An Infrastructure for Realizing Custom-Tailored Augmented Reality User Interfaces

W. Broll, I. Lindt, J. Ohlenburg, I. Herbst, M. Wittkämper, T. Novotny

Abstract— Augmented Reality (AR) technologies are rapidly expanding into new application areas. However, the development of AR user interfaces and appropriate interaction techniques remains a complex and time-consuming task. Starting from scratch is more common than building upon existing solutions. Furthermore, adaptation is difficult, often resulting in poor quality and limited flexibility regarding user requirements. In order to overcome these problems, we introduce an infrastructure for supporting the development of specific AR interaction techniques and their adaptation to individual user needs. Our approach is threefold: a flexible AR framework providing independence from particular input devices and rendering platforms, an interaction prototyping mechanism allowing for fast prototyping of new interaction techniques, and a high-level user interface description, extending user interface descriptions into the domain of AR. The general usability and applicability of the approach is demonstrated by means of three example AR projects.

Index Terms— C.2.4 Distributed Systems --- Distributed applications, H5.1. Multimedia Information Systems --- Artificial, augmented and virtual realities, H.5.2 User Interfaces, H.5.3.b Collaborative computing, I.3.2 Graphics Systems --- Distributed/network graphics, I.3.6 Methodology and Techniques --- Device independence, Graphics data structures and data types, Interaction techniques, I.3.7 Three-Dimensional Graphics and Realism --- Virtual reality

1 Introduction

Augmented Reality (AR) [3] [24] enhances our physical environment by adding virtual objects. Alongside Virtual Reality (VR), which is aimed at complete immersion in an artificial environment, it is becoming an increasingly more important technology. Due to the rapid development of new hardware - including but not limited to more powerful wearable computers [30], advanced visual display technologies (www.lumus-optical.com), and input devices - our understanding of using and interacting with computers is being redefined and is starting to affect our everyday work [38].

AR has the potential to positively impact a large number of application areas. During the last few years, a number of promising prototypes have been created (including architecture and urban planning [7], [28], [40], engineering and production planning [11], [14], [18], [37], [41], education, learning and training [42], and gaming [29]). Nevertheless, the creation of appealing content remains a difficult, cumbersome and time-consuming task. While the creation of appropriate 3D or 2D information is important, the development of content is not limited to this task. Moreover, the realization of appropriate user interfaces and the underlying interaction techniques turns out to be the most challenging part.

In AR environments, traditional computers and their well-known devices, and interfaces disappear from the user's point of view. In addition, a large variety of highly specialized devices is introduced, solving particular problems very well but consequently not adaptable or transferable to other application areas. Existing AR applications have this problem in common with most VR environments. While today's 2D interfaces rely on the well-established WIMP (windows, icons, menus, pointers) interface metaphor, standards for user interface techniques or a set of interaction devices in VR and AR have not yet been established.

Within our previous AR projects we analyzed the requirements of different user groups and AR applications. It became apparent that a single user interface metaphor or a particular set of interaction techniques is not able to satisfy individual user needs. Moreover, the individual requirements regarding personal equipment (including devices) for each user group differed significantly. The same applies to the work environment, which has a large influence on the overall system (a system using voice input, for example, cannot be used in a noisy environment).

Thus, in this paper, we do not aim to provide a new general interface metaphor for AR or VR since this does not seem to be feasible given the large variety of applications and their individual requirements. Instead, we want to introduce an infrastructure, providing an adequate support for the development and adaptation of individual AR user interface techniques.

The approach presented in this paper is based on our experience and lessons learned from various projects with AR-based applications. Identification of hardware currently used within such projects and limitations in existing soft-

Wolfgang Broll is with the Fraunhofer Institute for Applied Information Technology FIT in St. Augustin, 53754, Germany. E-mail: wolfgang.broll@fit.fraunhofer.de
Irma Lindt is with Fraunhofer FIT. E-mail: irma.lindt@fit.fraunhofer.de
Jan Ohlenburg is with Fraunhofer FIT. E-mail: jan.ohlenburg@fit.fraunhofer.de
Iris Herbst is with Fraunhofer FIT. E-mail: iris.herbst@fit.fraunhofer.de
Michael Wittkämper is with Fraunhofer FIT. E-mail: michael.wittkaemper@fit.fraunhofer.de
Thomas Novotny is with Fraunhofer FIT. E-mail: thomas.novotny@fit.fraunhofer.de

ware provides the basis for our approach.

For each application area we found user interface development is based on multiple design cycles. This allows users to experience different AR user interfaces and multiple revisions according to the development of their needs and their improving experience with AR in general. For most users this was their first contact with AR technology, although many of them had used 3D or even VR before. We tried to support this process by offering various alternative tools and mechanisms during each of the development phases. Additionally, we attempted to achieve high flexibility regarding the hardware and software in use. This includes the ability to switch to different platforms (for example from laptops to PDAs), use another tracking system (sensor-based vs. computer vision-based), or switch between input devices.

In most other systems this would have required a complete re-design or at least a major re-programming effort. Our goal was to provide an infrastructure or framework allowing a maximum in flexibility regarding the user interface and interaction techniques while keeping the development effort reasonable.

In this paper, we describe our approach of supporting development of AR interaction techniques and show its feasibility within several applications. The structure of the paper is as follows: In Section 2 we compare our approach to previous work, focusing on VR/AR frameworks, interaction prototyping and the realization of AR user interfaces. In Section 3 we provide a general overview of fundamental multi-modal AR interaction techniques, defining the general scope of this paper. In Section 4, we present the main mechanisms of our approach. We introduce our VR/AR framework before presenting our approach on interaction prototyping and on AR user interface authoring. In Section 5, we present a couple of sample applications, discuss the individual mechanisms in detail and analyze the pro and cons of the overall approach. Finally, we draw a conclusion and provide an outlook on our future work.

2 Related Work

There are several approaches that support the realization of AR user interfaces including VR/AR toolkits and frameworks, authoring tools, and user interface description languages. The approaches differ mainly in the level of abstraction and supported control, and are suitable for different development phases.

Toolkit and framework-based approaches for AR applications offer solutions for typical user interface problems and speed up the implementation of an AR user interface. There are quite a number of frameworks for VR/AR applications. Each of them has a different focus and therefore makes it more or less applicable to individual application scenarios. A standard framework for VR/AR frameworks has not yet been developed. A good overview of existing frameworks and toolkits is given in [5], [8] and [12].

DWARF [4] is a component-based AR framework with a focus on wearable AR applications. Its main concept is to provide core functionality for AR systems by collaborating distributed services. Each service describes its abilities, needs and connectors in XML. Service managers running at each system site connect the distributed services according to their needs and abilities. CORBA is used for communication setup, but services may deploy more efficient protocols. Developers of an AR system can easily use or extend existing services or develop new ones. Several projects have already demonstrated the applicability of DWARF.

Another component-based framework for VR/AR applications is VHD++ [31]. As most frameworks it provides different services for the application and acts as a middleware to abstract the complexity of the underlying operating system and hardware layer. VHD++ is most appropriate for VR/AR applications with virtual human simulations due to its focus on virtual character simulations.

One of the main features of AR frameworks is to provide easy access to a wide range of devices. In order to maximize reusability of code and to minimize the effort for application developers exploiting these devices, many frameworks use an abstraction layer for input devices. OpenTracker [32], which is part of Studierstube system [34], provides such an abstraction layer for tracking devices and enables application developers to easily process multi-modal input data. XML is used for development, documentation, integration and configuration of the tracking device and its data flow within the application. Apart from the ability to substitute different tracking devices, the library also provides a set of operations for filtering, fusion and state transformation of the input data.

Contrary to these frameworks, the focus of our VR/AR framework is on supporting multiple distributed users sharing various input and output devices. While these frameworks either provide only limited universality regarding such devices, or make use of external libraries, (such as OpenTracker), our approach comprises an integrated universal device abstraction. Another major difference of our framework is that it comes along with its own render engine, providing a dedicated support for distributed applications in contrast to frameworks, where such functionality is added to existing render engines or render libraries later on (e.g. OpenInventor used by Studierstube).

Toolkit and framework-based approaches for the realization of AR applications – and especially user interfaces – require the developer to implement an application on a rather low level using standard programming languages. The development effort is high compared to other approaches, but many aspects of the user interface can be controlled in detail.

Authoring tools allow the creation of AR user interfaces on a higher abstraction level than toolkit or framework-based approaches. Typically, they provide standard user interface elements for AR that can be assembled and adapted.

DART [23] builds upon Macromedia Director and is realized as a plug-in. It provides an interface to different technologies including live video capture and marker tracking. The toolkit comes with several behaviors written in the interpreted programming language Lingo. Behaviors are modified by users to include content such as video, sound and 3D models in the AR application and to program the user interface. Although several Lingo behaviors already exist, a user typically needs to understand and modify relatively complex Lingo scripts in order to realize an AR application.

The AMIRE [1] framework is component-oriented and allows for the development of various authoring tools adapted to domain-specific requirements. Authoring tools developed with the AMIRE framework, for example the Authoring Wizard [43] for furniture assembly or CATOMIR [44] for Mixed Reality authoring, use flow chart diagrams to visualize the steps required to assemble 3D objects. The authoring process is based on a graphical user interface using 3D widgets as interaction components and connection elements and tangible user interface elements. The provided authoring tools offer a large variety of interaction techniques for 3D content authoring, but are restricted to desktop environment usage.

iaTAR [21] is an immersive authoring tool for tangible AR applications. It offers authors a component-based approach to create 3D content in an intuitive way. Physical props such as simple pads and cubes are used for the authoring tasks. Additionally, virtual props such as a menu of 3D models and an inspector pad are available. To realize a tangible user interface, properties of 3D models can easily be connected using inspector pads. The connection status is displayed with visual “wires”. iaTAR is well suited for creating 3D content, but is limited to a specific set of interaction techniques.

Authoring tools as described above focus on an easy and rapid creation of AR user interfaces. They build up on standard AR user interface elements (for example 3D menus) and are frequently targeting non-programming experts. Typically, they require less effort than toolkit and framework-based approaches and guarantee faster results.

While component-based approaches in general are very suitable for fast authoring of AR user interfaces, they are quite often limited to a rather specific set of interaction techniques. The authoring environments presented above are restricted to a particular set of 3D widgets (AMIRE) or specific interaction (iaTAR). Adding new interaction techniques is either not possible, or requires a significant amount of programming. Contrary, our approach on interaction prototyping is based on rather simple yet general components providing basic functionality only. While assembling such components into an interaction prototype may be more complex than using the authoring tools above, it provides us with full flexibility to model arbitrary interaction techniques. It thus overcomes this basic limitation of these tools.

Description and language-based approaches allow for rather high-level descriptions of user interfaces using text-based descriptions.

APRIL [20], part of the Studierstube system, is an XML-compliant high-level description language for authoring AR presentations. APRIL describes media objects, their behaviors and possible user interaction. Possible devices, such as displays and input devices, are described in the hardware description layer that is partially based on OpenTracker. The description of the presentation flow (“the story”) is based on UML state charts and exported to the XML Metadata Interchange (XMI) format, the official standard for serializing UML diagrams. States within the UML diagram represent behaviors whereas transitions represent interaction. At runtime, depending on the available hardware, different media types may be used to render the same virtual object. APRIL focuses on user interfaces for digital storytelling and is well suited to author presentations.

CUIML [33] has been developed as part of DWARF framework. CUIML is based on UIML [2] and uses XSL transformations to convert a CUIML-based user interface description to markup languages that can be displayed on various output devices, including the markup languages VRML and HTML. Elements and states of a user interface are described as a finite state automaton. Due to its XSL-based implementation, CUIML is limited to markup languages.

InTml [13] is an XML-based specification language that allows describing a VR application irrespective of the underlying environment. In InTml scene objects, interaction devices, and interaction techniques are represented by components. A VR application is realized by interconnecting the input and output of different components.

GRAIL [27], a graphical user interface description language, was developed as part of the ARTHUR project [7] (see also Section 5.2). It uses a graphical WIMP-style 2D interface for associating 3D input devices with scene objects and for assigning particular behaviors. However, due to the huge amount of individual options and mechanisms, the creation of AR user interfaces remains limited to trained experts.

Description and language-based approaches allow for realizing AR user interfaces using standard text editors. They may speed up the development of AR user interfaces tremendously, but offer less control over the final user interface than toolkits and frameworks or authoring tools. In contrast to APRIL, InTml and GRAIL the user interface description language proposed in this paper (see Section 4.3) targets not only AR or VR user interfaces. While supporting AR-specific interaction techniques, such as tangible user interfaces, the description can also be rendered for graphical user interfaces. Contrary to our approach, CUIML generates user interfaces in a markup language, thus, limiting run-time flexibility.

In general, user interfaces for AR need to be carefully selected based on the application domain and the targeted user group. Early mock-ups and prototypes are helpful to keep users involved and to prevent them from becoming frustrated. A development methodology for AR user interfaces has been suggested by Kulas, Sandor and Klinker [19]. A thorough review of usability design and evaluation methods is given by Bowman, Gabbard, and Hix [9] and Gabbard, Hix, and Swan [15].

3 Interaction Techniques

Since established 3D or VR interaction techniques [10] cannot directly be applied to AR, we define a classification of interaction techniques specifically used within AR environments. They provide the scope of what has to be supported by our infrastructure. The different interaction techniques are subdivided into:

- spatial interaction
- command-based interaction

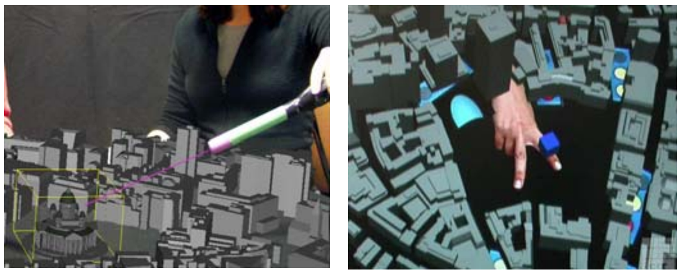

Fig. 1. Manipulation of a virtual object using a pointing device as an example of a spatial interaction technique (left), and gesture input as an example of a command-based interaction technique (right).

- virtual control interaction
- physical control interaction

Spatial interaction is based on manipulating spatial properties of physical objects. Spatial interaction is typically realized by dynamic pointing gestures, pointers and tangible user interfaces [16]. An example of spatial interaction is the selection of a virtual object using a pointer (see Fig. 1, left). While spatial interaction also exists in VR environments, it differs fundamentally as the physical objects are not part of the working environment and may not even be visible to the user (depending on the VR technology used). *Command-based interaction* consists of single or compound input linked to specific functionality. Command-based interaction is typically implemented as static gestures or voice commands. An example of a command-based interaction technique is the creation of a virtual object by performing a particular gesture (see Fig. 1, right). *Virtual control interaction* is based on 3D widgets representing a certain function. An example of virtual control interaction is a tool menu (see Fig. 2, left). *Physical control interaction* is based on physical tools or control panels that are extended to control not only physical but also virtual objects. Examples are a stage control panel to change the position of virtual backdrops (see Fig. 2, right). There is no equivalent to physical control interaction in VR environments.

The various interaction techniques offer different advantages. Spatial interaction for example is well suited for selecting virtual objects in 3D space and for spatial transformations. Command-based interaction is especially suitable to realize discrete input. Virtual control interaction implements a well-known interaction metaphor, whereas physical control interaction allows for the integration of physical tools in the user interface.

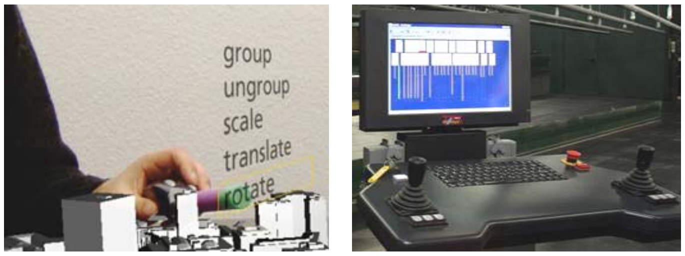

Fig. 2. Selection of a menu entry as an example of a virtual control interaction (left), and a stage controller as an example of a physical control interaction (right)

4 Realizing Interaction and User Interfaces

In this section, we introduce our infrastructure for realizing interaction techniques and user interfaces for AR applications. First, as a low-level mechanism, we present our AR framework. It provides flexible access to input devices and universal access to its components. Then we introduce our more advanced approach on interaction prototyping. While this approach still requires detailed knowledge of device output and scene contents, it provides full runtime flexibility and replaces programming by modeling. Finally, we present our high-level approach, which uses an XML-based description language to configure interaction techniques and to define whole AR user interfaces.

4.1 VR/AR Framework

Our extensible component-based VR/AR framework MORGAN [25] provides the basis for flexible integration and access to devices and to the creation and modification of AR content. It was especially developed to support distributed multi-user VR/AR applications. Due to the usage of software design patterns and the networking middleware CORBA, the framework enables developers to rapidly create applications by providing easy access to input and output devices and a high-level application-programming interface (API). It consists of a set of components, which provide special services for application developers including several input and output devices (for example tracking devices and speech input), and a sophisticated render engine. Fig. 3 provides an overview of the framework architecture. Applications are typically assembled from these components, individually configuring their instances and distribution. For a more detailed description of the MORGAN framework, see [25].

The central component of the framework is the *Broker*, which is responsible for the creation, deletion and retrieval of components. Each component instance has to register at the Broker in order to be accessible. However, the Broker can also be used for remote instantiation of such components. This feature is required when devices are physically connected to a specific computer and components providing access to these devices have to be executed locally (i.e., on that particular machine). The remote instantiation can be performed in several ways. One common strategy is to query the Broker for a specific component. If it does not yet exist, it will be instantiated on a designated host. Another strategy is to request a particular component on a specific host in case several instances of a particular component exist within a single application. The remote instantiation is realized by the factory pattern.

In order to enable the substitution of arbitrary input and output devices, we developed a concept for device independency. This concept defines a classification scheme for arbitrary computer devices with a focus on VR and AR as well as ubiquitous computing. Each device is classified within a hierarchy, which defines the common interfaces the device inherits from. Some devices combine logically different interaction metaphors. For example, a mouse is a pointing device that can be moved. It is also a device with buttons and a wheel. Such a combination of different aspects of a device is mapped in our class hierarchy by using

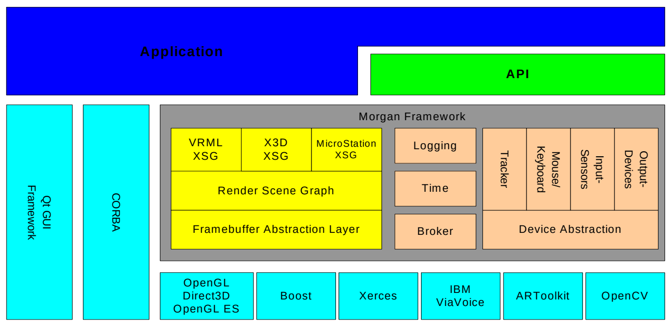

Fig. 3. Overview of the MORGAN framework

multiple inheritance. The actual *Mouse* device is derived from the classes *MousePointer*, *Button* and *Wheel*.

Therefore, it is possible to access a whole range of devices through their common super interfaces. For instance all 6DOF tracking devices inherit from the interface *Tracker6DOF*, which is subclassed from *PositionTracker3DOF* and *OrientationTracker3DOF*. A subscriber can subscribe at each of these interfaces and will only receive the relevant events, i.e. in case it subscribes at the *PositionTracker3DOF* interface it will only receive position updates without orientation.

This makes it very convenient to use for application developers since they are not only able to substitute different 6DOF trackers, but they are also enabled to substitute more unrelated devices, for example the keyboard for the mouse or even speech input. Exchanging speech input with keyboard command, such as in a noisy environment, is achieved by the adapter pattern, a design template, which maps input from one interface onto another.

Using the device abstraction hierarchy, MORGAN has already integrated many devices and is extended easily. Apart from a range of tracking devices, for example ARToolkit [17], InterSense IS-600, IS-900 and IS-1200, and the InertiaCube and GPS tracker, it already provides components such as speech command, mouse, keyboard, and eye tracker input, among others. Additionally, some output devices have been integrated, for instance tactile feedback devices or video and audio streams.

The integrated render engine accompanies the design of distributed multi-user VR/AR applications. This includes automatic distribution of updates, private sub-hierarchies, and an abstraction layer for frame buffers.

The render engine stores the scene including the 3D interface elements in a scene graph structure (a directed acyclic graph with a single root node). All geometric objects are located at the leafs of the scene graph, while the inner nodes of the graph are container nodes, which form the structure of the scene and group objects in a spatial or logical manner. The scene graph's purpose is to store the scene in an efficient way for rendering and querying purposes.

The render engine supports an abstraction of the frame buffer of the output device. Thus, the render engine is for example able to render the scene and the interface using OpenGL or Direct3D. Additionally, this mechanism simplifies the support of further frame buffers. Most PDAs, for example, do not support hardware accelerated frame buffers yet. It is also possible to omit the frame buffer entirely.

The distribution of application state changes and modified objects is crucial for multi-user applications. For example, if a user manipulates a virtual object, other users should immediately become aware of this action. On the other hand, not all information has to be distributed each time. For example, if a user opens a virtual menu to navigate through its entries, other users may be distracted by this information. Our distributed scene graph approach provides support for both cases. The virtual menus are an example of private sub-hierarchies used at the Mixed Reality Stage (see Section 5.1).

In general, the synchronization of distributed scene graphs is handled automatically by sending appropriate update messages. All updates are collected in regular intervals and distributed to all other scene graphs. This mechanism is also used to inform remote scene graphs about new objects. Updates containing the whole scene graph may be used to initialize additional (new) scene graph instances. Updates covering a certain period allow us to deal with temporary disconnections (for example for mobile participants using a smart phone or PDA).

Besides the possibility of developing applications by directly accessing the framework components using the CORBA interface, a C++ API is provided. The API hides the complexity of the framework and the distributed system, while providing full access to all input devices and the nodes of the scene graph. It allows inserting, copying, moving, deleting, and referencing of scene graph objects in addition to modifying node attributes directly. Furthermore, it publishes collision detection results and allows for picking queries. This provides the basis for fast and easy application development and incorporation of multiple devices.

4.2 Prototyping of Interaction Techniques

We use interaction prototyping as the fundamental testing and evaluation mechanism for interaction techniques within our projects. We distinguish interaction prototyping from pure user interface prototyping since it not only allows us to test the appearance or the look and feel of a user interface but also enables us to define and test interaction techniques with minimal effort.

When beginning to develop a particular VR or AR application the requirements for the final user interface are often not specified or not even known. Especially if the future users are not familiar with 3D interaction techniques (which is true for most application domains), they tend to have a very specific and restricted view of the possibilities offered by such an environment. Thus, starting the final implementation of a 3D user interface using the initial specifications based on the requirement analysis may predetermine the way users will have to use the system in the future. Thus, we use interaction prototyping to create early prototypes of such interfaces, while providing alternative solutions for particular interaction techniques at the same time.

Our approach to interaction prototyping is based on modeling interaction rather than programming it. We use an object-oriented, component-based approach. The text/XML-based component description is executed within the VR/AR-environment during runtime and may be

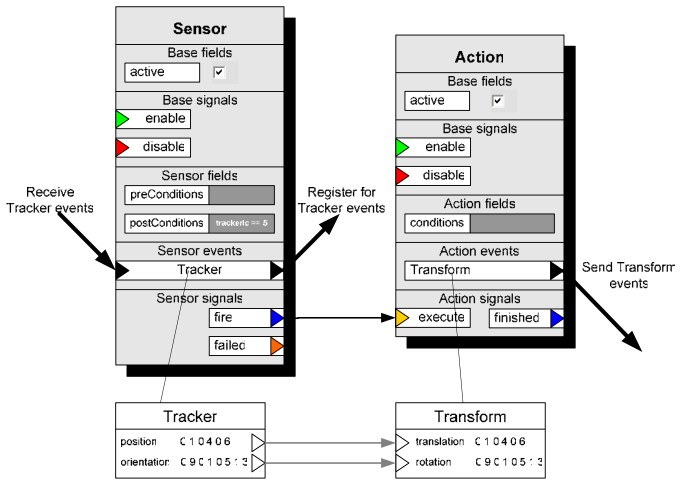

Fig. 4. Simple interaction prototype realizing a tangible user interface

modified (via an API) or reloaded at any time. Thus, many adaptations (such as switching from one speech command to another or replacing a button click by a gesture input) may be applied on the fly.

The components used in our approach represent the integral elements of user interaction. In general, all interaction depends on some form of user input. In a VR/AR environment, this is not limited to mouse or keyboard input but will typically include input from (6-DOF) trackers or other 3D input devices such as a space mouse or a data glove. In our approach, we use *Trigger* components to react to events related to a particular scene geometry (for example pressing a mouse button while the pointer picks a particular 3D object) and *Sensor* components to register for any other type of input events. We use *Action* components to apply changes to the scene graph by issuing appropriate events. In order to specify a control flow between the individual components, a signal/slot mechanism is used. Thus, the components actually form dataflow graphs. In addition, data values may be transferred between components by connecting the corresponding data fields. This is complemented by implicit type conversions. Invalid component connections will lead to runtime warnings but will not influence the execution of other components. This makes the overall approach very fault-tolerant.

Fig. 4 shows a simple example of an interaction prototype realizing a tangible user interface with a Sensor component that detects tracking events and an Action component that applies the received transformation to a particular scene graph object. The figure additionally shows the data transfer between the individual event fields. A set of fundamental interaction prototypes already exists, which can easily be applied to 3D scene objects or adapted to achieve the desired functionality. Cascading can be used to spread the dataflow across multiple interaction prototypes. This supports the modular definition of interaction techniques and user interface elements. It therefore allows for the formation of interaction technique abstraction layers, similar to the approach used by Unit [26] or InTml [13].

In order to realize a more complex interaction, the control flow can be influenced by *Decision* components. State-dependent interaction may be realized using *Memory* components. *Evaluator* and *Scripting* components may be used to evaluate expressions or even use small scripts. Additionally, interaction often depends on the current state of other objects. *Query* components allow gathering information about scene graph objects or the VR/AR environment. In 3D environments, general interaction (in contrast to particular interaction where for example a specific door is opened if the space key is pressed) requires either collision detection or picking (which actually is a collision or intersection between a ray and 3D scene objects). Thus, dedicated components (*Picking, Collision*) exist to pick along a ray or to register for collisions between objects. Finally, shared components provide support for distributed environments. *SharedTrigger* and *SharedSensor* components allow us to define the level of concurrency for a particular interaction. Two attempts to interact with one particular object may for example mutually exclude each other or be completely independent. *SharedAction* components allow the dataflow to spread across distributed graphs (similar to [26]), allowing for early or late synchronization. In order to realize time-dependent interaction, autonomous object behaviors and dynamics, *Timer*, *Sequencer*, and *Interpolator* components are available.

We extended Microsoft Visio by adding appropriate template shapes for each component to support the modeling of interaction prototypes (for example in Fig. 4 those templates were used).

Modeling interaction rather than programming it is not necessarily less complex. However, our experience so far indicates that modeling interaction based on a set of existing primitive components seems to be easier to understand for design-oriented professionals than programming. Although not the original intention of our approach, this allows designers or architects to create or adapt interaction techniques on their own. Furthermore, this approach extends naturally to new input devices as it provides universal (low-level) access to system events. This feature allows us to test new devices with this mechanism as soon as they are integrated in the underlying framework. Besides these advantages, we do not want to conceal the limitations of our approach. Application-specific interaction mechanisms often include plausibility checks or other application related "intelligent" mechanisms (often requiring access to an underlying application database), which are quite cumber-

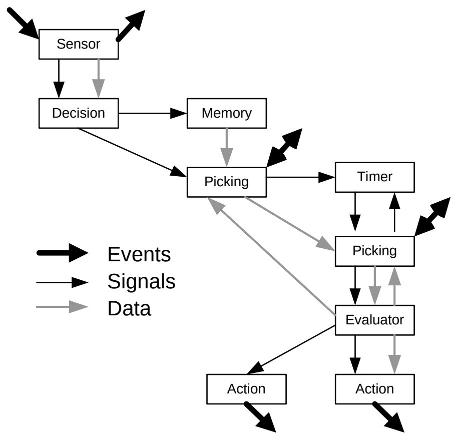

Fig. 5: More complex navigation interaction prototype

some or even impossible using our prototyping mechanism.

Finally, a second, more complex example demonstrates the realization of an interaction prototype implementing *Pacman*-style navigation for a 3D object, i.e. the target object will move into one direction until it collides with a wall where it will stop. It may be directed in any of the other four major directions (up, down, left, right), for example by appropriate voice commands. One possible solution based on our approach is as follows (see Fig. 5): A Sensor component is used to register for the appropriate voice commands. When any of the corresponding events is received, its data is forwarded to a Decision component, which is executed afterwards (i.e. the individual conditions are checked). Depending on the results, an appropriate picking vector is loaded from a Memory component into a Picking component. The Picking component is then executed, i.e. the picking is performed from the current position in the pre-loaded direction. If there is no barrier within the picking range, the actual moving direction is modified by forwarding the picking direction to a second Picking component, and a Timer component is started that will ensure a continuous movement into the selected direction. The Timer component is configured to be executed each frame. It activates the second Picking component to check whether further movement is possible. If that Picking component detects a barrier, it will disable the Timer engine. Otherwise, the picking direction is forwarded to an Evaluator component. This component will simply add the picking vector to the current location and provide the new location to the two Picking components and to an Action component. This Action component will send events to the scene graph, while a second Action component will issue events to play appropriate moving sounds during the movement. Both Action components will be executed by the Evaluator component.

4.3 User Interface Descriptions

While AR user interfaces are typically realized with a large variety of interaction techniques and interaction devices (see Section 3), most of them depend on specific hardware. Though abstraction layers for input devices exist (see Section 4.1), concepts for defining user interfaces independent of specific interaction techniques are still missing. As a result, many AR applications are hard-coded for a specific set of devices and cannot be used with different hardware setups. An exception to this is the use of different tracking devices since several tools exist that provide a universal interface to various trackers (see Section 2).

A more flexible approach to AR user interfaces can be realized by user interface description languages (UIDLs) [39]. UIDLs allow describing user interfaces irrespective of interaction devices and their capabilities. While several UIDL-based vocabularies for describing user interfaces for desktop and mobile PCs, PDAs and cellular phones [35] exist, there is only little or no support for AR user interfaces.

We developed a vocabulary for describing AR user interfaces in addition to traditional WIMP-based user interfaces. The vocabulary consists of user interface elements such as buttons, frames or sliders, input listeners, such as pose or selection listeners and actions referencing the actual application logic. It was designed to support all four types of interaction techniques as described in Section 3.

User interface elements can be perceived by the user and may have a visible, audible or tangible representation. For common user interface components, abstract entities are introduced. We distinguish between controls and containers. The control components include

- *Label*,
- *Button*,
- *Slider*,
- *TextInput*, and
- *Artifact.*

Label contains information that is typically related to another element. *Button* represents pre-defined functionality that can be selected by the user. *Slider* allows for the selection of numerical input within a certain range. *TextInput* may be used to provide textual input of arbitrary modality. *Artifact* represents arbitrary content.

The container components include

- *Frame*,
- *Menu*,
- *Sequence*, and
- *SelectionSet.*

Frame may be used to group elements. *Menu* is a hierarchical structure of arbitrary elements. *Sequence* can be used to display elements for a certain amount of time. *SelectionSet* contains a set of buttons and can be configured to allow single or multiple selections.

In order to support AR-specific physical control interaction techniques, each child element contains the attribute *mayBePhysical*. If this attribute is set to true and the current hardware setting contains an appropriate device with a button, the physical device is used. Otherwise, the element is realized by a virtual control. An example is a button, which may be available as a real physical button or will be rendered as a virtual button otherwise.

Furthermore, we use input listeners to deal with user input. Input listeners include:

- *PoseListener*,
- *FocusListener*,
- *ChoiceListener*, and
- *TextListener*.

A *PoseListener* receives position and/or orientation data from external devices such as trackers or from corresponding objects. A *FocusListener* is notified once the corresponding object is in focus, a *ChoiceListener* is called once the corresponding object is selected and a *TextListener* is called on

```
<part class="Artifact" id="objectID">
  <style>
    <property name="type">physicalObject</property>
    <property name="position">0.0 -2.0 0.2</property>
    <property name="orientation">
         0.3 0.1 0.0</property>
    <property name="isVisible">true</property>
    <property name="mayBePhysical">true</property>
  </style>
  ...
</part>
```

Fig. 6. An artifact described with our user interface definition language.

text input irrespective of the modality.

Spatial interaction techniques rely on *PoseListeners*. The position and/or orientation input can be used to create or manipulate objects. Command-based interaction listeners are supported by *ChoiceListeners*, *FocusListeners*, and *TextListeners*. Once a command has been recognized, the corresponding user interface element is notified.

Finally, actions realize the application logic. They are defined in a library and referenced in the user interface description. At runtime, an *ApplicationAdapter* calls the application logic that is dynamically linked.

We selected UIML [2] to implement the vocabulary. Fig. 6 shows an *Artifact* described with UIDL. The user interface description is fed into our user interface render engine. This engine determines the devices of the local environment and combines this information with personal preferences and the current platform settings to generate an appropriate user interface. Sample renderings exist for a GUI based on Qt (www.trolltech.com) and for an Augmented Reality interface based on the MORGAN API (see Section 4). An example of a user interface based on our vocabulary can be found in the CONNECT project in Section 5.3.

5 APPLICATION SAMPLES

We now present a couple of AR applications, which have successfully been realized using our infrastructure for creating interaction techniques. The individual subsections introduce the interaction techniques realized (TABLE 1) and the mechanisms used. While this proves the usability of the overall approach, it also provides valuable insight into the capabilities and limitations of each mechanism. The examples also show that a single-track approach may not be adequate to achieve suitable results, which take into account user requirements but also keep up with technological development in sufficiently complex applications.

TABLE 1

USED INTERACTION TECHNIQUES

Interaction Technique	mqube	ARTHUR	CONNECT
Spatial	Tangible Unit (TU, grey blocks with one to three dots in Fig. 7, left) View-pointer, a crosshair shown in the center of the user's view (Fig. 7, left)	3D pointers Placeholder Objects	Movable parts of the exhibit (for example airfoil in Fig.11)
Command-based	Voice commands Human Interface Device (HDI)	Gestures Voice commands	Voice commands Human Interface Device (HDI)
Virtual control	3D menus 3D tools	3D menus 3D tools	3D menu
Physical control	Light control panel Stage control panel (Fig. 7, right)	-	Sensors (temperature, CO_2 density) Human Interface Device (HDI)

5.1 mqube

Within the mqube project, we developed the Mixed Reality Stage, an interactive AR environment for collaborative planning of stage shows and events [6]. The planners of an event usually use a model stage with downscaled props, miniature lights and jointed dolls to illustrate ideas. On the one hand, this approach is more cost efficient than assembling real world objects on a real world stage. On the other hand, it contains a lack of dynamic and makes the planning of choreographies with jointed dolls impossible. To counteract this lack, we used a model stage as a frame of reference and augmented it with appropriate virtual objects such as props or virtual characters. The Mixed Reality Stage user interface supports the spatiotemporal and the collaborative aspects of the planning task. Therefore, we chose an approach that combines different interaction techniques (see TABLE 1) [22].

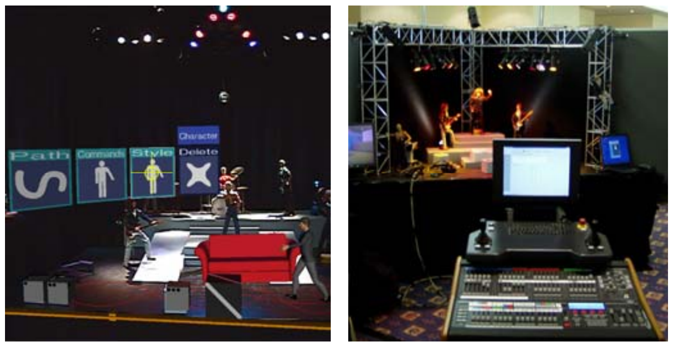

Fig. 7. 3D menu, view-pointer and tangible unit as user interface elements of the Mixed Reality Stage (left), stage control board and lighting board in front of the model stage (right)

While most parts of the final user interface were realized on top of the MORGAN API, the user interface description of the Mixed Reality Stage is partially based on an XML file. For example, it is possible to specify voice commands and the triggered actions, the look of 3D menus and tools, and the tracking devices used. The realization of this interface description provided useful insights regarding the development of the current UIDL-based approach (Section 4.3). Interaction prototyping was used for the initial tests of the view pointer and for applying the input from the physical control units for lighting and stage control to the 3D scene.

A well-suited technique for collaboration support is the use of graspable interfaces. The physical nature of the real objects restricts concurrent access. To realize private menus where the state of the individual menu is not shared among individual users, we relied on the mechanisms provided by our framework.

5.2 ARTHUR

ARTHUR [7] is a collaborative tabletop AR environment supporting architectural design and urban planning. The objective of ARTHUR was to create an interactive environment where architects, engineers, and customers involved in design and review meetings may collaborate without radically altering their established working procedures. We wanted to create a working environment where users would intuitively use the tools without a long training period. This approach is reflected by the use of intuitive interaction techniques, which allow even untrained users to

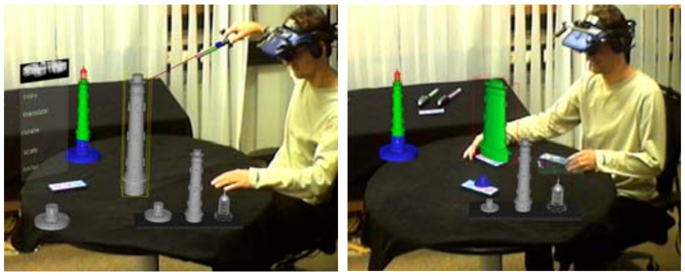

Fig. 8. A user testing different interaction techniques using a 5-DOF Pointer (left) and a 3-DOF placeholder object (right)

quickly benefit from the enhancements provided by the AR environment. Additionally, existing tools such as a CAD system and simulation programs already used by potential customers were integrated, providing seamless transition between their individual everyday work and the collaborative work at round table meetings.

The interaction techniques available within ARTHUR are spatial, command-based, and virtual control interaction techniques (see Section 3). They are based on placeholder objects, pointers, and hand gestures.

While the final tool for building user interfaces, GRAIL [27], was developed by a project partner based on our MORGAN API, basic interaction techniques were extensively tested using our interaction-prototyping approach.

Although our application partners (architects) were quite familiar with desktop-based 3D interfaces, they had not used AR before. Thus, a set of tools using different interaction techniques was created (see Fig. 8 and Fig. 9). We completely realized these tools as in-scene user interface descriptions based on interaction prototyping components.

This approach allowed us to compare various input techniques for creating and manipulating 3D content within the AR environment. Several manipulation tools, for example to move, to rotate, to scale, and to change the color of objects were tested using audible or optical feedback, and different versions of highlighting and information feedback.

When we incorporated support for multiple users within the ARTHUR system, we also tested multi-user interaction techniques. Again, our interaction prototyping mechanisms were applied. Fig. 9 shows two users operating a multimodal menu-based interaction technique via microphone to activate a particular manipulation tool.

To enable architects to interactively create and manipulate virtual CAD objects, the Microstation CAD system from Bentley Systems was integrated in the ARTHUR system. This allows us to display arbitrary CAD models on top

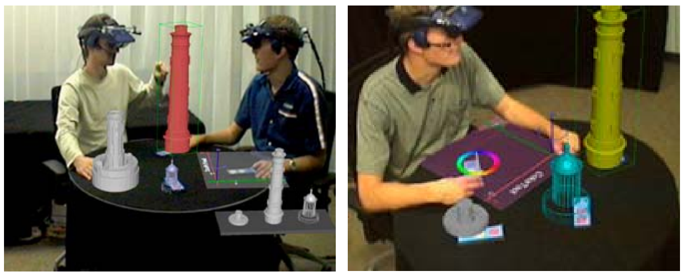

Fig. 9. Two users constructing a building (left) and using different tools to manipulate objects, for example a color tool (right)

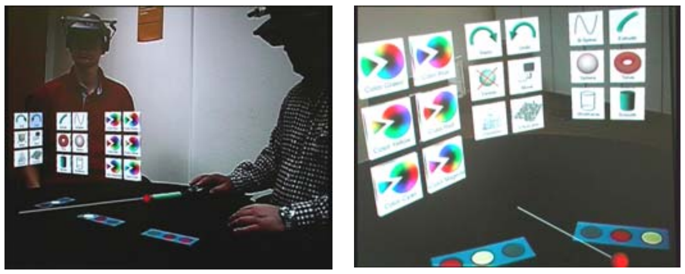

Fig. 10. 3D user interfaces providing access to the CAD system, generated upon XML-based user interface descriptions

of the augmented round table and to modify the scene in real-time choosing a CAD desktop or the ARTHUR system itself. While changes such as creating, modifying or deleting an object are sent to the CAD application to ensure the same behavior of commands, changes may also be made directly within the Microstation environment. In both cases, the results of the actions are distributed in real-time back to the system. As such, "full-fidelity" CAD models are created and interactively displayed by the system; there are no translation or conversion issues. Thus, the CAD model can be used in downstream design processes.

The user interface for the CAD integration was realized with an early version of the UIDL described in Section 4.3. The employed version already allowed the description of elementary user interface elements and their hierarchical grouping. With the XML-based description, user interface elements could easily be adapted for different scenarios. The description was especially useful for mapping the AR user interface elements to elements of the CAD application (see Fig. 10). Unique identifiers for certain functionality available in the CAD application were added to the description file. At runtime, the corresponding functions were called when an AR user interface element was selected.

5.3 CONNECT

Within the CONNECT project [36], we developed mobile AR systems that are used by students in technical science centers to learn the content of the curriculum in an informal way. In each of three test runs, four different exhibits at four European sites are individually augmented by virtual objects, which serve a number of purposes. Scientific models of complex phenomena are visualized in situ, making the invisible visible. Sensor data collected at the exhibits is prepared and displayed in a comprehensive way. Audiovisual introductory and background information, provided by the science centers, the teachers and the students, is presented around the exhibits. Interaction takes place either through the physical parts of the exhibits or via the user interface of the mobile AR system.

At the *Airfoil* exhibit – used to teach why planes fly – a physical wing can be rotated inside the air stream of a fan (see Fig. 11). Depending on the wing angle and the air stream around the wing, the lifting forces change, resulting in a continuous update of the augmented 3D content. At the *Airtrack* exhibit – a test bed for many mechanics experiments related to velocity, acceleration, force, and friction – gliders may be pulled along a track with dynamometers. The *Biotube* exhibit is used to explain photosynthesis of plants. By breathing into the tube, the CO_2 level increases.

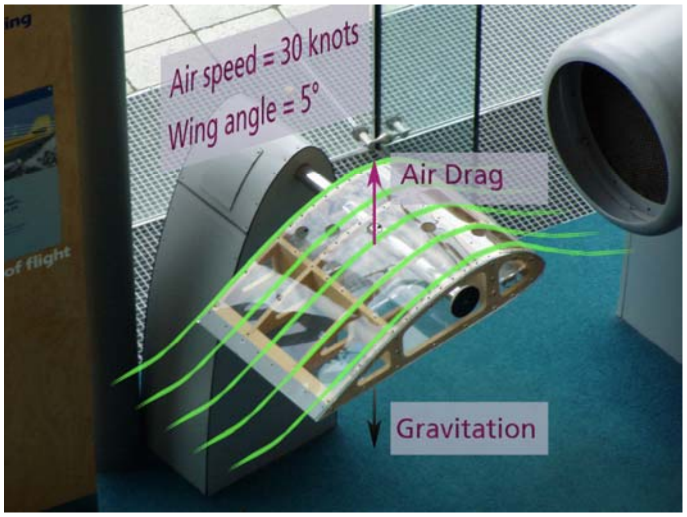

Fig. 11. Physical airfoil augmented by a virtual air stream around the wing and supplementary information.

This influences the augmented 3D content representing the oxygen and carbon dioxide concentration inside the tube. Neon lights may be switched on or off, influencing the photosynthesis process. The *Hot Air Balloon* Exhibit – demonstrating the Bernoulli Principle – is controlled by a heater, which makes the balloon rise when it is close to it. Access to functionality specific to an exhibit is provided through menu items. This way, individual 3D content or a specific representation can be selected. Navigating through virtual menus and selection of menu entries is achieved by voice input or a human interface device with a button and a wheel.

All three mechanisms provided by our infrastructure are integrated in the CONNECT project. After specifying user requirements, a prototype of the mobile AR system user interface was realized based on the interaction prototyping mechanism. For the improved version, all scenarios are specified in UIML based on the vocabulary proposed by us. This interface description language also serves as the interface to a web platform developed by a project partner. The platform enables teachers to modify the content of the students' user interface. For the final version of the user interface, the description is modified taking account of user tests performed with the two previous versions.

With the UIDL described in Section 4.3 the user interface of the airfoil exhibit can be described using *Label* and *Artifact* as controls and a *PoseListener* as an input listener. The physical wing is described as an *Artifact* with the *mayBePhysical* attribute set to true. Once the wing is rotated, the *PoseListener* is notified. The new air speed, wing angle, air drag and gravitation values are calculated, and the corresponding labels are updated.

5.4 Discussion

While all three mechanisms provided by our infrastructure were used in each of the applications presented in this paper, the extent to which they were realized differed significantly. All projects used the underlying MORGAN framework benefiting from the device abstraction facilities. One of the lessons learned in this area was that even with an appropriate framework the integration of new devices (such as the physical control units in mqube) may require a significant amount of work as these units do not provide a suitable API to access.

The API of the framework was used extensively in all applications. The actual applications and the final interaction techniques including major parts of the user interface were realized on that level. In ARTHUR and mqube, this interface was even used by external project partners for their realization of more advanced user interface techniques or even GUI builder tools (ARTHUR). Those project partners who were, for example, architects or media designers (i.e. no AR experts), would not have been able to contribute to the development without such a mechanism and an appropriate device abstraction. However, the use of such a relatively low-level API also requires deep insight into the scene graph structure. This fact and the learning curve for the API itself were the main restraints for other developers.
In all projects we also applied our interaction prototyping mechanism to early mock-ups and testing of user interface components and interaction techniques. It proved to be very flexible and even more powerful than we expected (for example in ARTHUR the 3D editing environment was originally planned to be programmed on the API level). Thus, certain remainders of this mechanism were often used even in the final applications as they proved to provide the required functionality and were easy to manipulate in the scene. The realization of more complex interfaces and techniques, however, proved to be difficult. One of the reasons was the text-based interface for assembling and configuring the appropriate components. Another reason was the lack of knowledge in object-oriented design of interaction techniques or how to realize them based on the prototyping mechanism. A set of common interface modules requiring only minor adaptations to assemble a basic scene would have provided a better starting point.

In all projects except the ongoing CONNECT project, the use of user interface descriptions was not originally planned. At a certain point during the development of the application interface however, the use of such a configuration was always desirable in order to be flexible and to provide a relatively abstract high-level definition of the user interface. Thus, our approach in this area originally evolved from the experience made in the mqube project. For the CAD interface within ARTHUR, a much more elaborate approach was used, which was already very similar to the current mechanism. Even with such a general mechanism finally available, the CONNECT project showed that new applications will always require additional, simpler mechanisms in earlier project phases.

We expect all three mechanisms to exist and to be applied in future projects. With appropriate support for graphical interaction modeling for the prototyping mechanism or with a more powerful and more flexible approach for the user interface description the amount of API-based interaction techniques and user interface development will probably be further reduced in the future. Nevertheless, certain application-specific tasks, which require the use of low-level realization, will always remain. However, the amount of such developments compared to the other two mechanisms will definitely decrease.

6 Conclusion and Future Work

In this paper, we presented our infrastructure for the support of realization of application-specific interaction techniques in order to create individual and adaptable user interfaces for AR environments. We identified the basic interaction techniques used in AR interfaces. Further, we introduced the three mechanisms of our approach in detail. First, we presented our MORGAN framework, providing universal access to input devices and interface components while using a relatively low-level programming interface. Then, we showed how our component-based interaction prototyping mechanism can be used to model AR interaction techniques in a very flexible manner. We further introduced an interface description language allowing for interface specifications irrespective of a specific environment and the devices available. Finally, we showed how these components had been successfully used in various AR projects to set up, develop, adapt, and at the end deploy custom interaction techniques and new user interface concepts.

In our future work, we will continue to test and improve the capabilities of the user interface description engine, especially towards mobile devices and pervasive environments. We further intend to realize a visual editing and simulation environment for interaction prototyping mechanisms that simplify modeling of interaction techniques while reducing the number of modeling errors. Finally, we are currently developing an object-oriented scene-graph-based application model. This will naturally complement our existing components.

Acknowledgment

The authors wish to thank their colleagues Peter Henne and Axel Linder for their contribution and comments, and Wolfgang Appelt and Andreas Strothmann for proofreading the paper. The ARTHUR project was partially funded by the European Commission as part of the IST program within the 5th framework (project no. IST-2000-28559). The mqube project was partially funded by the German Federal Ministry of Education and Research. The CONNECT project is partially funded by the European Commission in FP6 (FP6-2002-IST-1-507844).

References

[1] D. Abawi, R. Dörner, M. Haller, and J. Zauner, "Efficient Mixed Reality Application Development", in *Proc. of the 1st European Conference on Visual Media Production*, London, March, 2004.

[2] M. Abrams, "Device-Independent Authoring with UIML", in *Proc. of the W3C Workshop on Web Device Independent Authoring*, 1999.

[3] R. Azuma, "A Survey of Augmented Reality", in *Presence: Teleoperators and Virtual Environments*, Vol. 6, No. 4, 1997, pp. 355 – 385.

[4] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reichert, S. Riß, C. Sandor, and M. Wagner, "Design of a Component-Based Augmented Reality Framework", in *Proc. of The Second IEEE and ACM International Symposium on Augmented Reality (ISAR 2001)*, New York, October, 2001, Los Alamitos, California, pp. 45-54.

[5] A. Bierbaum, and C. Just, "Software Tools for Virtual Reality Application Development", *ACM SIGGRAPH 98 Course #14: Applied Virtual Reality*, pp. 3.1 – 3.45, ACM SIGGRAPH, 1998.

[6] W. Broll, S. Grünvogel, I. Herbst, I. Lindt, M. Maercker, J. Ohlenburg, and M. Wittkämper, "Interactive Props and Choreography Planning with the Mixed Reality Stage", in *Proc. of ICEC 2004*, Eindhoven, September 1-3, 2004, pp. 185-192 .

[7] W. Broll, I. Lindt, J. Ohlenburg, M. Wittkämper, C. Yuan, T. Novotny, C. Mottram, A. Fatah, and A. Strothmann, "ARTHUR: A Collaborative Augmented Environment for Architectural Design and Urban Planning", in *Proc. of the fourth International Symposium on Human and Computers (HC 2004)*, pp. 102-109, also to be published in the *Journal for Virtual Reality and Broadcasting, Vol. 1, No. 1,* 2004.

[8] B. Bruegge, A. MacWilliams, T. Reicher, "Study on Software Architectures for Augmented Reality Systems", Report to the ARVIKA consortium; technical report TUM-I0410.

[9] D. A. Bowman, J. L. Gabbard, D. Hix, "A survey of usability evaluation in virtual environments: classification and comparison of methods", in Presence: Teleoperators and Virtual Environments, Vol. 11 , Issue 4, 2002, pp. 404 – 424.

[10] D. Bowman, E. Kruijff, J. LaViola, and I. Poupyrev, *3D User Interfaces: Theory and Practice*, Addison-Wesley, Boston, 2004.

[11] F. Doil, W. Schreiber, T. Alt and C. Patron, "Augmented reality for manufacturing planning", in *Proc. of the Workshop on Virtual Environments 2003*, (Zurich, Switzerland), pp. 71-76, ACM Press, 2003.

[12] C. Endres, A. Butz, A. MacWilliams, "A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing", *Mobile Information Systems Journal*, Inauguration Issue: January-March 2005, pp. 41-80, IOSPress.

[13] P. Figueroa, M. Green and H. J. Hoover, "InTml: a description language for VR applications", in *Proc. of the 7th International Conference on 3D Web Technology,* Tempe, Arizona, USA pp. 53-58, 2002.

[14] W. Friedrich, "ARVIKA – Augmented Reality for Development, Production and Service", in *Proc. of the First International Symposium on Mixed and Augmented Reality (ISMAR 2002)*, pp. 3–4, IEEE Computer Society, 2002.

[15] J. L. Gabbard, D. Hix, J. E. Swan, "User-Centered Design and Evaluation of Virtual Environments", in *IEEE Computer Graphics and Applications*, Volume 19, Number 6, November/December, 1999, pages 51-59.

[16] H. Ishii, B. Ullmer, "Tangible Bits. Towards Seamless Interfaces between people, Bits and Atoms", in *Proc. of CHI'97*, Atlanta, Georgia 1997.

[17] H. Kato, M. Billinghurst, B. Blanding, and R. May, "ARToolKit". Technical Report. Hiroshima City University. December 1999

[18] G. Klinker, A.H. Dutoit, M. Bauer, J. Bayer, V. Novak, and D.Matzke, "FataMorgan – A Presentation System for Product Design", in *Proc. of the First International Symposium on Mixed and Augmented Reality (ISMAR 2002)*, Darmstadt, 2002.

[19] C. Kulas, C. Sandor, G. Klinker, "Towards a Development Methodology for Augmented Reality User Interfaces", in *Proc. of the International Workshop exploring the Design and Engineering of Mixed Reality Systems - MIXER 2004*, Funchal, Madeira, January 2004.

[20] F. Ledermann, D. Schmalstieg, "APRIL: A High-level Framework for Creating Augmented Reality Presentations", in *Proc. of* IEEE Virtual Reality 2005, 2005.

[21] G.A. Lee, C. Nelles, M. Billinghurst, and G.J. Kim, "Immersive Authoring of Tangible Augmented Reality Applications", in *Proc. of the Third International Symposium on Mixed and Augmented Reality (ISMAR 2004)*, 2004.

[22] I. Lindt, I. Herbst, and M. Maercker, "Interacting within the Mixed Reality Stage", in *Workshop Proc. AVIR 2003*, Geneva, Switzerland, September, 2003.

[23] B. MacIntyre, M. Gandy, S. Dow, and J. D. Bolter, "DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences", in *Proc. of the User Interface Software and Technology conference (UIST'04)*, Santa Fe, New Mexico, October 24-27, 2004.

[24] P. Milgram, F. Kishino, "A Taxonomy of Mixed Reality Visual Displays", in IEICE Transactions on Information Systems, Vol. E77-D, No. 12, December 1994.

[25] J. Ohlenburg, I. Herbst, I. Lindt, T. Fröhlich, and W. Broll, "The MORGAN Framework: Enabling Dynamic Multi–User AR and VR Projects", in *Proc. of the ACM Symposium on Virtual Reality Software and Technology (VRST 2004)*, pp. 166-169, ACM, 2004.

[26] A. Olwal, S. Feiner, "Unit: Modular Development of Distributed Interaction Techniques for Highly Interactive User Interfaces", in *Proc. of the International Workshop exploring the Design and Engineering of Mixed Reality Systems - MIXER 2004*, Funchal, Madeira, CEUR, Workshop Proceedings, 2004.

[27] A. Penn, C. Mottram, A. Fatah gen. Schieck, M. Wittkämper, M. Störring, O. Romell, A. Strothmann, and F. Aish, "Augmented Reality meeting table: a novel multi-user interface for architectural

design", *Recent Advances in Design and Decision Support Systems in Architecture and Urban Planning*, (Jos P. van Leeuwen and H. Timmermans, eds.), Kluwer Academic Publishers, 2004, pp. 213–231.

[28] W. Piekarski, B. Thomas, D. Hepworth, B. Gunther, and V. Demczuk, "An Architecture for Outdoor Wearable Computers to Support Augmented Reality and Multimedia Applications", in *Proc. of the 3rd International Conference on Knowledge-Based Intelligent Information Engineering Systems*, Adelaide, South Australia, pp 70-73, 2000.

[29] W. Piekarski, and B. Thomas, "ARQuake: the outdoor augmented reality gaming system", *Communications of the ACM, Vol 45, No. 1*, pp. 36-38, ACM, 2002.

[30] C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, L. Thiele, and G. Tröster, "The case for reconfigurable hardware in wearable computing", *Personal Ubiquitous Computing, Vol. 7, No 5,* pp. 299-308, Springer, 2004.

[31] M. Ponder, G.Papagiannakis, T. Molet, N. Magnenat-Thalmann, D. Thalmann, "VHD++ Development Framework: Towards Extendible, Component Based VR/AR Simulation Engine Featuring Advanced Virtual Character Technologies", in *Proc. of Computer Graphics International 2003 (CGI 2003),* IEEE Computer Society Press, 2003.

[32] G. Reitmayr and D. Schmalstieg, "An Open Software Architecture for Virtual Reality Interaction", in *Proc. of the ACM Symposium on Virtual Reality Software & Technology 2001 (VRST 2001)*, pp. 47-54, Banff, Alberta, Canada, 2001.

[33] C. Sandor and T. Reicher, "CUIML: A Language for the Generation of Multimodal Human-Computer Interfaces", in *Proc. of the European UIML Conference*, 2001.

[34] D. Schmalstieg, A. Fuhrmann and G. Hesina, "Bridging Multiple User Interface Dimensions with Augmented Realty", in *Proc. of IEEE and ACM International Symposium on Augmented Reality (ISAR 00)*, pp. 20-29, Munich, Germany, 2000.

[35] R. Simon, M. Jank, and F. Wegscheider, "A Generic UIML Vocabulary for Device- and Modality Independent User Interfaces", in *Proc. of the 13th International World Wide Web Conference*, New York, USA, May 17-22, 2004, ACM Press.

[36] S. Sotiriou, E. Chatzichristou, S. Savas, A. Pyrini, N. Ouzounoglou, M. Gargalakos, R. Makri, P. Tsenes, L. D. Dierking, H. Salmi, A. Hoffstein, and S. Rosenfeld, "CONNECT: Designing the Classroom of Tomorrow by Using Advanced Technologies to Connect Formal and Informal Learning Environments", in *Proc. of the IADIS International Conference on Cognition and Exploratory Learning in Digital Age (CELDA 2004)*, Lisbon, Portugal, December 15-17, 2004.

[37] A. Tang, C. Owen, F. Biocca, and W. Mou, "Comparative effectiveness of augmented reality in object assembly", in *Proc. of the Conference on Human factors in Computing Systems*, pp. 73-80, Ft. Lauderdale, Florida, USA, ACM Press, 2003.

[38] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. Picard, and A. Pentland, "Augmented Reality through Wearable Computing", *Presence, Vol. 6, No. 4*, August 1997, pp. 386-398, 1997.

[39] S. Trewin, G. Zimmermann, and G. Vanderheiden, "Abstract User Interface Representations: How well do they Support Univeral Access?", in *Proc. of CHI,* 2003.

[40] A. Webster, S. Feiner, B. MacIntyre, W. Massie, and T. Krueger, "Augmented Reality in Architectural Construction, Inspection and Renovation", in *Proc. of ASCE 3rd Congress on Computing in Civil Engineerings*, pp. 913 – 919, 1996.

[41] S. Wiedenmaier, O. Oehme, L. Schmidt, H. Luczak, "Augmented Reality (AR) for Assembly Processes Design and Experimental Evaluation", *International Journal of Human-Computer Interaction, Vol. 16, No. 3*: pp. 497-514, 2003.

[42] E. Woods, M. Billinghurst, J. Looser, G. Aldridge, D. Brown, B. Garrie, and C. Nelles, "Augmenting the science centre and museum experience", in *Proc. of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, (GRAPHITE 2004), pp. 230-236, Singapore, ACM Press, 2004.

[43] J. Zauner, M. Haller, A. Brandl, and W. Hartmann, "Authoring of a Mixed Reality Assembly Instructor for Hierarchical Structures", in *Proc. of the Second International Symposium on Mixed and Augmented Reality (ISMAR 2003)*, Tokyo, Japan, October, 2003.

[44] J. Zauner and M. Haller, "Authoring of Mixed Reality Applications including Multi-Marker Calibration for Mobile Devices", in *Proc. of 10th Eurographics Symposium on Virtual Environments (EGVE 2004)*, Grenoble, France, June 8-9, 2004, pp. 87-90.

Wolfgang Broll is the Head of the Collaborative Virtual and Augmented Environments Department at the Fraunhofer Institute for Applied Information Technology FIT. He is also a lecturer at the Aachen University of Technology (RWTH). He received his Diploma in Computer Science from the Darmstadt University of Technology in 1993 and a PhD in Computer Science from the Tübingen University in 1998. He has been doing research in the area of shared virtual environments, multi-user VR and 3D interfaces since 1993. He participated at the Esprit project COMIC and the i3 project eRENA. He was the project coordinator of the German federal research project mqube. He also was the project manager and the co-ordinator of the IST project ARTHUR. He is now concerned with multi-modal AR interfaces and advanced AR rendering issues. Wolfgang Broll was program chair of the ACM Web3D/VRML 2000 symposium and general chair of the ACM CVE 2002 conference. He is a member of ACM SIGGRAPH and a founding member of the VR/AR chapter of Germany's computer society (GI). He served on several program and conference committees including CVE 98-2002, Web3D 2001–2005 and VR 2005. He is the author of more than 45 reviewed papers and presented research at several conferences including IEEE VR and ACM SIGGRAPH.

Irma Lindt is working as a research associate at Fraunhofer FIT in St. Augustin, Germany, in the department Collaborative Virtual and Augmented Environments. She has been working on several research projects including ARTHUR (Augmented Round Table for Architecture and Urban Planning) and mqube (Mobile Multi-user Mixed Reality Environment). Currently she is coordinating the Cross-Media showcase in IPerG – an integrated EU project on Pervasive Games. Her research interests include adaptive user interfaces, interaction authoring and mobile Augmented Reality. Irma Lindt received her masters' degree in Computer Science from the Eastern Michigan University, USA.

Jan Ohlenburg is working as a research associate in the Collaborative Virtual and Augmented Environments Department at Fraunhofer FIT. After studying at University of Technology, Aachen, and a year abroad at Queen's University of Belfast, UK, he received his Diploma in Computer Science in 2003. He has been working on several research projects including ARTHUR and mqube and is currently involved in the integrated EU project IPerG. His research interests include computer graphics especially in the field of VR. Jan Ohlenburg is member of Germany's computer society (GI).

Iris Herbst is a research associate at the Collaborative Virtual and Augmented Environments Department at the Fraunhofer Institute for Applied Information Technology FIT. She has received her Diploma in Mathematics from the University of Hanover in 1999. She has worked in the area of software engineering with stress to multi-modal interfaces for several years. She joined Fraunhofer FIT in 2001 and was the local project manager of the national research project KoKoBel. She is now concerned with research and development of multi-modal and haptic user interfaces for VR and AR environments.

Michael Wittkämper is the department head assistant of the Collaborative Virtual and Augmented Environments Department at the Fraunhofer Institute for Applied Information Technology FIT. He has received his diploma (MSc) in Informatics from the University of Technology Karlsruhe. His thesis was about the simulation of complex lighting conditions on a mixed reality stage environment. He has worked in the area of stage lighting management for several years. Since he joined Fraunhofer FIT he has participated in the IST project ARTHUR and the national Mixed Reality project mqube. He is now project manager of the 6th framework project CONNECT. His research interests are advanced visualization techniques for mixing real and virtual environments.

Thomas Novotny has recently finished his diploma thesis in Media-Technology at the Ilmenau University of Technology. His thesis was concerned with the focus on user interface techniques in AR within the scope of the ARTHUR project. He has been working on AR projects at Fraunhofer FIT and on VR projects at the Düsseldorf University of Applied Sciences Düsseldorf. His research interests include user interface design and input devices for AR and VR applications.