Abstraction and Implementation Strategies for Augmented Reality Authoring

Florian Ledermann[1], István Barakonyi[2], Dieter Schmalstieg[2]

[1] Vienna University of Technology, Austria
[2] Graz University of Technology, Austria

Abstract. Augmented Reality (AR) application development is still lacking advanced authoring tools – even the simple presentation of information, which should not require any programming, is not systematically addressed by development tools. Moreover, there is also a severe lack of agreed techniques or best practices for the structuring of AR content. In this chapter we present APRIL, the Augmented Presentation and Interaction Language, an authoring platform for AR applications which provides concepts and techniques that are independent of specific applications or target hardware platforms, and should be suitable for raising the level of abstraction at which AR content creators can operate.

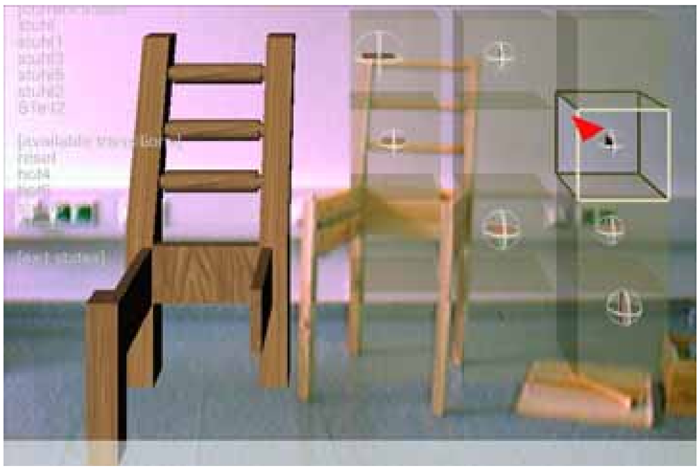


Figure 1: The interactive furniture construction guide is an example for an application which was created with APRIL by undergraduate students. The construction process is modeled with the state engine, and possible parts for the next step are shown to the user.

1 Introduction

While Augmented Reality (AR) technology is steadily maturing, application and content development for those systems is still taking place mostly at source code level. Besides limiting developer productivity, this also prevents professionals from other domains such as writers, designers or artists from taking an active role in the development of AR applications and presentations. Previous attempts to adapt authoring concepts from other domains such as virtual reality or multimedia have met with only partial success.

In order to develop authoring tools that genuinely support AR content creation, we have to look into some of the unique properties of the AR paradigm. We argue that a successful AR authoring solution must provide more than an attractive graphical user interface for an existing AR application framework. It must provide conceptual models and corresponding workflow tools which are appropriate and useful specifically within the domain of AR.

In this chapter, we explore such models and tools, and describe a working solution, the Augmented Presentation and Interaction Language, APRIL (for a typical example, see Figure 1). In particular, we discuss aspects relating to real-world interfaces, hardware abstraction, runtime engine and authoring workflow.

1.1 Requirements

1.1.1 Real-world interfaces

One aspect that makes AR setups fundamentally different from other media is the presence of the real world in the user's perception of the application space –a feature that we have to take into account when structuring application space and interaction. In our conceptual model we address the different possibilities of relating application content to the real world by borrowing from the world of theatre the concept of a *stage*. Stages are spatial containers that can be linked to places in the real world, to objects in the real world or to a user's display. For example, an object in the real world that is involved in 3D interaction must be explicitly modeled as an application object despite the fact that it will not be rendered graphically.

1.1.2 Hardware abstraction

A fundamental aspect of AR is the heterogeneity of hardware setups, devices and interaction techniques that usually prohibit a "write once, run anywhere" approach or the development of standardized interaction toolkits. We will present strategies for hardware abstraction and an interaction concept that can be transparently applied to a wide range of input devices. By applying these abstractions application portability can be improved, enabling applications to be developed on desktop workstations or in other test environments instead of directly on the target AR system, which may be scarce or expensive to use. Note that this virtualization of resources is equally useful for classic Virtual Reality (VR) development.

The primary requirement is that the framework should support the manifold combination possibilities of input and output peripherals found in the hybrid, distributed AR systems we are developing in our research. Applications and their components should be reusable in different setups, and applications developed for one system should run on another setup, with little or no modification.

This also opens up the possibility of cross-platform development and simulation. As most AR systems are prototypes, they are usually also a scarce resource. It should therefore be possible to develop applications in a desktop-based simulation environment, without monopolizing the target system during the entire development process. In some cases, such as when working with mobile systems or handheld devices, it is also much more convenient to develop the application on a desktop PC and then run it on the target system exclusively for fine-tuning and final deployment.

A simulation environment for AR development has additional advantages, such as full control over time and events. For example, video and tracking data can be played back from a file for deterministic testing, and the playback can paused and resumed at any time while debugging. Sources of input that are unavailable or malfunctioning can be substituted by synthetic input or Wizard of Oz input for early testing.

1.1.3 Runtime engine

Obviously, an authoring system cannot exist in isolation, but requires a runtime engine to execute the application and present the content created in the authoring system. Most AR systems are the product of research projects and require the use of programming languages such as C or C++. Such general purpose programming languages offer maximum flexibility for dealing with low-level time-critical issues such as tracking or graphics, but lack support for recurring high level patterns of interaction and content. In particular, the runtime engine for AR authoring must provide structuring of space and time.

Spatial organization of the content can be achieved by representing physical and virtual locations as stages, populated by real and virtual actors. Regarding the temporal organization of applications, we propose a state machine based model for flow control, leading the user through a nonlinear sequence of scenes, each containing a series of scheduled animations and property changes. Scene changes are triggered by user interaction, which can be bound to different interaction techniques depending on the execution platform. The state engine acts also as a visual representation of the application, allowing different members of the development team to collaborate within brainstorming and design sessions.

We will also examine the building blocks of applications and discuss different approaches including a component model encapsulating attributes of existing applications relevant to content authoring and interaction.

1.1.4 Authoring workflow

An authoring solution for AR is more than a pleasant user interface on top of an AR runtime engine. A useful solution for teams of professional multimedia developers is usually based on a workflow and a number of tools that support this workflow. Separation of concerns allows mixing and matching of different commercial and proprietary tools used concurrently by multiple individuals working on different aspects of the project. A complex and experimental medium such as AR typically requires lots of prototyping, and therefore the tools must support interactive and incremental work.

In our approach, we have chosen to use a declarative scripting language as the integrating component for authoring. It abstracts the low-level technical aspects of the system, and exposes high level concepts to the author in an open format, which can be created using any combination of manual text entry, graphical user interfaces or automatic generators. Rather than relying on a monolithic graphical user interface, we support third-party standard tools for multimedia content (graphics, sound, 3D modeling, video etc.) and process description (UML state charts) (Object Management Group, 2003) as a front-end to the authoring process. Users will typically spend only a small amount of time directly with the scripting language.

1.2 Contribution

The aforementioned concepts have been implemented in an authoring system called APRIL. We will present this system as a proof of concept implementation of the presented ideas, and discuss results and experiences. The contributions presented in this chapter are: (1) identification of key concepts and properties of AR systems that are relevant for content creation, (2) description of the state-of-the-art in AR authoring, (3) a consistent conceptual model for content creators covering hardware abstraction, interaction, spatial and temporal structuring, (4) presenting a reference implementation of an authoring system using the aforementioned model.

2 Related work

There is an extensive body of work on authoring tools for interactive 3D graphics, ranging from scripting languages to fully integrated graphical tools. For example, many of today's commercial computer games ship with an editor for the creation of new levels, artifacts or characters. However, authoring desktop 3D content meets only a subset of the requirements of authoring for AR. Moreover, we have to distinguish authoring solutions targeted at a specific purpose, such as a particular game or application, from general purpose authoring solutions.

2.1 3D modeling and scripting

The first attempts to support rapid prototyping for 3D graphics were based on text file formats and scripting languages such as Open Inventor (Strauss & Carey, 1992), VRML (VRML Consortium, 1997) or X3D (Web3D Consortium, 2005). New types of objects and behaviors can only be added by implementing them in C++ and compiling them to native code. While scriptable frameworks represent an improvement in the workflow of programmers, who can create application prototypes without the need to compile code, they do not offer the necessary concepts and abstractions for controlling an application's temporal structure and interactive behavior, and provide no built-in support for AR/VR devices. Platforms like Avango (Tramberend, 1999) or *Studierstube* (Schmalstieg, Fuhrmann, Hesina, Szalavari, Encarnacao, Gervautz & Purgathofer, 2002) add the necessary classes to such frameworks to support the creation of AR/VR applications. However, from the perspective of an author the power of these frameworks further complicates matters rather than providing the required level of abstraction.

Among the tools targeted towards beginners, the Alice system (Conway, Pausch, Gossweiler, & Burnette, 1994) is particularly noteworthy. It was designed as a tool to introduce novice programmers to 3D graphics programming. Alice comes with its own scene editor and an extensive set of scripting commands, but is clearly targeted at an educational setting. For creating "real world" applications, the reusability and modularity of Alice is insufficient. Also, Alice focuses on animation and behavior control of individual objects and does not offer any high-level concepts for application control.

2.2 Application specific authoring

Another set of authoring solutions provides high level support for AR/VR, but concentrates on providing optimal support for a particular domain, or within the bounds of a specific host application. One example is a system intended for presentations to a mostly passive audience. The Virtual Reality Slide Show system (VRSS) (Fuhrmann, Prikryl, Tobler & Purgathofer, 2001), provides a set of high-level concepts for authoring through a collection of Python macros. VRSS draws inspiration from conventional slide shows, and provides the concepts necessary to the user when creating similar slide shows within a VR environment.

Powerspace (Haringer & Regenbrecht, 2002) allows users to use Microsoft Powerpoint to create conventional 2D slides, which are then converted to 3D presentations by a converter script. These slide shows can be further refined in an editor that allows the adjustment of the spatial arrangement of the objects of the application, as well as the import of 3D models into the slides. Clearly, the Powerspace system is limited by the capabilities of the Powerpoint software and the slideshow concept, but it offers an interesting perspective on integrating pre-existing content into the Augmented Reality domain.

The Geist project (Kretschmer, Coors, Spierling, Grasbon, Schneider, Rojas, & Malaka, 2001) aims at the presentation of historical and cultural information for mobile AR users roaming a city. The Geist engine builds on a detailed analysis of drama theory and interactive storytelling and provides several runtime modules to support applications based on these concepts. Using Prolog, authors can create semiotic functions that drive the story. Virtual characters that are controlled by an expert system demonstrate compelling conversational and emotional behavior. While this approach is very general and powerful, it can only reveal its full potential in fairly complex applications, incorporating dynamic behavior of multiple real and virtual actors, and hence requires a correspondingly high effort in content creation. The authors do not provide details of their application examples, making it difficult to assess the final results.

The Situated Documentaries application developed at Columbia University, has similar goals for mobile AR users, but uses a more straightforward hypermedia approach towards content authoring. The hypermedia narratives are bound to locations in an outdoor environment, and can be browsed by the user by roaming the environment. The researchers have developed a custom visual editor for Situated Documentaries (Güven & Feiner, 2003), allowing them to implement an authoring paradigm tailored to the needs of their system. While the visual editor looks very promising, the underlying hypermedia system limits the scope of possible applications.

The need for an additional abstraction layer to support hybrid setups and AR specific features has been recognized by some researchers. For example, AMIRE provides a component model for authoring and playback of AR applications. On top of the AMIRE system, a graphical authoring tool for AR assembly instructions has been created (Zauner, Haller, & Brandl, 2003). However, this tool is limited to the domain of step-by-step instructions for assembly tasks. A similar approach is taken by researchers at Fraunhofer IGD (Knöpfle, Weidenhausen, Chauvigne, & Stock, 2005), who present a graphical editor for describing AR support for car maintenance procedures.

2.3 General purpose AR/VR authoring

There are few systems that support authoring (as opposed to programming) for general purpose AR/VR. An initial inspiration for the work presented in this paper is alVRed (Beckhaus, Lechner, Mostafawy, Trogemann, & Wages 2002), developed at Fraunhofer IMK. The alVRed project is an authoring solution which uses a hierarchical state machine to model the temporal structure of VR applications. In their model, a state represents a scene of the application, while the transitions between states represent changes in the application triggered by user interaction. alVRed provides a runtime engine built on top of the Avango (Tramberend, 1999) environment for the executing the state machines, as well as a number of editors for supporting various stages of content creation. Particularly interesting is an editor for fine-tuning graphics and animation parameters from within an immersive projection environment. The one area not adequately addressed by alVRed encompasses key AR requirements such as interaction abstraction and multi-user operation.

The Designers Augmented Reality Toolkit (DART) (MacIntyre, Gandy, Dow, & Bolter 2004) developed at GeorgiaTech is built on commercial software: DART extends Macromedia Director, the premier authoring tool for creating classical screen based multimedia applications. DART allows design students who are already familiar with Director to quickly create compelling AR applications, often using sketches and video based content rather than 3D models as a starting point. Director is an extremely versatile platform used by an extensive community of multimedia developers for a large variety of applications, and these properties are inherited by the DART plug-in. However, DART is ultimately limited by the technical constraints of Director, such as inadequate support for 3D models, stereoscopic rendering, optical see-through displays or multi-user applications.

Finally, the Distributed Wearable Augmented Reality Framework (DWARF), developed at Technische Universität München, deserves mentioning, although it is not strictly an authoring solution. DWARF is a strongly component-oriented middleware, composed of communicating objects. In (MacWilliams, Sandor, Wagner, Bauer, Klinker, & Bruegge 2003), the authors report on interactive development in “jam sessions”. This expression describes incremental prototyping of a running system by multiple programmers working concurrently. A graphical monitor program allows convenient inspection and remote control of the components. DWARF’s dynamic reconfiguration capabilities allow the developers to replace software components and restart parts of an application on the fly without re-starting the whole system. This is exceptional insofar as it pertains to a distributed multi-user system with full hardware abstraction capabilities rather than a single computer authoring workplace.

3 The APRIL language

APRIL, the Augmented Reality Presentation and Interaction Language, covers all aspects of AR authoring defined in the requirements analysis. APRIL provides elements to describe the hardware setup, including displays and tracking devices, as well as the content of the application and its temporal organization and interactive capabilities. Rather then developing APRIL from scratch, we built the authoring and playback facilities on top of our existing *Studierstube* runtime system. However, it should be possible to use other runtime platforms for playing back applications created using our framework.

We decided to create an XML-based language for expressing all aspects needed to create compelling interactive AR content. This language acts as the “glue-code” between those parts where we could use existing content formats. XML was chosen for three reasons: It is a widely used standard for describing structural data, allows the incorporation of other text or XML based file formats into documents, and offers a wide range of tools that operate on XML data, such as parsers, validators or XSLT, a technology to transform XML data into other document formats. Having made the choice of XML as the base technology for our content format, the next step was to design the vocabulary of our intended AR authoring language.

Enumerating all elements and features that APRIL provides is beyond the scope of this paper. Interested readers are referred to (Ledermann, 2004), where detailed information and the APRIL schema specification can be found. In this paper, we focus on the illustration of the main concepts of the APRIL language and an analysis of the implications of our approach. Whenever references to concrete APRIL element names are made, these will be set in `typewriter` letters.

3.1 Overview

The main aspects that contribute to an application are encapsulated in four top-level elements – `setup`, `cast`, `story`, `behaviors` and `interactions` – that can be easily exchanged, allowing customization of the application for different purposes (Figure 2).

The *story* is an explicit representation of the temporal structure of the application, composed of individual *scenes*. In each scene, a predefined sequence of *behaviors* is executed by *actors*, which are instances of reusable components which expose certain fields for input and output. The transitions that advance the story from one scene to the next are triggered by user interaction, potentially provided by interaction components.

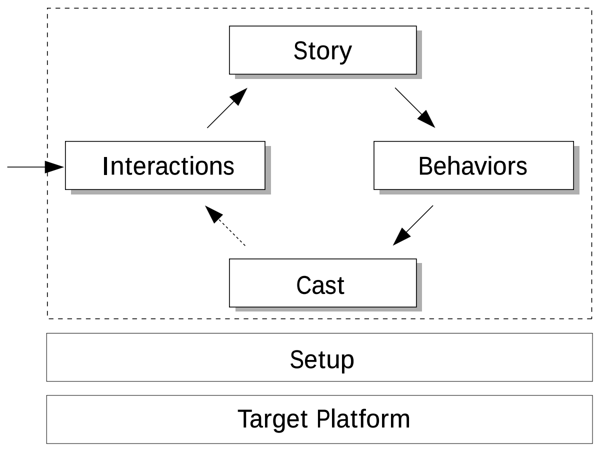


Figure 2: The main components of APRIL.

The decision to have a central storyboard controlling the application was made in full knowledge of other, agent-centric approaches, where the overall behavior of an application is the result of emergent behaviors of one or more autonomous agents. In contrast to other applications, for our AR applications we want the results to be predictable and easily controllable by a human author. Hence, a single central instance of a storyboard seemed best suited to model such an application.

The hardware setup provides a layer of abstraction that hides details of the underlying hardware setup from the user. Using different hardware description files, applications can be run on different hardware setups without changing their content.

3.2 Stages, Scenes and Actors

The two fundamental dimensions along which an application is organized have already been mentioned: temporal organization, determining the visibility and behavior of the objects of the application over time, and spatial organization, determining the location and size of these objects in relation to the viewer.

We call all objects that are subject to this organization, and therefore make up an application's content, actors. An actor may have a geometric representation, like a virtual object or a character that interacts with the user, but it could also be a sound or video clip or even some abstract entity that controls the behavior of other actors. APRIL allows nesting of actors, such that a single actor can represent a group of other actors that can be moved or otherwise controlled simultaneously. Each actor is an instance of a component that has a collection of input and output fields allowing reading and writing of typed values. Details of the APRIL component model will be explained in section 3.4.

When we refer to the behavior of an actor we mean the change of the fields of the actor over time. Aspects of the behavior can be defined in advance by the author by arranging field changes on a timeline, parts of which will be dynamic, determined by user interaction at runtime.

We decided to use UML state charts to model applications, a formalism that has been used successfully by other projects like alVRed and for which professional graphical modeling tools exist. UML state charts can be hierarchical and concurrent, meaning that a state can contain sub-states, and there might be several states active at the same time (Figure 3). Each state represents a scene in the APRIL model, and has three timelines associated with it: The `enter` timeline is guaranteed to execute when the scene is entered, the `do` timeline is executed as long as the scene remains active (this means that behaviors on that timeline are not guaranteed to be executed and can be interrupted whenever the scene is left), and the `exit` timeline, which is executed as soon as a transition to the next scene is triggered. On each of the timelines, field changes of actors can be arranged by setting or animating the field to a new value.

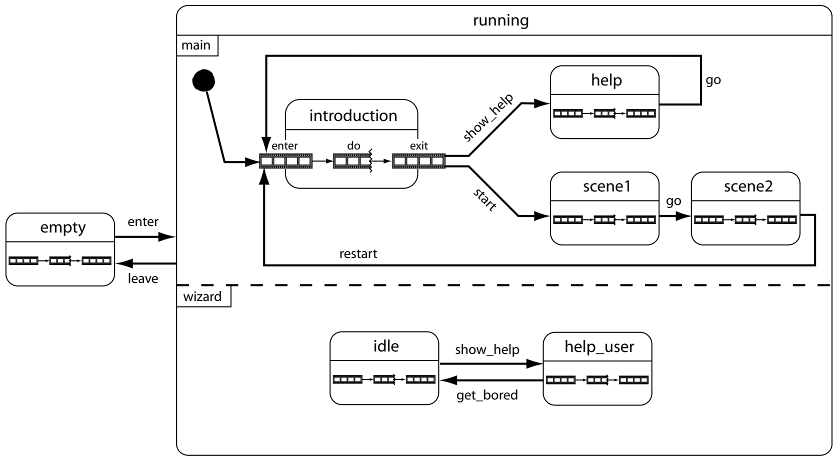


Figure 3: The storyboard of a simple APRIL application, modeled as a UML state diagram. For the "introduction" scene, the three timelines `enter`, `do` and `exit` are emphasized.

The temporal organization of applications can borrow from pre-existing concepts already used for developing virtual reality content. However, the spatial organization of content in an AR application differs from existing approaches. In VR applications, typically a single scene is rendered for all users, while one of the specific strengths of Augmented Reality systems is to provide multiple users with different views on the world. Even for a single-user AR system, there may be several "realities" that are simultaneously viewed: besides the real world and corresponding registered computer generated overlays: user interface elements, like head up displays (HUDs) or interaction panels, world in miniature (Stoakley, Conway, & Pausch 1995) for navigation or scenes rendered-to-texture for use as a 2D information display, and so on.

In APRIL, the top-level spatial containers for the content of an application are called stages. For each stage, authors can not only define spatial relationships with respect to the world and to other stages, but also the rendering technique used (e.g. three-dimensional or as a texture on a flat surface) and the association of stages with certain displays (to provide "private" content for particular users). By default, actors appear on the main stage, the area that is aligned with the real world. Interaction objects can be placed on interaction stages, where they will be rendered depending on the setup, for example as a HUD or as an interaction panel.

Stages require a coupling between the individual hardware setup and the application – the available stages are different for each setup, and therefore have to be defined in the hardware description (which will be described in section 3.3). If these stages were simply referenced in the application (e.g. by name), the portability of the application would be reduced, because the application could then only be run on similar setups with the same number of stages. To overcome this problem, the concept of roles has been introduced: each stage is assigned one (or more) roles from a predefined set that describe the function of a particular stage. These roles are tokens, for example `MAIN` would identify the world stage that is visible to all users, `UI_ALL` the user interface stage for all users, `UI1` the user interface of the primary user and so on. Actors can then be assigned to a list of stages, and the runtime system would look for the first stage on the list that is available on the target platform. If none of the substitution possibilities is available, a warning message is generated and the corresponding content is not displayed.

Another type of content specific to AR applications is the real world. Usually, for more advanced applications, some sort of world model is required, for calculating occlusions between real objects and virtual ones, or to be able to render content that is projected onto real world objects correctly. APRIL provides the `world` element as a container for geometry of the real world. The geometry can be obtained by careful modeling or by scanning the objects with a 3D-scanner.

3.3 Hardware abstraction

Flexibility in AR authoring requires separation of the content of the application from all aspects that depend on the actual system that the application will run on. However, the mapping from the hardware-dependent layer to the application must be sufficiently expressive to allow the application to make full use of the hardware features like tracking devices or displays.

APRIL allows all hardware description aspects to be placed into a separate file, and supports the running of an application on different setups each with their respective setup description files. Each of these files contains XML code that describes the arrangement of computers, displays, pointing and other interaction devices that the system is composed of, and the definition of stages and input devices that will be available in the application.

Each computer that is part of the setup is represented by a corresponding `host` element, that defines the name and IP-address of that host, and the operating system and AR platform that runs on that machine. Inside the `host` element, the displays and tracking devices that are connected to that host are specified by corresponding elements. For each display, a `display` element carries information about its size and the geometry of the virtual camera generating the image. For configuring tracking devices, we use the existing OpenTracker configuration language (Reitmayr & Schmalstieg, 2001), that is simply included in the APRIL file by using a namespace for OpenTracker elements. OpenTracker allows the definition of tracking sources and a filter graph for transforming and filtering tracking data. Rather than reinventing a similar technology, we decided to directly include the OpenTracker elements into the APRIL setup description files.

OpenTracker only defines tracking devices and their relations, but not the meaning of the tracking data for the application. In APRIL, OpenTracker elements are used inside appropriate APRIL elements to add semantics to the tracking data: `headtracking` or `displaytracking` elements inside a `display` element contain OpenTracker elements that define the tracking of the user's head or the display surface for the given display, `pointer` elements define pointing devices that are driven by the tracking data, and `station` elements define general-purpose tracking input that can be used by the application.

Pointing at objects and regions in space plays a central role in Augmented Reality applications, and several techniques have been developed to allow users to perform pointing tasks under various constraints. APRIL provides the `pointer` element to define a pointing device, allowing the author to choose from several pointing techniques. The simplest case would be a pointing device that operates in world space. Other applications have used a ray-picking technique, using a "virtual laser pointer" to select objects at a distance. Some techniques work only in combination with a display, such as performing ray-picking that originates from the eye point of the user, effectively allowing her to use 2D input to select objects in space. These pointers can only be used in conjunction with a specific display and are placed inside the corresponding `display` element.

Stages, the top-level spatial containers for the application's content, are also defined in the setup description file. A `stage` can be defined inside a `display` element, in which case the content of the stage will only be visible on that specific display. Content placed in stages that are defined at the top level of the configuration file is publicly visible for all users. As already mentioned, a stage can be assigned one or multiple roles to determine the type of content it is intended for – currently, supported roles include `MAIN` for the main content of the application, `UI` for user interface stages and `WIM` for world-in-miniature style overviews. In addition to assigning roles, for each stage it is possible to choose whether the content should be rendered in 3D or as a 2D texture, and whether it should be positioned relative to the global world coordinate system or located at a fixed offset from the display surface.

Figure 4 lists an example hardware configuration file for a single-host setup using a pointer and four stages.

```
<april XMLns="http://www.Studierstube.org/april"
       XMLns:ot="http://www.Studierstube.org/opentracker">
  <setup>
    <host name="mobile" ip="10.0.0.77" hwPlatform="Linux">
      <screen resolution="1280 1024"/>
      <screen resolution="1024 768"/>
      <display screen="1" screenSize="fullscreen" stereo="true"
        worldSize="-0.4 0.3" worldPosition="0.098 0.162 0"
        worldOrientation="-0.1856 0.9649 0.1857 1.6057" mode="AR">
          <headtracking>
            <ot:EventVirtualTransform translation="0.00 0.20 0.01">
            <ot:NetworkSource number="1" multicast-addr="10.0.0.7" port="12345"/>
            </ot:EventVirtualTransform>
          </headtracking>
        <stage role="WIM1" type="3D" location="DISPLAY" scaleToFit="true"
               translation="0 -0.5 0" scale="0.5 0.5 0.5"/>
        <stage role="UI1" type="2D" location="DISPLAY" scaleToFit="true"/>
        <pointer mode="2D-RAY"/>
      </display>
      <station id="tool">
        <ot:NetworkSource number="2" multicast-addr="10.0.0.7" port="12345"/>
      </station>
    </host>
    <stage role="WIM_COMMON" type="3D" location="WORLD" scaleToFit="true"
           translation="1.3 2.9 0.75" size="0.5 0.5 0.5"/>
    <stage role="MAIN" type="3D" location="WORLD"/>
  </setup>
</april>
```

Figure 4: Example hardware description file.

3.4 Component model

As stated in the requirements, the content of APRIL applications should be assembled from reusable components. Components should be defined outside the application, in individual files, to allow for re-use across applications and setups.

As these components constitute the content of our applications, sophisticated means to express geometry and multimedia content will be needed. Instead of creating a new XML-based syntax for defining these objects, another approach has been chosen, which is loosely inspired by the VRML97 EXTERNPROTO concept (VRML Consortium, 1997). An APRIL component is basically a template, using any existing, text based "host language" to express the intended content, plus additional XML markup to define the interface of the component, a collection of inputs and outputs that will be accessible from the APRIL application. The chosen content format must be supported by the target runtime platform. Therefore it is necessary to provide multiple implementations, sharing the same interface, in different formats to support different runtime platforms. The APRIL component mechanism itself is platform independent and can make use of any host language.

Using a platform specific language for content definition reduces portability of components, but makes all features and optimizations of a given platform available to developers. We considered creating a platform-neutral content definition language that could only use a set of features supported by all platforms, which would however preclude the creation of sophisticated content that uses state-of-the-art real time rendering features.

An APRIL component definition file consists of two main parts: the components interface definition, and one or multiple implementations. A component's interface is composed of the available input and output fields, and the specification of possible sub-components (called parts) that can be added to the component. This interface definition is shared across all implementations, and defines the features of the component that are available for scripting in APRIL.

A component can have multiple implementations in different host languages – the software used for playing the APRIL application will choose the most suitable implementation. Therefore, authors can provide different implementations for different runtime systems, for example to provide a simpler implementation to be run on handheld computers. Each implementation contains the code for implementing the component's behavior in the chosen host language, where the inputs and outputs used in the interface definition are marked with special XML marker elements, to indicate the language-specific entry points for setting and retrieving values from the component's fields. As the usage of these marker elements depends on the runtime platform that the application will be executed on, no general rules can be given for using them. Figure 5 shows a simple example component, containing the interface definition and a single implementation section for Open Inventor based frameworks (such as *Studierstube*).

```
<component id="model" XMLns="http://www.Studierstube.org/april">
  <interface>
    <field id="position" type="SFVec3f" default="0.0 0.0 0.0"/>
    <field id="visible" type="SFBool" default="TRUE"/>
    <input id="src" type="SFString" const="true"/>
    <part id="children"/>
  </interface>
  <implementation swPlatform="OpenInventor">
DEF <id/> Separator {
    DEF <id/>_Switch Switch {
        whichChild = DEF <id/>_Bool BoolOperation { # convert from Bool to Int32
            a <in id="visible"/>
            operation A
        }.output
        Group {} # Dummy Child for switching off
        Group { # actual content
            DEF <id/>_Transform Transform {
                translation <in id="position"/>
            }
            Separator {
                SoFileSubgraph { fileName <in id="src"/>}
            }
            <sub id="children"/>
        }
    }
}
    <out id="position"><id/>_Transform.translation</out>
    <out id="visible"><id/>_Bool.a</out>
  </implementation>
</component>
```

Figure 5: Definition of the "model" component, used to load geometry from an external file (Simplified

for demonstrational purposes).

3.5 Application control and interaction

As explained previously, each scene of the storyboard contains three timelines that are executed upon entering, execution and leaving the scene respectively. On these timelines, commands can be arranged to change the inputs of the application's actors. The two basic commands to change a field value are `set` and `animate`, that allow the author to set a field to a predefined value or to interpolate the value of the field over a given time span.

For more dynamic behavior of the application, the input of an actor can be `connect`ed to the output of another actor, or the `control` over a field value can be given to the user. In this case, either a pointing device can be referenced to provide the input, or a suitable user interface element is generated to control the value of the field. The connection or control possibility lasts as long as the state in which these behaviors are specified is active, so no elements for disconnecting and releasing control are needed.

The transitions between scenes are mapped to user interactions. APRIL provides built-in high-level user interactions, such as displaying a button on user interaction stages that triggers a transition when clicked (defined by the `buttonaction` element), or detecting the intersection of a pointer with the geometry of an actor (by using the `touch` element). APRIL also provides the pseudo-interactions `timeout`, `always` and `disabled`, to automatically trigger or disable certain transitions.

Customized user interaction can be realized by defining a condition that must be met to trigger the transition with the `evaluator` element. For these conditions, an output field of an actor can be compared to a constant value, or to another output. With this element, it is possible to realize complex user interactions by providing a component that encapsulates the user interface and the necessary calculations to trigger a transition.

Since all interactions are defined within the `interactions` top-level element, they can be easily exchanged. This process, called interaction mapping, can be used to derive different versions of the same application, suiting different needs. For example, a non-interactive version of an application, using only `timeout` transitions to linearly step through the application, can be provided for demo purposes, while a fully interactive version of the same application is run in user sessions.

4 Implementation

Rather than implementing a runtime platform that reads and executes APRIL files directly, we adopted a translation approach which transforms the APRIL source files into the necessary configuration files for our *Studierstube* framework, which runs on PCs and PDAs.

Because of the powerful capabilities of *Studierstube*, we can support high-level concepts with very little implementation effort for the APRIL specific runtime system (Figure 6). An implementation of a generic state-engine that controls the application at runtime according to the storyboard was implemented as an extension node, and a few utility classes were added to the *Studierstube* API. Most of the high-level concepts were however implemented by introducing a pre-processing step of the APRIL files, implemented in XSLT (Clark, 1999).

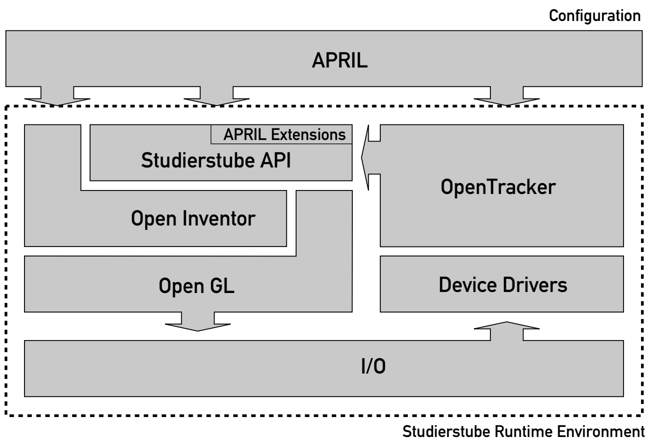


Figure 6: The overall architecture of the *Studierstube* system, using APRIL as a high-level authoring language.

XSLT is a template based language for transforming XML documents into other, text based document types. One or multiple input files can be processed in a non-linear fashion, generating arbitrary numbers of output files. XSLT is most often used to generate HTML pages from XML specification documents, or to transform and aggregate a collection of XML documents into other XML documents.

A typical *Studierstube* application consists of a number of input files – the application's content, tracking configuration, display configuration and user information are all stored in separate files, even for single user setups. One of the motivations that led to the development of APRIL is that, even in moderately complex setups, these files get quite large, and it is increasingly hard for the application developer to keep the information in the files consistent. In the APRIL preprocessing step, these files are generated by XSLT, using the information that is stored in the APRIL file in a concise way.

A schematic overview of this transformation process is shown in Figure 7. The story specification together with the corresponding interaction and behavior definitions constitutes the core of the APRIL application. Third party professional tools for content creation (for example, Maya for 3D modeling, Photoshop for image editing, and Poseidon UML for state chart editing) are employed to create the input files.

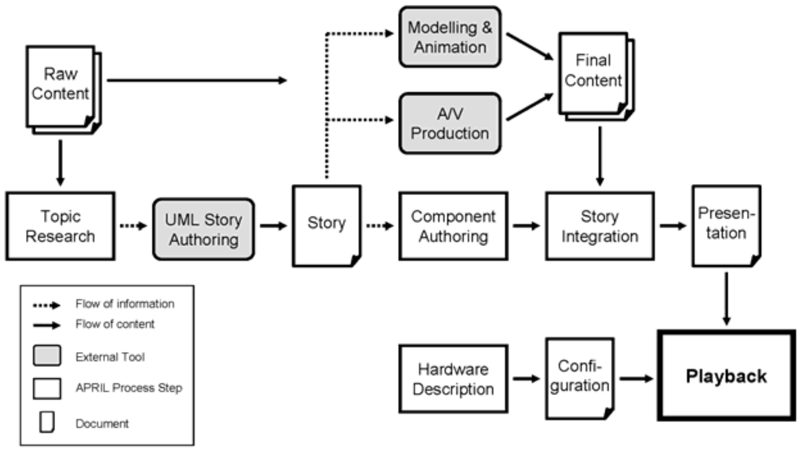
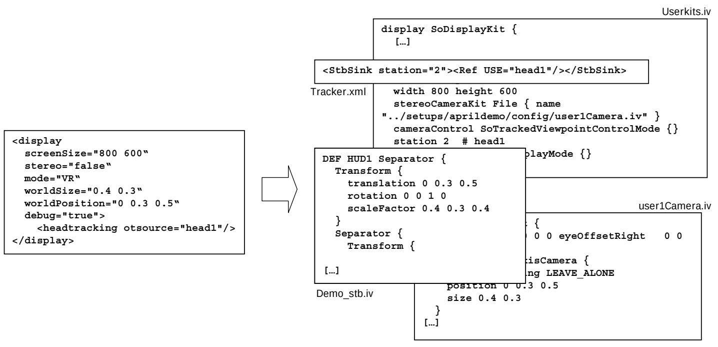


Figure 7: Schematic view of the APRIL transformation process.

Components, defined in separate files for reusability, are included in the application, and content like geometry or sound samples are included in their native file formats. At the time of the XSLT processing, the setup description file of the target platform is loaded, and the set of associated files is transformed into the necessary Open Inventor and OpenTracker files that serve as input for the *Studierstube* runtime (Figure 8).

Figure 7 also shows where human intervention in the APRIL authoring process is required. APRIL transforms the approach to AR application authoring from a technology-oriented workflow that can only be performed by programmers – implementing extensions in C++ and scripting the application logic on a low level of abstraction – to an authoring-centric view, allowing a smooth workflow and the distribution of tasks between different domain experts contributing to the application. This workflow is also much more scalable from single individuals who create an entire application to entire teams of collaborating professionals, using the storyboard as a central artifact for communication to contribute at different levels to the final result.



```
<display
  screenSize="800 600"
  stereo="false"
  mode="VR"
  worldSize="0.4 0.3"
  worldPosition="0 0.3 0.5"
  debug="true">
    <headtracking otsource="head1"/>
</display>
```

```
display SoDisplayKit {
  […]
  width 800 height 600
  stereoCameraKit File { name
"../setups/aprildemo/config/user1Camera.iv" }
  cameraControl SoTrackedViewpointControlMode {}
  station 2  # head1
                    playMode {}
```

```
<StbSink station="2"><Ref USE="head1"/></StbSink>
```

```
DEF HUD1 Separator {
  Transform {
    translation 0 0.3 0.5
    rotation 0 0 1 0
    scaleFactor 0.4 0.3 0.4
  }
  Separator {
    Transform {

 […]
```

```
                 {
                0 0 eyeOffsetRight   0 0

                isCamera {
                ng LEAVE_ALONE
    position 0 0.3 0.5
    size 0.4 0.3
  }
[…]
```

Figure 8: This figure illustrates which files in the output are affected by a single element in the hardware description file. Using conventional tools, authors would have to keep the information of these files in sync manually.

5 Results

Over the past two years we have created several applications with the help of the APRIL authoring platform. We present three examples, each illustrating important aspects of the APRIL authoring workflow. The first application shows an animated story built solely from existing APRIL components minimizing the need for customization and illustrating the power of built-in basic building blocks. The second application is a multi-user museum scenario with a complex hardware setup to facilitate rich visual elements and user interaction. The third application embeds an anthropomorphic virtual animated character into a mobile indoor navigation system. It exemplifies encapsulation of independent, complex systems and communication between them using components and fields, and shows how the same application can be run on different hardware setups simply by exchanging the setup configuration file.

5.1 Magic Book story

We have made significant effort to enable the creation of visually attractive AR applications from simple yet powerful building blocks. These building blocks offer an easy way to facilitate the rendering of various multimedia elements such as 3D models, sound and animation, and support elementary user interaction with these components using standard AR techniques and hardware setups.

The Magic Book metaphor (Billinghurst, Kato, & Poupyrev, 2001) is popular and well-known in the AR community as a basic yet attractive application, in which users browse through a real book to view animated virtual content displayed on top of fiducials printed on the pages and tracked by a camera (see Figure 9). Using built-in APRIL components such as `model` to display and control visual parameters of a 3D model, `sound` to render background music or sound effects, `canvas` to display 2D textures on a 3D image plane or `label` to display 2D text information in AR scenes allow authors to quickly compose simple sequential stories with rich multimedia content.

User interaction in the Magic Book application relies solely on predefined APRIL methods. The `buttonaction` method automatically renders a virtual button (the use of a real button is also possible) with a user-defined caption and defines a navigation point to jump to a desired scene in the presentation sequence. The `touch` method checks when a user-manipulated fiducial serving as a 3D cursor enters or exits a 3D area surrounding another marker printed on a real book page. This area is defined by the `hotspot` component. To enable the use of fiducial-based pose tracking only the configurable hardware setup description file for optical tracking needs to be completed with the appropriate marker and camera parameter information. The `control` element connects the pose of a certain marker to an application object, the `visibility` attribute of which can be modified by the `set` method depending on the current story state to show or hide it on the display.

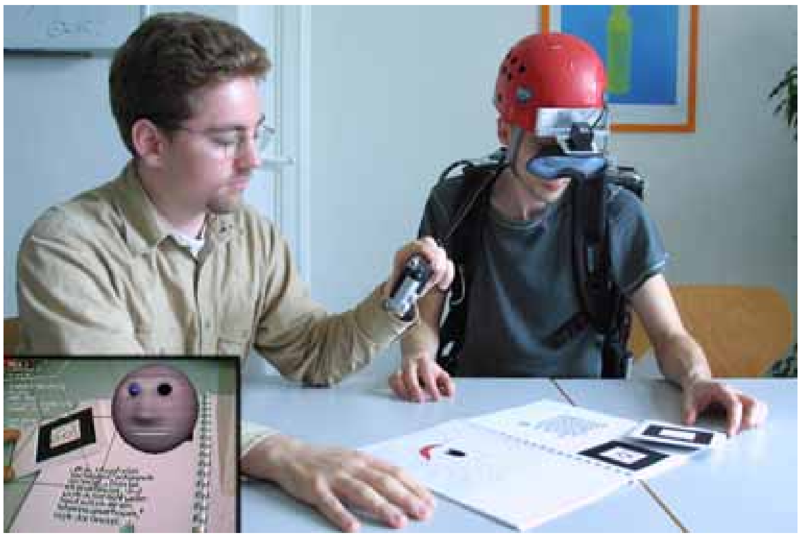


Figure 9. Two users with different AR platforms using the same application, a "Magic Book" created with the APRIL authoring toolkit.

5.2 Virtual Showcase

The Virtual Showcase has been designed to show interactive augmented reality presentations in a museum setting by overlaying virtual information on top of physical exhibition objects. User interfaces for museum visitors impose numerous requirements and constraints onto content authors: they should be intuitive and easy to handle, responsive, attractive and rich in multimedia elements, and should render a user-specific view for multiple visitors. Our demonstration application presents the architectural highlights and the historical background of the Austrian archaeological ruin Heidentor (see Figure 10).

The hardware setup of the Virtual Showcase application is prepared to render individual views for up to four users using built-in CRT monitors. The displayed virtual information is merged with a scale model of the Heidentor by half-silvered mirrors making up the walls of a pyramidical showcase. Shutter glasses, tracked by a magnetic tracking system, ensure a stereo view of virtual augmentations correctly aligned with the physical model and matching the users' viewpoint. Visitors can interact with the scene using a trackball, physical and virtual buttons and a tracked pen serving as a raypicker, and select the desired type of information such as imaginary reconstructions of the ruin, historical images or an explanatory narrative about the real model.

Although such a setup normally requires a complex combination and configuration of hardware devices and software tools, APRIL provides a simple way to compose the setup description file for Virtual Showcase. The `screen` and `display` elements allow the configuration of virtual information rendering with adjustable parameters such as resolution, stereo rendering, head tracking, raypicking device and mouse pointer. The definition of `stage` elements enables the easy integration and registration of the physical model of the Heidentor with its virtual counterpart, and the rendering of an optional head-up display to display viewpoint independent 2D information.

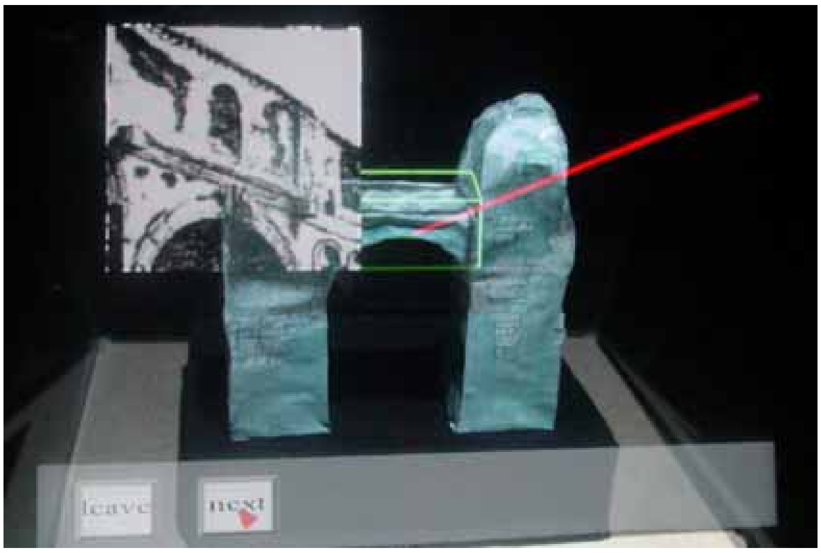


Figure 10. An archaeological ruin inside the Virtual Showcase. A raypicker is used to select parts of the real object for retrieving further information. A projector is used to cast virtual shadows onto the physical model.

5.3 Virtual Tour Guide

The Virtual Tour Guide application embeds a virtual animated character acting as a tour guide into a mobile indoor navigation application called Signpost (Reitmayr & Schmalstieg, 2003). The user wears a mobile AR setup integrated into a backpack and a helmet, and perceives the augmented world through a head-mounted display (HMD). A camera mounted above the HMD tracks fiducials placed onto walls of the building area covered by the application. The markers help locate the user within this area since the system knows their exact position in a precisely measured virtual model of the building that has been registered with its real counterpart.

The virtual tour guide character is placed into the reference frame of the real building (Figure 12). While walking around, the character provides assistance to find selected destinations and provides location-specific explanation about the content of various rooms and people working in them using body gestures (e.g. looking towards, pointing, asking the user to follow, etc.), 2D and 3D visual elements and sound. Since the tour guide is aware of the building geometry, it appears to walk up real stairs and go through real doors and walkways, thus further enhancing integrity with the user's physical environment.

The tour guide character is controlled by the AR Puppet framework that is a hierarchical character animation system enabling the use of embodied animated agents in AR applications.

Both AR Puppet and the Signpost navigation application are large systems with a complex network of internal modules responsible for various subtasks, therefore it is difficult and undesirable to modify their internal structure in order to establish communication between them. Relying on the APRIL framework's component model and turning AR Puppet and Signpost into custom APRIL components enables the encapsulation of these frameworks' functionality. These components can be used as black boxes that expose relevant input and output fields for communication with other, external components while hiding internal implementation details. Figure 11 illustrates the fields exposed by Signpost and monitored by AR Puppet to provide the tour guide character with relevant navigation information.

The expensive and bulky mobile AR system required by the Signpost application in its original form makes content and application authoring, debugging and testing a difficult task, therefore we needed to develop a desktop simulator system that is able to run the same navigation application with simple keyboard input and screen based output. The hardware abstraction feature of APRIL conveniently hides details such as the type of display or exact tracking setup from authors and components. Only symbolic names are used that allows exchanging the internal implementation of the hardware setup. See Figure 12 for an illustration of the Virtual Tour Guide application running on the desktop simulator and the mobile AR system.

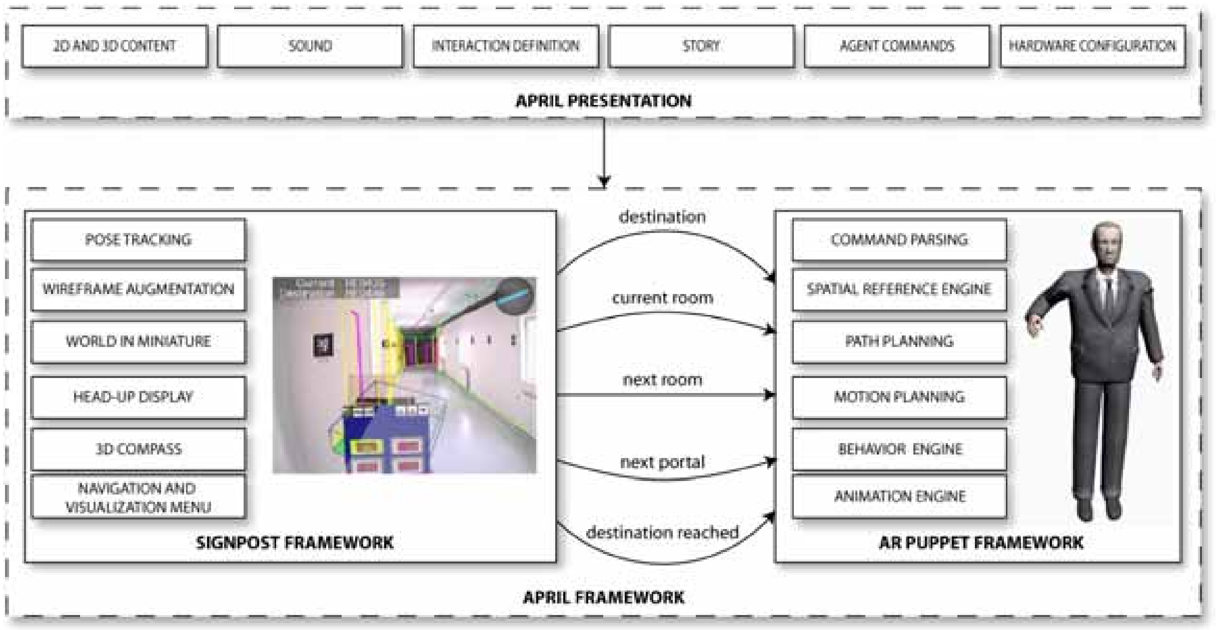


Figure 11. Communication between the AR Puppet and Signpost systems within the APRIL framework.

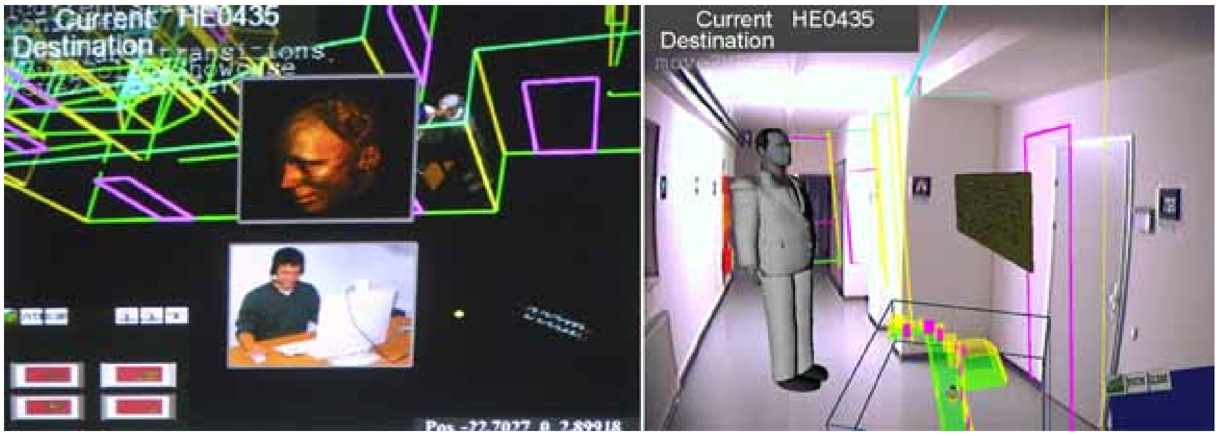


Figure 12. a) The indoor tour guide application running on the desktop developer setup. b) The indoor tour guide application view captured from the HMD of a user wearing the mobile AR backpack system. In both setups a world-in-miniature view of the building model is shown and location-dependent HUD overlay graphics is presented to the user as she roams the building.

6 Conclusions and future work

As pointed out by (MacIntyre, Bolter, Moreno, & Hannigan 2001) a new medium such as AR requires effort beyond overcoming technical difficulties before it is truly understood and becomes established. One such effort is identifying and addressing the requirements that make AR unique and that must be targeted in order to make an AR specific authoring solution successful. We have identified real-world interfaces, hardware abstraction, runtime engine and authoring workflow tools as the most prominent requirements.

Previous work has only partially addressed these requirements, or has been limited to a specific application domain or software platform. APRIL is also limited in some respects, in particular it completely relies on third party graphical editors for compiling the content, and its translation approach only allows limited support for runtime reconfigurability. However, when applied to a number of application design problems, we have found that it combines the high level authoring concepts also found in DART and alVRed in a very comprehensive way with the technical virtues of flexible component systems such as DWARF and AMIRE. It can therefore cater to a wide audience ranging from artists to engineers, and provides good support for multi-disciplinary teams.

The XSLT-based reference implementation, using the *Studierstube* framework, provides templates for the implementation of the APRIL features. We found that these templates not only provide a working implementation of APRIL, but implicitly document best practices for implementing common AR design patterns on top of that framework.

Having developed a description language and runtime engine, the next step is to concentrate on front-end tools for the interactive visual creation of AR applications. It will be challenging to examine whether such a tool proves more productive than using a conventional 2D user interfaces or direct manipulation in AR in a style similar to (Lee, Nelles, Billinghurst, & Kim 2004).

Another interesting perspective is the automatic generation of APRIL content. While previously this required detailed knowledge of the target application framework to be able to create the complex, interdependent files necessary for a non-trivial application, APRIL provides the high level of abstraction that allows the content for an application to be auto-generated by software. This opens up possibilities of using large amounts of existing content (e.g. in museums) in a Augmented Reality context with little manual effort.

7 Acknowledgements

The authors would like to thank Daniel Wagner for his AR framework for handheld devices, Gerhard Reitmayr for valuable suggestions throughout the development of APRIL and Joe Newman for proofreading this chapter. Licenses of the Maya software were donated by Alias Systems. This research was funded in part by EU contract #IST-2000-28610, FWF contract #Y193 and bm:bwk contract #TUWP16/2001.

8 References

Beckhaus, S., Lechner, A., Mostafawy, S., Trogemann, G., & Wages, R (2002). alVRed - Tools for storytelling in virtual environments. Internationale Statustagung Virtuelle und

Erweiterete Realität, Leipzig, Germany.

Billinghurst, M., Kato, H., & Poupyrev, I. (2001). The Magic-Book - Moving Seamlessly between Reality and Virtuality IEEE Computer Graphics and Applications, vol. 21(3), 6-8.

Bimber, O., Fröhlich, B., Schmalstieg, & D., Encarnacao, L. M. (2001). The virtual showcase. IEEE Computer Graphics and Applications, 21(6):48-55.

Clark J (1999). XSL transformations (XSLT) version 1.0 - W3C recommendation. http://www.w3.org/TR/xslt.

Conway, M., Pausch, R., Gossweiler, R., & Burnette, T.(1994). Alice: A rapid prototyping system for building virtual environments. Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems, volume 2, 295-296.

Fuhrmann A., Prikryl, J., Tobler, R., & Purgathofer, W. (2001). Interactive content for presentations in virtual reality. Proceedings of the ACM Symposium on Virtual Reality Software & Technology.

Güven, S., & Feiner, S. (2003). Authoring 3D hypermedia for wearable augmented and virtual reality. Proceedings of the 7th International Symposium on Wearable Computers, pages 118-126, White Plains, NY.

Haringer, M., & Regenbrecht, H. (2002). A pragmatic approach to Augmented Reality authoringProceedings of ISMAR 2002, Darmstadt, Germany.

Knöpfle, C., Weidenhausen, J., Chauvigne, L., & Stock, I. (2005). Template based authoring for AR based service scenarios. Proceedings of IEEE Virtual Reality 2005, 237-240, Bonn, Germany.

Kretschmer, U., Coors, V., Spierling, U., Grasbon, D., Schneider, K., Rojas, I., & Malaka, R. (2001). Meeting the spirit of history. Proceedings of VAST 2001, Athens, Greece.

Ledermann, F. (2004). An authoring framework for augmented reality presentations. Master's thesis, Vienna University of Technology.

Lee, G., Nelles, C., Billinghurst, M., & Kim, G. (2004). Immersive Authoring of Tangible Augmented Reality Applications. In Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality 2004 (ISMAR 2004), Arlington, USA.

MacIntyre, B., Bolter, J., Moreno, E., & Hannigan, B. (2001). Augmented Reality as a New Media Experience. In Proceedings International Symposium on Augmented Reality (ISAR 2001), New York, USA.

MacIntyre, B., Gandy, M., Dow, S., & Bolter, J. (2004). DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences. Proceedings of User Interface Software and Technology (UIST'04), Sante Fe, New Mexico.

MacWilliams, A., Sandor, C., Wagner, M., Bauer, M., Klinker, G., & Bruegge, B. (2003). Herding Sheep: Live System Development for Distributed Augmented Reality. Proceedings ISMAR 2003, Tokyo, Japan.

Object Management Group (2003). Unified modeling language (UML), version 1.5. http://www.omg.org/technology/documents/formal/uml.htm

Reitmayr G., & Schmalstieg, D. (2001). OpenTracker - an open software architecture for reconfigurable tracking based on XML. Proceedings of IEEE Virtual Reality 2001, pages 285-286, Yokohama, Japan.

Reitmayr, G. & Schmalstieg, D. (2003): Location Based Applications for Mobile Augmented Reality. Proceedings of the 4th Australasian User Interface Conference, pp. 65-73, Adelaide, Australia.

Schmalstieg D., Fuhrmann, A., Hesina, G., Szalavari, Zs., Encarnacao, L. M., Gervautz, M., & Purgathofer, W.¸ The Studierstube augmented reality project. PRESENCE - Teleoperators and Virtual Environments, 11(1).

Stoakley, R., Conway, M., & Pausch, R. (1995). Virtual reality on a WIM: interactive worlds in miniature. Proceedings on human factors in computing systems, 265-272, Denver, USA.

Strauss, P., & Carey, R. (1992). An object oriented 3D graphics toolkit. Proceedings of

SIGGRAPH '92.

Tramberend H (1999). Avocado: A distributed virtual reality framework. Proceedings of IEEE Virtual Reality 1999.

VRML Consortium (1997). VRML97 specification. Specification 147721:1997, ISO/IEC.

Web3D Consortium (2005). X3D specification website. http://www.web3d.org/x3d/specifications/.

Zauner, J., Haller, M., & Brandl, A.. Authoring of a mixed reality assembly instructor for hierarchical structures. Proceedings of ISMAR 2003, 237-246, Tokyo, Japan, October 7-10.