fast as possible.

The main advantage of the component-oriented approach is the high degree of flexibility and extensibility: even if the underlying technology is different, we can change the tracking system quite simply, but the system runs just as well.

REFERENCES

Amire (2003), http://www.amire.net

April (2003), http://www.studierstube.org/april/

Azuma, R. (1997). A survey of Augmented Reality. In Presence: Teleoperators and Virtual Environments 6, 4 (August 1997), 355-385.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., MacIntyre, B. (2001). Recent Advances in Augmented Reality. IEEE Computer Graphics and Applications 21, 6, 34-47.

Bauer M. et al. (2001), Design of a Component-Based Augmented Reality Framework. Int. Symposium on Augmented Reality, ISAR 2001, New York

Bimber, O., Fröhlich, B., Schmalstieg, D., and Encarnação, L.M. (2001). The Virtual Showcase. IEEE Computer Graphics & Applications, vol. 21, no.6, pp. 48-55, 2001.

Brandl A. (2003), Entwicklung einer interaktiven Möbelbauanleitung auf Mixed Reality Basis, Master Thesis, Upper Austria University of Applied Sciences, Media Technology and Design, July, 2003, Hagenberg, Austria.

Dachselt R. (2001), Contigra - Towards a Document-based Approach to 3D Components, In Proceedings “Structured Design of Virtual Environments and

3D-Components", Workshop at the Web3D Workshop, Shaker Verlag, Aachen, ISBN: 3-8265-9801-6.

Dobler D., Haller M., Stampfl P. (2002), ASR - Augmented Sound Reality,., In ACM SIGGRAPH 2002 Conference Abstracts and Applications, San Antonio, Texas, pp. 148.

Dörner R. and Grimm P. (2001) - Building Building 3D Applications with 3D Components and 3D Frameworks, in Proceedings Structured Design of Virtual Environments and 3D-Components: Workshop at the Web3D Workshop, Shaker Verlag, Aachen, ISBN: 3-8265-9801-6.

Dwarf (2003), http://www.augmentedreality.de

EON Reality (2003), http://www.eonreality.com/

Friedrich W. (2000), ARVIKA Augmented Reality for Development, Production, and Service, Tagungsband des Informationsforum Virtuelle Produktentstehung (IVIP).

Geiger C. Reimann C., Rosenbach W. (2000), Design of Reusable Components for Interactive 3D Environments. In Usability Centred Design and Evaluation of Virtual 3D Environments, Germany.

Haller M. (2001), A component oriented design for a VR based application, In International Workshop on Structured Design of Virtual Environments and 3D-Components at the Web3D 2001 Conference, Paderborn, Germany.

Haller M., Dobler D., Stampfl P. (2002), Augmenting the Reality with 3D Sound Sources, In ACM SIGGRAPH 2002 Conference Abstracts and Applications, San Antonio, Texas, pp. 65.

Haller M., Holm R., Priglinger M., Volkert J., and Wagner R. (2000), Components for a virtual environment, Workshop on Guiding Users through Interactive Experiences: Usability Centred Design and Evaluation of Virtual 3D Environments, Paderborn, Germany.

Haller M., Zauner J., Hartmann W., and Luckeneder T. (2003), A generic framework for a training application based on Mixed Reality, Technical report, Upper Austria University of Applied Sciences, Media Technology and Design.

Kato H., Billinghurst M., Blanding B., May R. (1999), ARToolKit, Technical Report, Hiroshima City University.

MacIntyre B., Maribeth Gandy, Jay D. Bolter, Steven Dow, Brendan Hannigan (2003), DART: The Designer's Augmented Reality Toolkit. ISMAR 2003: 329-330

Milgram, P., Kishino, F. (1994). A taxonomy of mixed reality visual displays. IEICE Transactions on Information Systems, Vol. E77-D, No.12 December 1994.

Piekarski W., Thomas B. H. (2003), An Object-Oriented Software Architecture for 3D Mixed Reality Application. In ISMAR 2003, The Second International Symposium on Mixed and Augmented Reality, pp. 247-256, IEEE, Tokyo.

Save (2003), www.faw.uni-linz.ac.at/save

Schmalstieg D., Fuhrmann A., Szalavari Zs., Gervautz M. (1996), "Studierstube" - An Environment for Collaboration in Augmented Reality, Extended abstract appeared in: Proceedings of Collaborative Virtual Environments '96, Nottingham, UK.

Studierstube (2003), http://www.studierstube.org

Virtools (2003), http://www.virtools.com/

VRPN (2003), http://www.cs.unc.edu/Research/vrpn/

Web3D (2003), http://www.web3d.org/x3d/spec/vrml/vrml97/

Zauner J., Haller M., Brandl A., Hartmann W. (2003), Authoring of a Mixed Reality Assembly Instructor for Hierarchical Structures, In ISMAR 2003, The Second International Symposium on Mixed and Augmented Reality, pp. 237-246, IEEE, Tokyo.